

**MUIbase**

**COLLABORATORS**

	<i>TITLE :</i> MUIbase		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		January 23, 2025	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>MUIbase</b>	<b>1</b>
1.1	MUIbase	1
1.2	MUIbase/Copying	2
1.3	MUIbase/Registration	2
1.4	MUIbase/Distribution	4
1.5	MUIbase/Disclaimer	5
1.6	MUIbase/Third party material	5
1.7	MUIbase/Welcome to MUIbase	7
1.8	MUIbase/Getting started	8
1.9	MUIbase/Tutorial	10
1.10	MUIbase/Basic concepts	20
1.11	MUIbase/Projects	20
1.12	MUIbase/Tables	20
1.13	MUIbase/Records (concept)	21
1.14	MUIbase/Attributes	22
1.15	MUIbase/Attribute types	22
1.16	MUIbase/String type	22
1.17	MUIbase/Integer type	23
1.18	MUIbase/Real type	23
1.19	MUIbase/Bool type	24
1.20	MUIbase/Choice type	24
1.21	MUIbase/Date type	24
1.22	MUIbase/Time type	24
1.23	MUIbase/Memo type	25
1.24	MUIbase/Reference	25
1.25	MUIbase/Virtual	26
1.26	MUIbase/Button	26
1.27	MUIbase/Table of attribute types	26
1.28	MUIbase/Memory consumption	27
1.29	MUIbase/Relationships	28

---

1.30 MUIbase/One to one relationships . . . . .	28
1.31 MUIbase/One to many relationships . . . . .	29
1.32 MUIbase/Many to many relationships . . . . .	29
1.33 MUIbase/User interface . . . . .	31
1.34 MUIbase/Windows . . . . .	31
1.35 MUIbase/Masks . . . . .	32
1.36 MUIbase/Panels . . . . .	32
1.37 MUIbase/Attribute objects . . . . .	32
1.38 MUIbase/Text objects . . . . .	33
1.39 MUIbase/Images . . . . .	33
1.40 MUIbase/Space objects . . . . .	33
1.41 MUIbase/Groups . . . . .	33
1.42 MUIbase/Balance objects . . . . .	34
1.43 MUIbase/Register groups . . . . .	34
1.44 MUIbase/Managing projects . . . . .	34
1.45 MUIbase/File format . . . . .	35
1.46 MUIbase/Info . . . . .	35
1.47 MUIbase/Clear project . . . . .	35
1.48 MUIbase/Open project . . . . .	36
1.49 MUIbase/Save project . . . . .	36
1.50 MUIbase/Delete project . . . . .	37
1.51 MUIbase/Close project . . . . .	37
1.52 MUIbase/Check data integrity . . . . .	38
1.53 MUIbase/Swap records . . . . .	38
1.54 MUIbase/Preferences . . . . .	39
1.55 MUIbase/Record memory . . . . .	40
1.56 MUIbase/Record delete requester . . . . .	40
1.57 MUIbase/External editor for programming . . . . .	40
1.58 MUIbase/Icon creation . . . . .	41
1.59 MUIbase/Formats . . . . .	41
1.60 MUIbase/Icon tool name . . . . .	42
1.61 MUIbase/External editor . . . . .	42
1.62 MUIbase/External viewer . . . . .	42
1.63 MUIbase/Popups in cycle chain . . . . .	42
1.64 MUIbase/Confirm save & reorg . . . . .	43
1.65 MUIbase/Confirm quit . . . . .	43
1.66 MUIbase/Program include directory . . . . .	43
1.67 MUIbase/Program debug information . . . . .	44
1.68 MUIbase/Program output file . . . . .	44

---

---

1.69 MUIbase/Project dependent settings . . . . .	45
1.70 MUIbase/MUI . . . . .	45
1.71 MUIbase/Load and save preferences . . . . .	46
1.72 MUIbase/Empty display image . . . . .	47
1.73 MUIbase/Record-editing . . . . .	47
1.74 MUIbase/Active object . . . . .	47
1.75 MUIbase/Adding records . . . . .	48
1.76 MUIbase/Changing records . . . . .	48
1.77 MUIbase/Deleting records . . . . .	50
1.78 MUIbase/Browsing records . . . . .	51
1.79 MUIbase/Filter . . . . .	51
1.80 MUIbase/Record filter . . . . .	52
1.81 MUIbase/Filter expression . . . . .	52
1.82 MUIbase/Changing filters . . . . .	52
1.83 MUIbase/Filter examples . . . . .	53
1.84 MUIbase/Reference filter . . . . .	54
1.85 MUIbase/Order . . . . .	54
1.86 MUIbase/Empty order . . . . .	55
1.87 MUIbase/Order by attributes . . . . .	55
1.88 MUIbase/Order by a function . . . . .	56
1.89 MUIbase/Changing orders . . . . .	56
1.90 MUIbase/Reorder all records . . . . .	57
1.91 MUIbase/Search for . . . . .	58
1.92 MUIbase/Search requester . . . . .	58
1.93 MUIbase/Forward-backward search . . . . .	59
1.94 MUIbase/Search pattern examples . . . . .	59
1.95 MUIbase/Import and Export . . . . .	59
1.96 MUIbase/Import file format . . . . .	60
1.97 MUIbase/Sample import file . . . . .	60
1.98 MUIbase/Importing records . . . . .	61
1.99 MUIbase/Exporting records . . . . .	62
1.100 MUIbase/Data retrieval . . . . .	62
1.101 MUIbase/Select-from-where queries . . . . .	63
1.102 MUIbase/Query editor . . . . .	64
1.103 MUIbase/Printing queries . . . . .	65
1.104 MUIbase/Query examples . . . . .	66
1.105 MUIbase/Structure editor . . . . .	67
1.106 MUIbase/Table management . . . . .	68
1.107 MUIbase/Creating tables . . . . .	68

---

---

1.108MUIbase/Changing tables . . . . .	69
1.109MUIbase/Deleting tables . . . . .	70
1.110MUIbase/Sorting tables . . . . .	70
1.111MUIbase/Attribute management . . . . .	70
1.112MUIbase/Creating attributes . . . . .	71
1.113MUIbase/Type specific settings . . . . .	71
1.114MUIbase/Label editor . . . . .	73
1.115MUIbase/Copying attributes . . . . .	73
1.116MUIbase/Changing attributes . . . . .	74
1.117MUIbase/Deleting attributes . . . . .	74
1.118MUIbase/Sorting attributes . . . . .	75
1.119MUIbase/Display management . . . . .	75
1.120MUIbase/Display field . . . . .	75
1.121MUIbase/Panel editor . . . . .	77
1.122MUIbase/Attribute object editor . . . . .	78
1.123MUIbase/Text editor . . . . .	81
1.124MUIbase/Image editor . . . . .	82
1.125MUIbase/Space editor . . . . .	82
1.126MUIbase/Group editor . . . . .	83
1.127MUIbase/Register group editor . . . . .	83
1.128MUIbase/Window editor . . . . .	84
1.129MUIbase/Print structure . . . . .	85
1.130MUIbase/Programming MUIbase . . . . .	85
1.131MUIbase/Program editor . . . . .	86
1.132MUIbase/Preprocessing . . . . .	86
1.133MUIbase/#define . . . . .	87
1.134MUIbase/#undef . . . . .	88
1.135MUIbase/#include . . . . .	88
1.136MUIbase/#if . . . . .	88
1.137MUIbase/#ifdef . . . . .	89
1.138MUIbase/#ifndef . . . . .	89
1.139MUIbase/#elif . . . . .	89
1.140MUIbase/#else . . . . .	90
1.141MUIbase/#endif . . . . .	90
1.142MUIbase/Programming language . . . . .	90
1.143MUIbase/Why lisp? . . . . .	91
1.144MUIbase/Lisp syntax . . . . .	91
1.145MUIbase/Kinds of programs . . . . .	92
1.146MUIbase/Name conventions . . . . .	92

---

---

1.147MUIbase/Accessing record contents . . . . .	93
1.148MUIbase/Data types for programming . . . . .	94
1.149MUIbase/Constants . . . . .	94
1.150MUIbase/Command syntax . . . . .	95
1.151MUIbase/Defining commands . . . . .	96
1.152MUIbase/DEFUN . . . . .	96
1.153MUIbase/DEFUN* . . . . .	97
1.154MUIbase/DEFVAR . . . . .	97
1.155MUIbase/Program control functions . . . . .	98
1.156MUIbase/PROGN . . . . .	98
1.157MUIbase/PROG1 . . . . .	99
1.158MUIbase/LET . . . . .	99
1.159MUIbase/SETQ . . . . .	100
1.160MUIbase/SETQ* . . . . .	100
1.161MUIbase/FUNCALL . . . . .	101
1.162MUIbase/IF . . . . .	101
1.163MUIbase/CASE . . . . .	101
1.164MUIbase/COND . . . . .	102
1.165MUIbase/DOTIMES . . . . .	103
1.166MUIbase/DOLIST . . . . .	103
1.167MUIbase/DO . . . . .	104
1.168MUIbase/FOR ALL . . . . .	105
1.169MUIbase/NEXT . . . . .	106
1.170MUIbase/EXIT . . . . .	106
1.171MUIbase/RETURN . . . . .	106
1.172MUIbase/HALT . . . . .	107
1.173MUIbase/ERROR . . . . .	107
1.174MUIbase/Type predicates . . . . .	108
1.175MUIbase/Type conversion functions . . . . .	108
1.176MUIbase/STR . . . . .	109
1.177MUIbase/MEMO . . . . .	110
1.178MUIbase/INT . . . . .	110
1.179MUIbase/REAL . . . . .	111
1.180MUIbase/DATE . . . . .	111
1.181MUIbase/TIME . . . . .	112
1.182MUIbase/Boolean functions . . . . .	112
1.183MUIbase/AND . . . . .	113
1.184MUIbase/OR . . . . .	113
1.185MUIbase/NOT . . . . .	113

---

---

1.186MUIbase/Comparison functions . . . . .	114
1.187MUIbase/Relational operators . . . . .	114
1.188MUIbase/CMP . . . . .	115
1.189MUIbase/CMP* . . . . .	115
1.190MUIbase/Mathematical functions . . . . .	115
1.191MUIbase/add . . . . .	116
1.192MUIbase/sub . . . . .	117
1.193MUIbase/1+ . . . . .	117
1.194MUIbase/1- . . . . .	117
1.195MUIbase/mul . . . . .	118
1.196MUIbase/fdiv . . . . .	118
1.197MUIbase/DIV . . . . .	118
1.198MUIbase/MOD . . . . .	119
1.199MUIbase/MAX . . . . .	119
1.200MUIbase/MIN . . . . .	119
1.201MUIbase/ABS . . . . .	119
1.202MUIbase/TRUNC . . . . .	120
1.203MUIbase/ROUND . . . . .	120
1.204MUIbase/RANDOM . . . . .	120
1.205MUIbase/String functions . . . . .	121
1.206MUIbase/LEN . . . . .	122
1.207MUIbase/LEFTSTR . . . . .	122
1.208MUIbase/RIGHTSTR . . . . .	122
1.209MUIbase/MIDSTR . . . . .	123
1.210MUIbase/SETMIDSTR . . . . .	123
1.211MUIbase/INSMIDSTR . . . . .	123
1.212MUIbase/INDEXSTR . . . . .	124
1.213MUIbase/INDEXSTR* . . . . .	124
1.214MUIbase/INDEXBRK . . . . .	124
1.215MUIbase/INDEXBRK* . . . . .	125
1.216MUIbase/RINDEXSTR . . . . .	125
1.217MUIbase/RINDEXSTR* . . . . .	125
1.218MUIbase/RINDEXBRK . . . . .	126
1.219MUIbase/RINDEXBRK* . . . . .	126
1.220MUIbase/REPLACESTR . . . . .	126
1.221MUIbase/REMCHARS . . . . .	126
1.222MUIbase/TRIMSTR . . . . .	127
1.223MUIbase/WORD . . . . .	127
1.224MUIbase/WORDS . . . . .	128

---

---

1.225MUIbase/CONCAT . . . . .	128
1.226MUIbase/CONCAT2 . . . . .	128
1.227MUIbase/COPYSTR . . . . .	129
1.228MUIbase/UPPER . . . . .	129
1.229MUIbase/LOWER . . . . .	129
1.230MUIbase/ASC . . . . .	130
1.231MUIbase/CHR . . . . .	130
1.232MUIbase/LIKE . . . . .	131
1.233MUIbase/SPRINTF . . . . .	131
1.234MUIbase/Memo functions . . . . .	134
1.235MUIbase/LINE . . . . .	134
1.236MUIbase/LINES . . . . .	134
1.237MUIbase/MEMOTOLIST . . . . .	134
1.238MUIbase/LISTTOMEMO . . . . .	135
1.239MUIbase/FILLMEMO . . . . .	135
1.240MUIbase/FORMATMEMO . . . . .	136
1.241MUIbase/INDENTMEMO . . . . .	136
1.242MUIbase/List functions . . . . .	136
1.243MUIbase/CONS . . . . .	137
1.244MUIbase/LIST . . . . .	137
1.245MUIbase/LENGTH . . . . .	138
1.246MUIbase/FIRST . . . . .	138
1.247MUIbase/REST . . . . .	138
1.248MUIbase/LAST . . . . .	139
1.249MUIbase/NTH . . . . .	139
1.250MUIbase/APPEND . . . . .	139
1.251MUIbase/REVERSE . . . . .	139
1.252MUIbase/MAPFIRST . . . . .	140
1.253MUIbase/SORTLIST . . . . .	140
1.254MUIbase/SORTLISTGT . . . . .	141
1.255MUIbase/Input requesting functions . . . . .	141
1.256MUIbase/ASKFILE . . . . .	142
1.257MUIbase/ASKDIR . . . . .	142
1.258MUIbase/ASKSTR . . . . .	142
1.259MUIbase/ASKINT . . . . .	143
1.260MUIbase/ASKCHOICE . . . . .	143
1.261MUIbase/ASKCHOICESTR . . . . .	144
1.262MUIbase/ASKOPTIONS . . . . .	145
1.263MUIbase/ASKBUTTON . . . . .	145

---

---

1.264MUIbase/ASKMULTI . . . . .	146
1.265MUIbase/I-O functions . . . . .	148
1.266MUIbase/FOPEN . . . . .	148
1.267MUIbase/FCLOSE . . . . .	149
1.268MUIbase/stdout . . . . .	149
1.269MUIbase/PRINT . . . . .	149
1.270MUIbase/PRINTF . . . . .	150
1.271MUIbase/FPRINTF . . . . .	150
1.272MUIbase/FERROR . . . . .	150
1.273MUIbase/FEOF . . . . .	151
1.274MUIbase/FSEEK . . . . .	151
1.275MUIbase/FTELL . . . . .	152
1.276MUIbase/FGETCHAR . . . . .	152
1.277MUIbase/FGETCHARS . . . . .	152
1.278MUIbase/FGETSTR . . . . .	153
1.279MUIbase/FGETMEMO . . . . .	153
1.280MUIbase/FPUTCHAR . . . . .	153
1.281MUIbase/FPUTSTR . . . . .	154
1.282MUIbase/FPUTMEMO . . . . .	154
1.283MUIbase/FFLUSH . . . . .	154
1.284MUIbase/Record functions . . . . .	154
1.285MUIbase/NEW . . . . .	155
1.286MUIbase/NEW* . . . . .	155
1.287MUIbase/DELETE . . . . .	156
1.288MUIbase/DELETE* . . . . .	156
1.289MUIbase/DELETEALL . . . . .	157
1.290MUIbase/GETMATCHFILTER . . . . .	157
1.291MUIbase/SETMATCHFILTER . . . . .	158
1.292MUIbase/GETISSORTED . . . . .	158
1.293MUIbase/SETISSORTED . . . . .	158
1.294MUIbase/RECNUM . . . . .	159
1.295MUIbase/COPYREC . . . . .	159
1.296MUIbase/Attribute functions . . . . .	160
1.297MUIbase/ATTRNAME . . . . .	160
1.298MUIbase/MAXLEN . . . . .	160
1.299MUIbase/GETLABELS . . . . .	161
1.300MUIbase/SETLABELS . . . . .	161
1.301MUIbase/Table functions . . . . .	162
1.302MUIbase/TABLENAME . . . . .	162

---

---

1.303MUIbase/GETORDERSTR . . . . .	162
1.304MUIbase/SETORDERSTR . . . . .	163
1.305MUIbase/REORDER . . . . .	164
1.306MUIbase/REORDERALL . . . . .	164
1.307MUIbase/GETFILTERACTIVE . . . . .	165
1.308MUIbase/SETFILTERACTIVE . . . . .	165
1.309MUIbase/GETFILTERSTR . . . . .	165
1.310MUIbase/SETFILTERSTR . . . . .	166
1.311MUIbase/RECORDS . . . . .	166
1.312MUIbase/RECORD . . . . .	167
1.313MUIbase/SELECT . . . . .	167
1.314MUIbase/Gui functions . . . . .	168
1.315MUIbase/SETCURSOR . . . . .	169
1.316MUIbase/GETDISABLED . . . . .	169
1.317MUIbase/SETDISABLED . . . . .	169
1.318MUIbase/GETWINDOWDISABLED . . . . .	169
1.319MUIbase/SETWINDOWDISABLED . . . . .	170
1.320MUIbase/GETWINDOWOPEN . . . . .	170
1.321MUIbase/SETWINDOWOPEN . . . . .	171
1.322MUIbase/Project functions . . . . .	171
1.323MUIbase/PROJECTNAME . . . . .	171
1.324MUIbase/CHANGES . . . . .	171
1.325MUIbase/System functions . . . . .	172
1.326MUIbase/EDIT . . . . .	172
1.327MUIbase/EDIT* . . . . .	173
1.328MUIbase/VIEW . . . . .	173
1.329MUIbase/VIEW* . . . . .	173
1.330MUIbase/SYSTEM . . . . .	174
1.331MUIbase/STAT . . . . .	174
1.332MUIbase/TACKON . . . . .	174
1.333MUIbase/FILENAME . . . . .	175
1.334MUIbase/DIRNAME . . . . .	175
1.335MUIbase/TODAY . . . . .	175
1.336MUIbase/NOW . . . . .	176
1.337MUIbase/MESSAGE . . . . .	176
1.338MUIbase/GC . . . . .	176
1.339MUIbase/Pre-defined variables . . . . .	177
1.340MUIbase/Pre-defined constants . . . . .	177
1.341MUIbase/Functional parameters . . . . .	178

---

---

1.342MUIbase/Type specifiers . . . . .	178
1.343MUIbase/Semantics of expressions . . . . .	180
1.344MUIbase/Function triggering . . . . .	180
1.345MUIbase/onOpen . . . . .	181
1.346MUIbase/onClose . . . . .	181
1.347MUIbase/onChange . . . . .	182
1.348MUIbase/New trigger . . . . .	182
1.349MUIbase/Delete trigger . . . . .	183
1.350MUIbase/Comparison function . . . . .	183
1.351MUIbase/Attribute trigger . . . . .	185
1.352MUIbase/Programming virtual attributes . . . . .	185
1.353MUIbase/ABCConvert . . . . .	186
1.354MUIbase/Menus . . . . .	187
1.355MUIbase/Acknowledgments . . . . .	188
1.356MUIbase/Author . . . . .	189
1.357MUIbase/Function index . . . . .	190
1.358MUIbase/Concept index . . . . .	193

---

# Chapter 1

## MUIbase

### 1.1 MUIbase

MUIbase Version 1.5

\*\*\*\*\*

MUIbase is a relational programmable database that uses MUI as user interface.

Copying	Your rights.
Welcome to MUIbase	MUIbase in brief.
Getting started	How to install & start MUIbase.
Tutorial	First steps for learning MUIbase.
Basic concepts	Some elementary definitions.
Managing projects	How to organize your projects.
Preferences	Customizing MUIbase.
Record-editing	Record manipulations and browsing.
Filter	Filtering records.
Order	Ordering records.
Search for	How to search for a record.
Import and Export	Data from/to other databases.
Data retrieval	How to query data from a project.
Structure editor	Setting up your own database.
Programming MUIbase	How to write MUIbase programs.

#### Appendix

ABConvert	Utility to convert projects from AmigaBase.
Menus	Description of all menu items.
Acknowledgments	People the author wants to thank.
Author	Who you should contact.
Function index	Programming functions and variables.
Concept index	Concepts.

(C) 1998-2000 Steffen Gutmann

---

## 1.2 MUIbase/Copying

MUIbase Copying Conditions

\*\*\*\*\*

MUIbase is (C) 1998-2000 Steffen Gutmann. All rights reserved.

MUIbase is not public domain or free software. If you use MUIbase then you have to register your copy after a short while. You are allowed to install and run MUIbase without registration for a period of 4 weeks. After that time you have to register your copy of MUIbase or delete it from your system. Reinstalling MUIbase does not extend the license.

After registration you receive your personal MUIbase key-file which enables all crippled features of MUIbase.

Reverse engineering of the MUIbase software protection is strictly forbidden.

Registration	How to register your copy.
Distribution	Giving MUIbase to others.
Disclaimer	This Software is "as is".
Third party material	External stuff MUIbase uses.

## 1.3 MUIbase/Registration

Registration

=====

The unregistered MUIbase version is crippled in several ways:

- \* Only up to 5 tables per project.
- \* Only up to 10 attributes per table.
- \* Only up to 30 function definitions for programming per project.
- \* No preprocessing for programming (#-directives).
- \* No remembering of window dimensions in projects.
- \* No images for structure editor.
- \* No register groups for structure editor.

In the author's opinion the absence of these features doesn't make MUIbase unusable. You should be able to see the power of MUIbase within the 4 weeks testing period without these features.

After registration you will receive a key-file which enables the disabled features. The key-file is expected to work with all future

versions of MUIbase.

#### Registration fee

-----

The registration fee for MUIbase is DEM 60. Registration fee is accepted only in these currencies:

- \* 40 USD (US dollar)
- \* 30 EUR (Euro)
- \* 60 DEM (Deutsche Mark)

The registration fee does not include a printed hard-copy of the manual. Please print the documentation on your own printer or use the online help (AmigaGuide or html) while working with MUIbase.

#### Payment methods

-----

##### Cash

This is one of the preferred payment method and maybe the cheapest for you and me. Enclose the cash in two pieces of nonwhite paper to disguise it. Do not send coins.

##### Eurocheque

Write the cheque for 60 DEM or 30 EUR. Other currencies than DEM and EUR are not accepted here! You must write your card number on the back of the cheque, otherwise I can't cash it in.

##### Money transfer

Transfer 60 DEM or 30 EUR to my bank account Bank 24, Germany, BLZ 380 707 24, account number 314 726 100.

If you have internet access then you can also transfer the money to my account and send me the registration by email or use the registration interface on the MUIbase home page <http://www.amigaworld.com/support/muibase>. This is probably the best way to register your MUIbase copy. I am checking my bank account almost every day, thus I will see your money pretty fast.

##### Postal Money Order

You can pay using postal money order, but you must mail or email me the registration form because it's possible that only your name is included with the PMO when I receive it, so I also need your address. Ask your local post office for more information.

##### Credit card via ShareIt.

This might be a good option for you if you want to order MUIbase by internet. Please see the instructions on the MUIbase web page <http://www.amigaworld.com/support/muibase> on how to use this registration method.

#### Ordering

-----

---

The easiest way for ordering MUIbase is to fill in a registration form or to use the Register program, which asks you all the necessary information and creates an ASCII registration form. Blank registration forms are available in these formats:

- \* Final Writer

- \* ASCII

You'll find these files in the Register directory.

If you don't have a printer then create an ASCII registration form with the Register program and copy the necessary information to a piece of paper. The information I need is:

- \* your name

- \* address

- \* email address (if available)

- \* payment method and currency

- \* method of delivery

When you have filled in the registration form, please send it to me with your payment. My address is:

Steffen Gutmann  
Orleanstr. 47  
81667 Munich  
GERMANY

Or send it by email to:

gutmann@ieee.org

Method of delivery

-----

If you have an Internet email address, I will email you the key-file. Otherwise I will send you a 3.5" disk by post with the key-file and the latest version of MUIbase.

The latest version of MUIbase can always be down-loaded from Aminet or from

<http://www.amigaworld.com/support/muibase>.

## 1.4 MUIbase/Distribution

Distribution

=====

---

You are granted the right to share MUIbase with others, as long as you distribute the MUIbase archive exactly as you received it, with all associated files included. Registered users may not distribute their private file "MUIbase.key".

Under no circumstances may you charge more than a normal copying fee and shipping costs for distributing the MUIbase archive without express written consent from the copyright holder.

Permission is hereby granted, without written agreement and without license fees, to copy and distribute the MUIbase archive, provided that the above conditions are met, to

- \* All who will distribute this software for free!
- \* All free accessible INTERNET servers and PHONE boxes!
- \* All Aminet sites
- \* All others who do NOT take more than \$5.- for one disk that includes this software!
- \* All others who do NOT take more than \$20.- for one CD that includes this software!

Distributing beta-versions of MUIbase is strictly prohibited.

## 1.5 MUIbase/Disclaimer

Disclaimer  
=====

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 1.6 MUIbase/Third party material

MUI  
===

This application uses

---

## MUI - MagicUserInterface

(c) Copyright 1993-2000 by Stefan Stuntz

MUI is a system to generate and maintain graphical user interfaces. With the aid of a preferences program, the user of an application has the ability to customize the outfit according to his personal taste.

MUI is distributed as shareware. To obtain a complete package containing lots of examples and more information about registration please look for a file called "muiXXusr.lha" (XX means the latest version number) on your local bulletin boards or on public domain disks.

If you want to register directly, feel free to send

DM 30.- or US\$ 20.-

to

Stefan Stuntz  
Eduard-Spranger-Straße 7  
80935 München  
GERMANY

Support and online registration is available at

<http://www.sasg.com/>

## BetterString &amp; TextEditor

=====

MUIbase uses BetterString.mcc & TextEditor.mcc, (c) 1997-2000 by Allan Odgaard. See <http://www.diku.dk/students/duff/> for more info or latest version.

## Additional custom classes

=====

MUIbase uses NList.mcc (C) 1996-2000 Gilles Masson.

## Icons

=====

Some icons used in the MUIbase distribution are copied from the DefaultIcons set which is distributed e.g. on the Meeting Pearls CD 3, directory Contrib/DefaultIcons. These icons are Copyright by Michael-Wolfgang Hohmann and Angela Schmidt (for a more detailed Copyright description see MP3).

---

## 1.7 MUIbase/Welcome to MUIbase

Welcome to MUIbase

\*\*\*\*\*

MUIbase is a fast and very flexible database for the Amiga. It is for users who want to manage data in a comfortable and easy way. MUIbase is able to manage any kind of data, e.g. addresses, CD series, movies, or your income and expense. The power of MUIbase lies in its clear and powerful graphical user interface and its programming capabilities. The latter makes it possible to automatically calculate and maintain nearly everything, starting from automatically summing up values, e.g. for calculating the total amount of income or the total amount of recorded time of a CD, up to automatically creating and printing letters for any purpose.

MUIbase is the successor of AmigaBase, a hierarchical programmable database which is still available but considered obsolete with the appearance of MUIbase. All registered users of AmigaBase can get a free upgrade to MUIbase by sending their AmigaBase registration number and return address to me (Email preferred).

MUIbase offers the following features:

- \* Handling of multiple projects at the same time.
  - \* Attributes can be of type string, memo (multi line text), integer, real, date, time, bool, choice (one item out of many items), reference (easy way to reference a record of another table), button (for starting MUIbase programs), and virtual (compute value on the fly).
  - \* The string type can also manage lists of strings, files, and fonts. An OS 3.x datatype gadget allows displaying external images.
  - \* Unlimited number of records.
  - \* Dynamic loading of records. Records which are not needed may be flushed from memory (e.g. when memory is low).
  - \* Programmability. With the easy and powerful MUIbase programming language complex tasks can be implemented. The language also includes a SELECT FROM WHERE query for easy and fast data retrieval.
  - \* Ordering of records by any combinations of attributes.
  - \* Flexible and powerful search and filter facility.
  - \* Query editor which allows entering and managing of SELECT FROM WHERE queries. The queries can be saved and the results can be printed.
  - \* Import and export facility.
-

- \* Uses MUI as user interface. The interface is highly customizable. External images can be included in the user interface.
- \* Portability. MUIbase development has been done under the idea of easy portability. The system/gui part has been separated from the ANSI/C part such that porting MUIbase will result in only porting the system/gui part. The author is currently investigating a Java implementation of the user gui. This would make the database available for nearly all computer systems, including Linux, Windows, and MacOS.

The unregistered version of MUIbase is crippled in several ways. See Registration, for a list of limitations in the unregistered version.

## 1.8 MUIbase/Getting started

Getting started

\*\*\*\*\*

This chapter describes how to install MUIbase on your computer, what the hard- and software requirements are, and how to start MUIbase.

Installing MUIbase

=====

For installing MUIbase on your hard disk you need the Commodore Installer. This tool can be found in the Aminet under directory /pub/aminet/util/misc. The Installer may also be included in the MUIbase archive. Be sure to use Installer version 43.3 or up, otherwise the installation script may fail.

Before beginning the installation, make sure that your computer and your system software is compatible to MUIbase. See required hardware and required software sections.

Required hardware

-----

You need an Amiga with an 68020 processor or higher, 2 MB of ram, and a hard disk with at least 2 MB of free space on it. For larger applications more hard disk space and ram is necessary.

Required software

-----

MUIbase needs OS version 3.0 or up. It may also run under OS 2.x but there is no guarantee that all features will work properly there.

Additionally, MUIbase requires that MUI version 3.8 or higher is installed on your system.

Starting the installation

-----

---

If you have received MUIbase as an archive file, unpack the archive to a temporary directory. Do not unpack it to the target directory!

Double click the MUIbase installer script Install-MUIbase and follow its instructions. The script asks for a directory where the software should be copied to. Do not enter the directory where you have unpacked the MUIbase archive to.

The script is also capable of updating an existing MUIbase installation. In this case enter the directory where you have previously installed MUIbase. When updating an existing MUIbase installation all necessary files are replaced by new ones. This includes the sample projects in the Demos directory. Therefore do not place projects into this directory nor use one of the sample projects for managing your own data!

After a successful installation you can remove the MUIbase archive from your system and store it elsewhere, e.g. on a floppy disk.

#### Key-file

-----

If you are a registered user of MUIbase and have received a key-file, you should move the key-file to one of the following directories:

- \* directory-containing-the-MUIbase-executable
- \* MUIbase:
- \* KEYS:
- \* KEYFILES:
- \* S:

MUIbase will search in these directories for the file MUIbase.key and if found (and the key-file is valid) enable all crippled features.

Remember that the key-file contains your personal data and you are not allowed to give it away. Also do not modify the file in any way or MUIbase is likely to crash.

Please keep a safety copy of your key-file on one or two diskettes.

#### Starting MUIbase

=====

MUIbase can be started from Workbench or from CLI. From Workbench double click the MUIbase icon. You can also double click a MUIbase project icon. This starts MUIbase and the selected project is automatically loaded by MUIbase. It's also possible to mark several MUIbase projects by shift clicking them and double click the last project.

From CLI type MUIbase [FILE1 ...], where FILE1 ... are optional projects to be loaded by MUIbase.

Quitting MUIbase

=====

To quit MUIbase select menu item Project - Quit or close all opened projects.

## 1.9 MUIbase/Tutorial

Tutorial

\*\*\*\*\*

The making of a family-tree database.

This chapter is a small tutorial describing how the basic things of MUIbase work. Within the tutorial a small project is developed that allows managing of your family tree. The project that results after applying all steps in this tutorial can be found in the Demos directory of your MUIbase installation.

How MUIbase works

=====

MUIbase could be said to operate in two different modes, record-editing and structure-editing mode.

The record-editing mode is where you change, add, or delete your records.

The structure-editor lets you edit how your database should look like and what tables and attributes it contains.

Besides this there is the program-editor where you write program functions that are executed either automatically when entering some data in an attribute or when pressing a program button.

Starting with a project, the structure-editor

=====

To create a database you first of all need to define it's contents. In MUIbase this is done in the structure-editor. To enter the structure-editor you either press RAMIGA-s (Right Amiga button and the letter s) or by choosing Structure Editor from the Project menu. There you will find three different sections:

Tables

In Tables you change, add, or delete the tables you need.

Attributes

In Attributes you change, add, or delete attributes. These attributes each belong in one of the above mentioned tables.

Display

In Display you design the looks of your database, how it should be

displayed.

#### Adding a table

=====

First we need a table, press the New button just below the list view in the Table section. You will then get a requester which asks you to enter some data:

##### Name

This is where you enter the name of a table. The name must start with an uppercase letter and can consist of up to 20 characters. The name can be changed later. In this tutorial we set the name to Persontable since it's going to hold all the persons names in this database.

##### Number of records

A table can either consist of only one or of an unlimited number of records. In this case it should be set to unlimited since we are going to add more than one person.

##### Trigger functions

Any call from the user to add or delete records can be controlled by a program function, this is where you set which function to call. Since we don't have written any program functions yet, a look in any of the list-views will not show anything.

When this is done, just press the OK button and we have our first table, Persontable.

#### Adding an attribute

=====

Then we need a string attribute for that table, press New in the attributes-section. Attributes also need some settings:

##### Name

The same as for a table, first letter an uppercase one and altogether a maximum of 20 characters. This attribute we will set to Name since it's to contain the names of the persons we are about to add.

##### Type

Here we choose what type this attribute should be. There are a couple of different ones but for this attribute we use a string attribute.

##### Max length

Here you can define the maximum number of characters a user can enter for the string. We set this to 30.

##### Initial value

It's possible to have some attributes to use an initial value for every new record you add, here is where you enter what it should contain. Leave this line blank.

##### Trigger

---

An attribute could also trigger a program function to be executed. For example, if you enter a name then you can have a program to check if this name already exists.

#### Displaying the project

=====

After pressing OK you now should notice some changes in the display section. Change the choice button on the top of the display section to Root window. There you see what the root window holds, currently Persontable. If you change the choice button back to Table mask you can see how this table, Persontable, is presented. Currently it's displayed as one panel with one attribute.

Now double-click on Panel(Persontable) at the top of the list in the display section and a window should appear, allowing you to set how this panel should be displayed:

#### Title

The title of a table can be different than what it's real name is. Our table is called Persontable but here we could set it to be THIS IS THE TABLE PERSONTABLE! if we prefer that better.

#### Background

The background can be changed to whatever suits your taste.

#### Gadgets

Here we can define what gadgets we want the panel to have.

Press OK and then double-click on Name in the list-view in the display section. That should bring up the window that contains the settings for how to display the string attribute Name.

#### Title

The same as for the panel. The string you enter here is what is really displayed when in record-editing mode.

#### Shortcut

Here you can set a letter that should be used together with RAMIGA to jump to this attribute, when in record-editing mode.

#### Home

Makes the cursor to jump to this attribute whenever a new record is added. In our case we will always or most of the time enter the name first in a new record, so set it.

#### Read only

Set this if the attribute should be read only. Leave it unset.

#### Weight

Decides how much of this attribute should be visible when competing for the space with other attributes. For example, if three 50-character strings resides in a window that only has room for 100 characters, then this number decides how much space the string gets relatively to the other ones. Leave it at 100.

#### Background

---

The same as for the panel.

#### Bubble help

Here you enter any text you think would be helpful to the user. The bubble help appears after holding the mouse still over an attribute for some seconds. Set this to If you need help, call the author at 112.

Leave the structure-editor (RAMIGA-s or Structure Editor in the Project menu) and go to record-editing mode, to see how the database looks in real life. You'll see the headline which is the string you may have entered in the display section for the panel. The record counter should say #0/0 since we haven't added any records yet. Thereafter is the filter button and the two step-buttons. Below all this you should have Name and the text you may have entered in the display section for this attribute. If you haven't changed any text at all when in the display section then the panel should be named Persontable, and the string attribute Name. Move the mouse above the string attribute Name and leave it there for a couple of seconds. If you entered something in the bubble help above then this text should appear in a bubble.

#### Adding two reference attributes

=====

Now we will add two reference attributes. Reference attributes are a bit different than other attributes. As their name might imply they refer to records. This will get more understandable as you try it out for yourself in just awhile.

Enter the structure-editor again and add two more attributes to Persontable. Press New in the attributes section, name it Father, and change it's type to Reference. A reference attribute has only one setting:

#### Set reference to

Tells which table the attribute it's about to refer to. Should already be pointing to Persontable, leave it that way and press Ok.

Add another attribute by pressing New in the attribute section and call it Mother. The attributes type should also be set to reference and point to table Persontable.

As you may have noticed there are now three attributes in the display section. Click on Father and then on the buttons up and down located just to the left. This will change where Father is positioned when looking at it while in record-editing mode. Put Father at the top, Name in the middle, and Mother at the bottom.

Then we just have to set what contents the reference attributes Father and Mother should display from the referenced records. Double-click on Father in the display section and then click on Extras. There we choose to display the string attribute Name, then we press Ok and repeat the procedure for Mother.

#### Adding records

=====

---

Now we should add some records. Leave the structure-editor and enter record-editing mode. To add a new record you simply press RAMIGA-n or choose New record from the Table menu. The cursor should now automatically jump to the attribute we have set to Home earlier in the display section in the structure-editor. Add two records, one with your fathers name in Name and another one with your mothers name in Name. Thereafter you add another record with your own name in Name.

Now it's time to understand those reference attributes. By pressing on the list-view button on Father we get a list of all records this reference attribute could refer to. Choose your fathers name and do accordingly down below on the mothers list-view.

Now you should have three records, you, your father and your mother. In your record, your fathers name should be visible in Father at the top and your mothers name should be in Mother at the bottom. You can browse through the records by pressing ALT together with Cursor up/down.

But hey! My parents also has/had parents you say! So let's add another four records, the third generation. Just add the records one by one and write their names in Name. If you don't remember their names then just enter fathers father, mothers father or something like that instead. Then you browse through all the records and set Father and Mother to what they should contain. When this is done you should have at least seven records, your record, your parents records and your grandparents records.

Filter  
=====

Since we now have some records to work with, we could try out the filter function. The filter can sort out records you don't want to display, they will still remain in the database itself but they will not be displayed.

To edit the filter you press LAMIGA-f or choose Change filter from the Table menu. When you have done so a strange looking window with loads of operators will appear. This is where you set what conditions a record must fulfill to get displayed.

In this small example we will use the LIKE command, which lets you do a joker comparison of an attribute. Press once on the LIKE button to the right and then double-click on Name in the list to the left and (LIKE Name ) should appear in the string just above the Ok and Cancel buttons. Thereafter you type "\*a\*" so the whole string shows (LIKE Name "\*a\*"). This means that MUIbase should display all records that contain the letter a anywhere in Name.

Press Ok and you may notice that records with no a in Name, no longer are visible. Since the letter a is quite common in most languages and names, all records might still be visible but you can try other letters to make the filter function more clear. When you're done, go to record-editing mode.

As mentioned earlier there is a button on the panel that says F. This F indicates if the filter is on or off. Finally when you're done

---

testing, turn the filter off so that all records are visible.

#### Queries

=====

Now that we have played with the filter function a bit, we might as well play around with the query feature that's in MUIbase. Queries can be used to display information from a database matching certain criteria.

Choose Queries from the Program menu or press RAMIGA-\* to open the query editor. Now a window with some buttons on the top and two larger areas below appear. The string to the upper left is where you enter the name of what you want to call the query you make.

#### Run

Compiles and runs the query. It goes through the database and then displays the info according to your specifications.

#### Print

Prints out the result of the query.

#### Load and Save

Lets you load and save each of the queries.

The first large area is where you enter the query. The second large area is where the result is displayed.

Now let's produce a list of all those persons we tried to filter out previously. Type Persons with an a in their name in the string to the upper left. This is the title for this query. In the upper large area, type:

```
SELECT Name FROM Persontable WHERE (LIKE Name "*a*")
```

Now when you run this query, either by RAMIGA-r or by pressing the Run button, it will produce a list of all persons with the letter a in their name. Try changing the letter to see different results.

At this time we can introduce the AND command. Press the list-view button just to the left of the Run button, in the query editor. Then choose New and name it Persons with both letter a and s in their names. Then type

```
SELECT Name FROM Persontable WHERE
(AND (LIKE Name "*a*") (LIKE Name "*s*"))
```

Note that we are still using the LIKE command for choosing records containing the letters a or s in their names, but the AND command requires that BOTH LIKE criteria are met. Therefore, only records with BOTH the letters a and s in their name are displayed when the query is run.

#### Adding a table with a memo and a button attribute

=====

This was two ways of selecting and displaying the database. Another

way of displaying data can be done by a program. In order to display data we can use an attribute type called memo.

Enter the structure-editor and press New in the table section. Name the new table Controltable and set it's number of records to Exactly one. Click and hold down the mouse button on the new table. Now drag it just a bit above the middle of Persontable and release the button. In the table section, Controltable should now be on top, and Persontable below of it.

Make sure that Controltable is activated, then press New in the attribute section. Set this new attributes type to Memo and give it the name Resultmemo. Press Ok and then add another attribute to Controltable by once again pressing New in the attribute section. This time, set it's type to Button and name it Pedigree.

To give the database a better look, click once on Pedigree in the display section and push it to the top by pressing the Up button once.

Programming MUIbase to do a pedigree  
=====

Ok so now we have a button that can start a program and a memo to display data in. It's therefore time to enter the program-editor. This is done either by pressing RAMIGA-p or by choosing Edit from the Program menu. The editor has three buttons:

Compile & Close

Which does just that. It compiles the program and leaves the program-editor.

Compile

Compiles the programs but stays in the program-editor.

Undo

Undo all changes from the time you entered the program-editor.

As all program functions you write will reside in this one window, we will need to separate them from each other. In MUIbase this is done by the DEFUN command. Everything between the two parenthesis will be a part of the function pedigree in this example:

```
(DEFUN pedigree ()
; This is DEFUN's end parenthesis
)
```

With this in mind we now type in the first function which will produce a family tree of the current person in the database and place the result in Resultmemo. This pedigree function is in fact three functions:

- \* pedigree which sets Controltable.Resultmemo by calling another function.
  - \* getpedigree which collects the pedigree to a list.
-

```

* pedigree2memo which converts this list to the memo.

; The program pedigree

(DEFUN pedigree ()
  (SETQ Controltable.Resultmemo (pedigree2memo (getpedigree Persontable NIL ←
    ) 0 3))
)

; The program getpedigree

(DEFUN getpedigree (person:Persontable level:INT)
  (IF (AND person (OR (NULL level) (> level 0)))
    (LIST person.Name
      (getpedigree person.Father (1- level))
      (getpedigree person.Mother (1- level))
    )
  )
)

; The program pedigree2memo

(DEFUN pedigree2memo (pedigree:LIST indent:INT level:INT)
  (IF (> level 0)
    (+
      (pedigree2memo (NTH 1 pedigree) (+ indent 8) (1- level))
      (IF pedigree (SPRINTF "%*s%s\n" indent "" (FIRST pedigree)) "\n")
      (pedigree2memo (NTH 2 pedigree) (+ indent 8) (1- level))
    )
    ""
  )
)

```

Ok, so here we have the final program functions. Type this in and make sure that all the parenthesis are where they should be. Too many or too few parenthesis is the most common fault you'll get when MUIbase is pre-compiling your programs. The error message from MUIbase will probably be Syntax Error, in that case. Press Compile & Close and hopefully the window will close, which means that MUIbase couldn't find any faults in it while pre-compiling.

Don't worry too much if you don't understand all the commands at first. As with all programming languages it requires some time and practice to master.

Now we have a program to run, but first we have to assign the program function to the Pedigree button. This is done by entering the structure-editor, selecting Controltable in the table section and double-clicking on the Pedigree attribute in the attribute section. Then open the list-view Trigger. In this list, all your program functions will be listed, currently there should be three functions: pedigree, getpedigree and pedigree2memo. Double-click on pedigree as the program function the Pedigree button will trigger, then press Ok and leave the structure-editor.

Now if everything is done correctly, pushing the Pedigree button will produce a pedigree of the current person. Try changing person to see some different pedigrees.

Programming MUIbase to list a person's children

As the next addition to this database requires some more records, you should add your brothers and sisters. If you don't have any then write My faked sister 1, My faked brother 1 which of course should be set to have the same parents as you.

Then go to the program-editor and type the following for creating another program.

```

; The program children counts how many children a person has.
; First we define the variables we will use, i.e "children" is set to contain ←
  "\n\n".

(DEFUN children ()
  (LET ( (children "\n\n") (nrofchildren 0) (currentperson Persontable) )

    ; For all records in Persontable do the following:
    ; If the current person appears as father or mother in any of the records ←
      then:
    ;   add the name to the variable children
    ;   increase the nr of children - counter with 1.

    (FOR ALL Persontable DO
      (IF (OR (= currentperson Father) (= currentperson Mother))
        (
          (SETQ children (+ children Name "\n"))
          (SETQ nrofchildren (+ nrofchildren 1))
        )
      )
    )

    ; Then we write the result into the Controltable memo, Resultmemo.
    ; If the current person doesn't have any children then write one string.
    ; If he/she has children then write another string.

    (SETQ Controltable.Resultmemo
      (+ Persontable.Name (IF (> nrofchildren 0)
        (+ " is the proud parent of " (STR nrofchildren) " children(s).")
        " does not have any children (yet :-).")
      ))
    )

    ; If the current person has children then add their names.

    (IF (<> nrofchildren 0)
      (SETQ Controltable.Resultmemo
        (+ Controltable.Resultmemo "\n\n"
          (IF (= nrofchildren 1)

```

```

        "The child's name is:"
        "The children's names are:"
    )
    children
)
)
)

; This is the end parenthesis of the LET-command.
)

; This is the end parenthesis of DEFUN children.
)

```

To create variables, we use the LET command. Variables created with the LET command are local and only visible within the LET commands open and closing parenthesis. So any command that want to access these variables will have to be within these parenthesis.

All we need to execute this program is a new program button, so enter the structure-editor and add a button attribute in Controltable. Call it Children and choose children as the program function it should trigger.

To bring some order in the mask of Controltable it's now time to introduce groups. All objects can be ordered into vertically or horizontally aligned groups.

In the display section, click on Pedigree and shift-click on Children, thereafter you click on the Group button to the left. Now the two program buttons will be together in a vertically aligned group. However, we want this one to be horizontally aligned so double-click on the VGroup that has appeared in the display section. This will open a window that lets you change the settings for this group. Set the title to Programs and check the Horizontal button.

At this time we can remove the name of Resultmemo in Controltable. Double-click on Resultmemo in the display section and remove the name. Resultmemo will still exist but it's name won't be shown anymore.

To make things easier if we add more programs or attributes in the Controltable, we should place Resultmemo and the Programs group in a vertical group. Be sure of that you only have marked the group Programs and Resultmemo and then press on Group. This will put Programs and Resultmemo into a vertical group.

Leave the structure-editor and take a look at the result. Then press the Children button to see the number of children and their names of the current person.

This example could very well be extended into a full-grown pedigree-program. The only real limits are your fantasy and the size of your hard drive.

## 1.10 MUIbase/Basic concepts

Basic concepts

\*\*\*\*\*

Before you start setting up your own database projects and entering data for them, you should now about some basic concepts MUIbase is built on.

Projects	MUIbase projects.
Tables	Basic data management.
Records	One row of a table.
Attributes	One column of a table.
Attribute types	Types available for attributes.
Table of attribute types	Summary about attribute types.
Memory consumption	How much space each type needs.
Relationships	Connecting tables.
User interface	Elements for layout.

## 1.11 MUIbase/Projects

Projects

=====

A MUIbase project consists of all relevant information you need for managing your data. This includes the project's user interface, the project's data you entered and the programs you wrote for the project.

A project can be loaded from, saved to, and deleted from disk. Any change you make to a project is only done in memory. At any time you can go back to the state of the last saved project by reloading it.

MUIbase is able to handle multiple projects at a time. Therefore it is not necessary to start MUIbase twice just to load another project.

## 1.12 MUIbase/Tables

Tables

=====

MUIbase manages data in tables. A table is organized in rows and columns, where rows are called records and columns are called attributes.

See the following table for an example on how to structure a set of addresses in a table.

Name	Street	City
------	--------	------

```

-----|-----|-----
Steffen Gutmann | Wiesentalstr. 30 | 73312 Geislingen/Eybach
Charles Saltzman | University of Iowa | Iowa City 52242
Nicola Müller | 21W. 59th Street | Westmont, Illinois 60559

```

There exists a special table kind which can hold exactly one record. A table of this kind is sometimes useful for controlling the database project, e.g. you can put buttons into this table for executing various actions, or a read-only attribute for displaying project related information. For an example suppose you have an account database where you store all your income and expense. An exactly-one table now could have a read-only attribute of type real for displaying the total balance.

Each table has two record pointers, a pointer to the record that is currently displayed in the user interface (called gui record pointer) and a pointer to the record that is the current one while executing a MUIbase program (called program record pointer).

You can define any number of tables for a MUIbase project. (Note: in the unregistered MUIbase version there is a limitation of 5 tables per project).

Tables can be added to, renamed, and deleted from a project.

## 1.13 MUIbase/Records (concept)

Records  
=====

A record is one row of a table. It holds all information about one set, e.g. in a table that manages addresses, one record holds one address.

Each record has a record number that reflects the record's position in the table. This number may change when you add or delete records.

For each table a record called initial record exists that holds the default values for initializing new records. The initial record always has a record number of 0.

Records can be added to, changed, and deleted from a table. There is no upper limit on the total number of records for a table. The records are not necessary held in memory but are loaded from and stored to disk when needed. Ok, there are two upper limits for the total number of records of one table. One comes from the fact that the record number must fit into a long value, which limits the total number of records to 4294967295. Another limitation is that for each record a small record is kept in memory. These limitations should make MUIbase still be usable for record numbers of 10,000 and more.

## 1.14 MUIbase/Attributes

Attributes  
=====

An attribute defines one column of a table. It specifies the type and appearance of the corresponding column.

Attributes can be added to, renamed, and deleted from a table. There is no upper limit on the number of attributes per table. (Note: in the unregistered MUIbase version there is a limitation of 10 attributes per table).

For each attribute you have to specify a type that restricts the contents of this attribute. See the next section for a list of available attribute types.

## 1.15 MUIbase/Attribute types

Attribute types  
=====

For attributes the following types are available:

String	Any single line of text. Can also be used to store font-names, filenames, and external images.
Integer	Ordinary numbers.
Real	Floating point numbers.
Bool	Boolean value.
Choice	One item out of many items.
Date	Date values.
Time	Time values.
Memo	Multi-line text.
Reference	Reference to another record.
Virtual	Compute value on the fly.
Button	For triggering MUIbase programs.

Some of the attribute types support a special value called NIL. This special value has the meaning of an undefined value, e.g. for a type of date it means an unknown date. The NIL value is similar to the NULL value of other database systems.

Please note that once you have set the type of an attribute, you cannot change it later.

## 1.16 MUIbase/String type

String attributes  
-----

String attributes can store any single line of text. Strings are the most often used attribute type in a database project. For example an address database will store the name, street, and city of a person each in its own string attribute.

For a string attribute you have to specify the maximum number of characters allowed in the string. This number does not directly affect the amount of memory or disk space that is used by this attribute because only the actual string contents are stored (other databases have called this feature compressed strings). If necessary the number can be changed after you have installed a string attribute.

String attributes can also be used to store font- and filenames. For filenames external viewers can be launched to display the file contents. Furthermore an in-line image class allows displaying the image of a file.

String attributes do not support the NIL value.

## 1.17 MUIbase/Integer type

Integer attributes

-----

Integer attributes store integral values in the range of -2147483648 to 2147483647. They are mostly used for storing quantities of any kind, e.g. the number of children of a person, or the number of song titles on a CD.

Integer attributes support the NIL value representing an undefined integer value.

## 1.18 MUIbase/Real type

Real attributes

-----

Real attributes store floating point values in the range of -3.59e308 to +3.59e308. They are used for storing numbers of any kind, e.g. the amounts of money in an income/expense project.

For each real attribute you can specify the number of decimal places used for displaying the real value, though internally always the full precision is stored.

Real attributes support the NIL value representing an undefined real value.

## 1.19 MUIbase/Bool type

Bool attributes  
-----

Bool attributes store one bit of information. They are used for storing yes/no or true/false values, e.g. in a project managing invoices a bool attribute could store the has paid? information.

Bool attributes use TRUE and NIL as bool values. NIL in this case stands for a value of FALSE.

## 1.20 MUIbase/Choice type

Choice attributes  
-----

Choice attributes store one item out of an enumeration of items. For example, in an address project a choice attribute can be used for storing the state, where state is one out of USA, Canada, Germany, or others.

A choice attribute does not store the whole item string but the item number (index) in a record. The number of items and the items itself can be modified after the attribute has been created. However when making changes to a choice attribute, values in existing records are not changed to reflect the new situation.

Choice attributes do not support the NIL value.

## 1.21 MUIbase/Date type

Date attributes  
-----

Date attributes store ... ehm ... dates. For example, a date attribute can be used for storing birthdays.

The format for entering and displaying date values can be one of DD.MM.YYYY, MM/DD/YYYY, or YYYY-MM-DD, where DD, MM and YYYY are standing for two and four digit values representing the day, month and year of the date respectively.

Date attributes support the NIL value representing an undefined date.

## 1.22 MUIbase/Time type

---

## Time attributes

-----

Time attributes store times or periods of time. For example, a time attribute can be used for storing the durations of music titles on a CD.

The format for entering and displaying time values is fixed to HH:MM:SS where HH is a two digit value in the range of 0 to 23 representing the hours, MM a two digit value in the range of 0 to 59 representing the minutes, and SS a two digit value in the range of 0 to 59 representing the seconds.

Time attributes support the NIL value representing an undefined time.

## 1.23 MUIbase/Memo type

### Memo attributes

-----

Memo attributes store multi-line text of any size. Text size is handled dynamically which means that memory is only allocated for the actual text size. In a project managing movies for example, a memo attribute can be used to store summaries of the movies.

Memo attributes do not support the NIL value.

## 1.24 MUIbase/Reference

### Reference attributes

-----

Reference attributes are a special type of attributes, normally not found in other database systems. Reference attributes store a pointer to another record. The referenced record may reside in the same or in any other table than the reference attribute belongs to.

For example in a pedigree project two reference attributes can be used for storing pointers to the father and mother record. Or in a project managing CDs and music titles, a reference attribute in the table holding the music titles can be used to point to the records of the corresponding CDs.

For displaying a reference attribute, any attributes of the referenced record can be specified. Entering a reference attribute can be done by selecting a record from a list of records.

Reference attributes support the NIL value. Here a value of NIL stands for a pointer to the initial record of the referenced table.

---

## 1.25 MUIbase/Virtual

### Virtual attributes

Virtual attributes are a special type of attributes that do not store any information in the database itself, but compute them on the fly when needed.

For example, in a project managing invoices where a real attribute holds the amounts of money excluding tax, a virtual attribute can be used to "store" the amounts of money with tax. Every time the value of the virtual attribute is needed, e.g. for displaying it, it is computed from the corresponding value without tax.

For displaying virtual attributes three kinds exists: bool, string and list. These three kinds allow showing the value of the virtual attribute as a TRUE/FALSE value, as a single line of text including numbers, dates, and times, or as a list of several single lines, e.g. for listing all music titles of a CD.

Virtual attributes support the NIL value standing for FALSE (bool kind), undefined (string kind), or empty (list kind).

## 1.26 MUIbase/Button

### Buttons

Actually, buttons are not a real attribute type as they cannot store or display any information. Buttons are just used for triggering MUIbase programs.

The reason to keep them as an attribute type is because as an attribute buttons can be accessed by their attribute name. This allows to use the name of a button in a MUIbase program, e.g. to disable or enable the button. Another reason is that buttons have similar properties as attributes, e.g. trigger functions.

## 1.27 MUIbase/Table of attribute types

### Table of attribute types

The following table summarizes all available attributes types:

Type	Description	Nil allowed?
String	For strings of lengths 1..999. A string can also be used for storing	No

	filenames, font-names or one-string-out-of-n-strings. For filenames you can add a field where the contents of the file are displayed as an image.	
Integer	For storing integer values.	Yes
Real	For floating point numbers.	Yes
Bool	TRUE or NIL.	Yes (NIL = FALSE)
Choice	One number out of n numbers. Numbers are represented by label strings.	No
Date	For storing a date value (1.1.0000 - 31.12.9999).	Yes
Time	For storing time values (00:00:00 - 23:59:59)	Yes
Memo	Multi-line text of unlimited length.	No
Reference	For storing a reference to a record of another table.	Yes (NIL means initial record)
Virtual	For displaying results from a MUIbase program.	Yes
Button	For triggering a program function	No (N/A)

## 1.28 MUIbase/Memory consumption

### Memory consumption

=====

Each attribute type needs a certain amount of memory for storing one value in a record. All types except virtual and button have in common that they need a 2 bytes header holding internal information. Additionally, type dependent space is needed for storing the actual value. The following table lists how much memory including a possible 2 byte header one value of the given type needs in memory and on disk.

Type	Memory space	Disk space
String	2 + 4 + STRING-LENGTH + 1	2 + STRING-LENGTH + 1
Integer	2 + 4	2 + 4
Real	2 + 8	2 + 8
Bool	2 + 0	2 + 0
Choice	2 + 2	2 + 2
Date	2 + 4	2 + 4
Time	2 + 4	2 + 4
Memo	2 + 4 + MEMO-LENGTH + 1	2 + MEMO-LENGTH + 1
Reference	2 + 4	2 + 4
Virtual	0	0
Buttons	0	0

Here STRING-LENGTH stands for the length of the string to be stored and MEMO-LENGTH for the text size of the memo to be stored.

## 1.29 MUIbase/Relationships

### Relationships

=====

Up to now you know how to organize your information into tables with records and attributes. But you may also want to setup relationships between tables.

For example if you want to collect CDs in a database project you would have two tables, one for the CDs them-self and one for the music titles of the CDs. Of course you could also have all music titles within the CD table but then you would have a fixed number of music titles for each CD.

So having these two tables, you now need a link for each music title to the CD containing this title. This is called a relationship between the two tables. Normally you use a reference attribute for setting up such a relationship.

By installing a reference attribute into a table you automatically have a relationship between the table the attribute resides in and the table it refers to.

The following classes of relationships can be distinguished:

One to one relationships	Simple ones.
One to many relationships	Most often used ones.
Many to many relationships	Complex ones.

## 1.30 MUIbase/One to one relationships

### One to one relationships

-----

One to one relationships are very simple relationships where for each record you have one or zero partners in another or in the same table.

For example in a database project that manages your favorite movie actors you could setup a reference attribute called married with that shows the person the actor is married with. An actor that is currently not married does have a NIL value for this reference field.

Of course, no one prevents the user to set the married with references of several actors all to the same person. However by programming MUIbase it is possible to detect such cases and handle accordingly.

## 1.31 MUIbase/One to many relationships

One to many relationships  
-----

One to many relationships are useful for connecting a set of records to one record in another or the same table.

For example in a project managing your bank accounts you could have one table for all bank accounts and one table for all transactions. Now you surely want to know which transaction belongs to which account so you setup a reference attribute in the transaction table referring to the account table.

One to many relationships are the most often used ones. You can use them for managing any hierarchical-like structures, e.g. CDs with music titles, bank accounts with transactions, family trees, etc.

One to many relationships are also the basis for realizing many to many relationships as described in the next section.

## 1.32 MUIbase/Many to many relationships

Many to many relationships  
-----

Many to many relationships are used when you want a set of records to refer to another set of records.

For example in a project that manages movies and actors you would have two tables, one for the movies and the other one for the actors. Now for each movie you want to know the actors that took part in the movie. So you might think to setup a reference attribute in the actor table that refers to the movie table. But when doing this you could only have one movie referenced for each actor because there is only one reference field in the actor table. So what you need is an unlimited number of references from the actor table to the movie table.

This is done by adding a new table that just has two reference attributes, one pointing to the actor table and the other to the movie table. Now you can enter the relationships by adding new records to this table. For each movie-actor constellation you add a new record and specify the movie and actor by setting the corresponding reference fields.

If you want to know in which movies an actor took part then you only have to search for all records in the new table that refer to the actor in question and look at the movie records the found records refer to. Such a search can be done automatically by MUIbase and the result can be displayed in a list-view.

The following tables show an example of how to connect a set of actors to a set of movies.

---

	Title	Country
-----		
m1:	Batman	USA
m2:	Batman Returns	USA
m3:	Speechless	USA
m4:	Tequila Sunrise	USA
m5:	Mad Max	Australia
m6:	Braveheart	USA

	Name
-----	
a1:	Michael Keaton
a2:	Jack Nicholson
a3:	Kim Basinger
a4:	Danny DeVito
a5:	Michelle Pfeiffer
a6:	Geena Davis
a7:	Christopher Reeve
a8:	Mel Gibson
a9:	Kurt Russell
a10:	Sophie Marceau
a11:	Patrick McGoohan
a12:	Catherine McCormack
a13:	Christopher Walken

MovieRef	ActorRef
-----	
m1	a1
m1	a2
m1	a3
m2	a1
m2	a4
m2	a5
m2	a13
m3	a1
m3	a6
m3	a7
m4	a8
m4	a5
m4	a9
m5	a8
m6	a8
m6	a10
m6	a11

From these tables you can find out for example that Mel Gibson took part in the movies Tequila Sunrise, Mad Max, and Braveheart, or that in movie Batman the actors Michael Keaton, Jack Nicholson, and Kim Basinger took part.

### 1.33 MUIbase/User interface

User interface

=====

MUIbase uses a graphical user interface (gui) organized in a hierarchical way for displaying record contents and for letting the user enter new data. Each project owns its own root window in which further gui elements (including sub windows) can be placed. The gui elements are also called display objects.

A table is displayed in an own gui element called mask. A mask can display only one record at a time. Its layout and the attributes included in the mask are customizable by the user.

The following gui elements are available for designing a project's gui layout:

Windows	Root and sub windows.
Masks	Displays a table.
Panels	Controls a table.
Attribute objects	Shows a data item of one record.
Text objects	Static text descriptions.
Images	Static images for decoration.
Space objects	Layout and separator item.
Groups	Groups gui elements vertically and horizontally.
Balance objects	Dynamic sizing of group elements.
Register groups	Pages of gui elements.

### 1.34 MUIbase/Windows

Windows

-----

Windows can be used to spread information of a project across several independent areas.

Each project automatically has its own root window. If needed, e.g. if the space of the root window exceeds, additional sub windows can be created. Sub windows can also have further sub windows.

For each sub window a window button is placed into the parent window allowing to open and close the sub window. The window button looks like a normal text button but has a small window icon showing the open/close state of the corresponding sub window.

Root windows do not have a parent window and therefore have no window button. Closing a root window means closing the whole project.

A window can have any other gui elements (except panels) as children. If no child has been added to a window then an empty display image (see Empty display image) is shown.

## 1.35 MUIbase/Masks

### Masks

-----

A mask is used to display the contents of a table. Only one record of the table can be shown at a time.

The mask may include a panel (see next section) for controlling the table. Other gui elements like attribute or text objects can be placed into a mask to show the record contents.

Masks cannot be placed inside other masks as this would lead to a hierarchy of masks and therefore to a hierarchy of tables which is not supported in MUIbase. If you want to setup a hierarchy of tables, use an 1:n relationship between two tables.

## 1.36 MUIbase/Panels

### Panels

-----

A panel is a small rectangular area placed at the top edge of a mask. A panel can display a title, e.g. the name of the corresponding table, a pair of numbers showing the record number of the current record and the total number of records, and several buttons for controlling the table, e.g. for displaying the next or previous record.

Only one panel can be defined for a mask. If you setup a panel for a mask then an additional border is drawn around the mask, otherwise no border is drawn.

## 1.37 MUIbase/Attribute objects

### Attribute objects

-----

Attribute objects are used to display the contents of one item of a record.

Depending on the type of the attribute the gui element is either a string gadget (types string, integer, real, date and time), a check-mark gadget (type bool), a cycle gadget or a set of radio buttons (type choice), an editor gadget (type memo), a pop-up list-view (type reference), a text, check-mark, or list-view gadget (type virtual) or a text or image button (type button). In some cases the gui element may

---

also be a simple text gadget if the attribute object is set to read-only.

## 1.38 MUIbase/Text objects

Text objects  
-----

Text objects are used for describing the various field elements of a record mask or just to display some static text like copyright information somewhere in a window.

## 1.39 MUIbase/Images

Images  
-----

Images can be displayed anywhere in a window. An image can be an internal MUI graphic, a simple color field, or a pictures from an external file. The image size can be set to resize-able or fixed.

The image is static. If you would like to store images in a table you should use a string attribute (see String type).

Please note that images are not available in the unregistered version of MUIbase.

## 1.40 MUIbase/Space objects

Space objects  
-----

Space objects are used to insert space in the layout of a window or a table mask. A space object can have a vertical (or horizontal) bar for delimiting other gui elements.

## 1.41 MUIbase/Groups

Groups  
-----

Gui elements can be grouped into horizontal or vertical groups. A group places its children from left to right (horizontal group) or from

---

top to bottom (vertical group).

A group can surround its child objects with a rectangular frame, an optional title can be displayed to the top of a group, and a flag controls whether space is inserted between the child objects or not.

## 1.42 MUIbase/Balance objects

Balance objects  
-----

Balance objects can be placed anywhere between other child objects into a window, mask, or group object. A balance object allows the user to control the weight values of the other child objects and therefore how much space each child gets.

## 1.43 MUIbase/Register groups

Register groups  
-----

A register group can be used to layout some gui elements into several pages that can be activated one at a time. This is useful if the user interface becomes large and you don't want to spread it over several windows.

## 1.44 MUIbase/Managing projects

Managing projects

\*\*\*\*\*

In this chapter you find:

File format	MUIbase project file structure.
Info	Information about current project.
Clear project	Start a new project.
Open project	Load project from disk.
Save project	Save project to disk.
Delete project	Delete project from disk.
Close project	When the project is done.
Check data integrity	If your project ever gets corrupted.
Swap records	Learn about MUIbase memory management.

## 1.45 MUIbase/File format

File format

=====

A MUIbase project consists of several files stored under its own directory. From Workbench you will not notice the directory structure since double clicking a project icon will not open the directory but start MUIbase loading the project. From CLI however you can see the directory structure of a MUIbase project.

All files of a project are put into one directory which is created by saving a project. Do not remove or place any files or further directories in this directory! They will get lost when reorganizing the project.

The directory contains a file called Structure.mb where the descriptions of all tables, attributes, filters, etc. are stored. The record headers are also placed here. For each table you will find a file with the table's name. Here all records of a table are stored. Last but not least there is a file called .lock. Do not remove this file. If you did it by accident then simply recreate it, the file contents are not important. This file is used for locking a project, that is, MUIbase first locks this file exclusively and then opens other files. If the locking fails, MUIbase knows that there is already a MUIbase application working on this project. Only one MUIbase application is allowed to work on the same project at a time since record files are opened in read/write mode and we don't want to have mixed data written by two or more applications. ;-)

## 1.46 MUIbase/Info

Info

=====

MUIbase keeps some information about each project. Select menu item Project - Info to get information about the current project. The information you get consists of the project's name, the number of tables, the total number of records in all tables, and a value that shows how many bytes a reorganization of this project would gain. The gain is however only a rough estimate and should not be treated as an exact number. Especially if you have made many changes to the structure of the project (adding or removing attributes) then this value is far from accurate.

## 1.47 MUIbase/Clear project

Clear project

=====

---

To begin a new project select menu item Project - Clear - Project. This clears the current project and MUIbase opens an empty window displaying the MUIbase logo. After starting MUIbase without any projects you are automatically in this mode.

By selecting menu item Project - Clear - Records you start a new project using the structure of the current one. This means that everything except the record data of the current project is used for the new project.

If the current project at the time you selected one of the clear menu items has not been saved to disk then a safety requester appears asking for confirmation of the operation.

## 1.48 MUIbase/Open project

Open project

=====

MUIbase can handle any number of projects at a time. You are only limited by the size of your available memory. If you want to edit another project, select menu item Project - Open new. This opens a new project root window. You can now load a project for this window.

To load a project select menu item Project - Open - Project. This opens a file-requester where you can choose a project from. It's also possible to only load the structure of a project, that is, the whole project is loaded except the record data. To do this, select menu item Project - Open - Structure.

If you were editing a project at the time of choosing any of the above menu items and the project has not been saved then a safety requester appears asking for confirmation.

## 1.49 MUIbase/Save project

Save project

=====

All changes you make to a project are only done in memory. Thus if you want to make them permanent you have to save the project to disk. This is done by choosing menu item Project - Save. If your project doesn't have a name yet then a file-requester appears first asking for a filename.

The reason why MUIbase does not automatically save a project when you change it is that this way it is you who decides when to save a project and you can always go back to the last saved version of your project. This mechanism is similar to the COMMIT and ROLLBACK commands in SQL database systems.

If you save a project, all modified records are written to disk and the file Structure.mb is recreated. Before creating the new Structure.mb file, MUIbase first renames a possibly existing Structure.mb file to Structure.old to have a safety copy in case the save operation fails.

This mechanism guarantees fast load & save operations but it is not reorganization free. If you have modified many records then the physical place where the records lie and the resulting fragmentation may become disadvantageous. Therefore a menu item Project - Save & Reorg exists that does a save & reorganize operation. This operation may take some time depending on the number and size of the records. The save & reorganize operation creates a new directory and rewrites all project related files. The old directory is deleted on success.

Another good place to execute a reorganization is when you have done changes to the data-structure of a project, e.g. after you have installed a new attribute in a table. These changes are not applied immediately to all records because it would take too much time to load each record, modify it, and save it back to disk. Therefore these changes are put on an internal todo list which is applied after loading a record. Applying this list to a record takes only little time. However the longer the list gets the more time it needs. Reorganizing a project causes the todo list to be applied to all records, so if you have made many changes to the project structure then reorganizing a project will shorten the time for loading records.

You can also save & reorganize a project to a new filename keeping the old project untouched. To do this select menu item Project - Save & Reorg As. MUIbase will prompt you to enter a new name for the project.

## 1.50 MUIbase/Delete project

Delete project  
=====

MUIbase offers a menu item to delete a project. If you select Project - Delete, enter a filename in the file-requester and confirm the deletion in an additional safety requester then the specified project is deleted from disk.

MUIbase does nothing special here, it simply deletes the directory with all files in it. It is safe to do this in any other way, e.g. by using Workbench or an Amiga Shell.

## 1.51 MUIbase/Close project

---

Close project  
=====

If you are done with a project you can close it by selecting menu item Project - Close. This removes the project from memory and closes all windows belonging to it. If the project contains changes that have not been saved then a safety requester appears offering to save, continue or cancel the operation.

For closing a project you can also select menu item Project - Save & Close which saves the project first if there were any changes and then closes it.

## 1.52 MUIbase/Check data integrity

Check data integrity  
=====

MUIbase can verify if all data in your project is still valid and not corrupted by programs trashing your project files or by system crashes. Choose menu item Project - Check data integrity for starting this process.

Usually you should never need this functionality and MUIbase should always report that data integrity is perfect. But if it ever happen to you that your project contains internal errors, that is, some records can't be loaded anymore, then you can repair your project by using this menu item.

MUIbase will write a log file of all affected records and you can save and reorganize the project afterwards. In the log file, records that might be corrupted and records that are not accessible anymore and therefore have been deleted are listed by their record number in the old (corrupted) project and by their record number (in parenthesis) in the reorganized project.

Note: Prior to MUIbase V1.4 there was a bug that caused writing corrupted project files. You should be able to repair these projects with this menu item. Since MUIbase V1.4 this bug has been fixed and you should never get corrupted projects anymore and checking data integrity should always report that data integrity is perfect.

## 1.53 MUIbase/Swap records

Swap records  
=====

MUIbase doesn't need to keep all records of a project in memory. Thus loading and saving of projects is much faster. When loading a

---

project a record header is allocated for each record. The data itself is only loaded when needed, e.g. when it is displayed on the screen. The total number of records is still limited by available memory since each record header needs some few bytes of memory.

You can specify how much memory MUIbase should use for the records of a project. Choose one of the predefined values found in menu item Preferences - Record memory. MUIbase will not preallocate a block of the specified memory size, it only checks from time to time if the current size of allocated memory is larger than the specified value. The specified value is not a hard limit, only a suggestion. If MUIbase needs more memory then it will ignore the upper bound and allocate it.

If MUIbase runs out of memory or if the upper limit for the record memory size has been reached then MUIbase tries to free as much record memory as possible. In this case MUIbase may write modified records to disk to get the maximum available memory possible. You can also force MUIbase to do this by choosing menu item Project - Swap records.

If there is enough memory to hold all records in memory and you have specified an upper memory limit that is high enough (e.g. unlimited) then MUIbase never needs to swap out records.

MUIbase maintains a free list for each record file. If you delete a record then the record's file space is added to the free list. Also if you change a record and the record needs to be written to disk then the old file space is added to the free list. However MUIbase makes sure that by reloading you can always go back to the point of the last save operation. MUIbase will not trash areas which are free but where a record still sits that could be reached by reopening the project.

## 1.54 MUIbase/Preferences

Preferences

\*\*\*\*\*

MUIbase offers several preferences items the user can set to his likes. This chapter shows what preferences items are available and gives general information about how the preferences system works.

Record memory	Size of record buffer.
Record delete requester	Safety requester when deleting records.
Ext. editor for prog.	Use your favorite editor for programming.
Icon creation	Create project icons.
Icon tool name	Tool name in project icons.
Formats	Real and date formats.
External editor	Specify your external editor.
External viewer	Specify your external viewer.
Popups in cycle chain	Include popup buttons in the cycle chain.
Confirm save & reorg project.	Safety requester when saving and reorganizing a ↵
Confirm quit	Safety requester when quitting MUIbase.
Program include dir	Where to look for external include files.

Program debug info	Compile with or without debug information.
Program output file	Where program output goes.
Project dep. settings	Global versus project local preferences.
MUI	MUI's preferences.
Load and save prefs	Make settings permanent.
Empty display image	Image for empty windows.

## 1.55 MUIbase/Record memory

Record memory

=====

MUIbase does not need to keep all records of a project in memory. Instead it uses a buffer for holding only a small number of records. By choosing a value from menu item Preferences - Record memory you can set the size of this buffer. Each project has its own buffer, so if you have opened two projects each having a record buffer size of 1MB, MUIbase will use up to 2MB for the records of both projects.

MUIbase will not allocate the memory a priori, it uses a dynamic allocation scheme. Also, the buffer size you specify is not a hard limit. If MUIbase decides it needs more memory then it will try to allocate it.

Once the buffer gets full, or if MUIbase runs out of memory, all records are flushed from the buffer. This means that unchanged records are simply freed and changed records are first written to disk and then freed.

By giving MUIbase a higher value for the record buffer you will notice a speed increase in accessing the records because now since more records are held in memory fewer records have to be loaded from disk. If you set the record memory size to unlimited and all records fit into memory then MUIbase operates with optimal speed.

## 1.56 MUIbase/Record delete requester

Record delete requester

=====

You should check menu item Preferences - Record delete requester if you want MUIbase to pop up a safety requester asking for confirmation whenever you try to delete a record. Leave the item unchecked if records should be deleted silently. Default is checked.

## 1.57 MUIbase/External editor for programming

External editor for programming

=====

You can check menu item Preferences - External editor for programming if you always want to use the external editor for editing MUIbase programs. This means that whenever you open the program editor, the external editor is also started automatically. If you don't check this field then you can still launch the external editor by choosing the corresponding item in the editor's context menu. Default is unchecked.

## 1.58 MUIbase/Icon creation

Icon creation

=====

Check menu item Preferences - Icon creation if you want MUIbase to create an icon for each project. You can still paint and use your own icon for a project. MUIbase will not overwrite existing icon images as long as they are project icons. Default is checked.

## 1.59 MUIbase/Formats

Formats

=====

By selecting menu item Preferences - Set formats you can specify the formats used when displaying or printing real and date values. After selecting the menu item a new window appears containing the following items:

- \* a field Real format for setting the decimal character of real values. You can choose between Decimal point and Decimal comma.
- \* a field Date format for specifying how date values are output. You can choose between Day.Month.Year, Month/Day/Year and Year-Month-Day.
- \* two buttons Ok and Cancel for leaving the window.

The initial values for real and date formats are determined according to the information found in the operating system's Locale library.

When you are done with all settings, press the Ok button to leave the window and update the display.

## 1.60 MUIbase/Icon tool name

Icon tool name  
=====

By selecting menu item Preferences - Icon tool name you specify the tool name that should be executed when you double click a project icon. Normally you should specify the path to the MUIbase program, e.g. MUIbase:MUIbase which is the default.

## 1.61 MUIbase/External editor

External editor  
=====

MUIbase's editor fields offer a special menu item where you can call an external editor for editing the text contents. The name of the editor and its parameters must be specified by selecting menu item Preferences - External editor. You should enter a command string that gets executed when calling the external editor. Use %f at the place where the filename normally stands. The %f string is replaced with the actual filename of the temporary file MUIbase creates before executing the command.

For example you can use CED %f -keepio for using CED as external editor (make sure that your default stack size is at least 8192 bytes in size, otherwise CED might crash).

Default is Ed %f.

## 1.62 MUIbase/External viewer

External viewer  
=====

In MUIbase you can use strings for storing filenames. For displaying the contents of such a filename, an external viewer is needed. Normally this viewer uses the Amiga-OS datatype system for displaying pictures, showing animations, or playing music. To specify this viewer select menu item Preferences - External viewer.

Like for the external editor (see External editor) you have to specify a command string here. Default is Multiview %f.

## 1.63 MUIbase/Popups in cycle chain

Popups in cycle chain

=====

In the graphical user interface entered in the structured editor there might exist popup buttons, e.g. file, font or listview popups beneath a string gadget. These buttons are usually not included in the cycle chain, that is, you can't use the Tab key to activate them. However if you check menu item Preferences - Popups in cycle chain then all popup buttons are include in the cycle chain.

Please note that changing the status of this menu item has only an effect after rebuilding the user interface, e.g. by switching to the structure editor and back to the user interface.

## 1.64 MUIbase/Confirm save & reorg

Confirm save & reorg

=====

Saving and reorganizing a project can take quite some time depending on the size of the project. Therefore, if you select menu item Project - Save & Reorg or Project - Save & reorg as, a safety requester pops up asking for confirmation of this operation.

The requester only appears if menu item Preferences - Confirm save & reorg is checked, thus you can disable this requester by de-selecting the menu item.

## 1.65 MUIbase/Confirm quit

Confirm quit

=====

If you try to quit MUIbase and there are unsaved projects then MUIbase pops up a safety requester asking for confirmation. However if all projects have been saved, the program usually quits silently.

If you want MUIbase to always pop-up a requester when quitting then check menu item Preferences - Confirm quit. In this case you always get a safety requester when selecting menu item Project Quit. Nevertheless it is still possible to quit MUIbase silently by closing all projects.

## 1.66 MUIbase/Program include directory

Program include directory

=====

The programming feature of MUIbase allows to include external sources within the project's program (see #include for more information). Menu item Program - Include directory allows to set a directory where MUIbase should search for such include files. Default is MUIBase:Include.

## 1.67 MUIbase/Program debug information

Program debug information

=====

For compiling a project's program, you can choose whether debug information should be included in the executable or not. If you compile without debug information and a run-time error occurs then an error discription is generated but there is no information about where exactly the error occured. If you compile with debug information then you also get the exact error location.

Use menu item Program - Debug information to turn debug information for compilation on and off. After changing this state, don't forget to recompile the project's program by choosing menu item Program - Compile.

## 1.68 MUIbase/Program output file

Program output file

=====

When executing a MUIbase program all output which is directed to stdout can be printed to a file. The filename has to be entered in the requester that appears when choosing menu item Program - Output file. Here, you can also specify if the file should be opened in appended mode or if it should be cleared before the first output happens. Besides traditional files, the Amiga-OS system allows several special filenames here, e.g.:

- \* PRT: prints the output on your printer.
- \* CON:///MUIbase output/CLOSE/WAIT prints the output in a Shell window.
- \* CONSOLE: prints the output in the Shell window where MUIbase has been started from.

## 1.69 MUIbase/Project dependent settings

Project dependent settings

=====

MUIbase can manage several projects and for each project you can have your own set of preferences. Sometimes however it is desirable to have all projects sharing the same setting for one of the preferences items. This is realized by using a status flag for each preferences item which indicates whether each project has its own value for this field or if the value is shared among all projects.

By choosing menu item Preferences - Project dependent settings a window with a list of several preferences items appears. All items in this list can be set either project dependent or global. The list contains the following items:

- \* Record memory
- \* Record delete requester
- \* Formats
- \* Confirm save & reorg
- \* Program include directory
- \* Program debug information
- \* Program output file

The status flag of preferences items not included in this list cannot be changed and is either project dependent or global, e.g. the setting of preferences item icon tool name is always global since it doesn't make sense to have different icon tool names for different projects.

If you check the box to the right of a preferences item then this item is made project dependent, otherwise it is global. Project dependent settings are stored in the project file Structure.mb, whereas global settings are stored in a global preferences file.

## 1.70 MUIbase/MUI

MUI

===

Since MUIbase is a MUI application you can also specify the MUI preferences for this application by choosing menu item Preferences - MUI.

---

## 1.71 MUIbase/Load and save preferences

Loading and Saving of preferences  
=====

The global preferences can be loaded and saved to disk. When MUIbase is started, it automatically loads its preferences from ENV:MUIbase.prefs.

MUIbase doesn't save its global preferences automatically. Thus, if you make changes to the global preferences and you want MUIbase to remember them after a restart then you have to save them by choosing menu item Preferences - Save preferences. The global preferences are written to both ENVARC:MUIbase.prefs and ENV:MUIbase.prefs. The following items are stored:

- \* Record memory size
- \* Record delete requester
- \* External editor for programming
- \* Icon creation
- \* Icon tool name
- \* Formats
- \* External editor
- \* External viewer
- \* Confirm save & reorg
- \* Confirm quit
- \* Program include directory
- \* Program debug information
- \* Program output file
- \* Project dependent settings
- \* Directory names for several file requesters
- \* Default max length for new string attributes
- \* Default text format for new attribute objects
- \* Cursor position of program editor
- \* Name of help file

There is also a menu item called Preferences - Load preferences. Choose this menu item in case you made changes to the preferences and

---

want to go back to the previously saved version.

## 1.72 MUIbase/Empty display image

Empty display image  
=====

If you start MUIbase without loading any projects then MUIbase opens a window with an empty display. The empty display usually consists of a image which is loaded from the file MUIbase:Images/EmptyWindow.iff.

By default this image is the MUIbase logo but you can replace it by copying your favorite picture to MUIbase:Images/EmptyWindow.iff. It is also possible to have no image for the empty display window, just make sure that the above filename doesn't exist. In this case MUIbase only prints the text Empty display in the window.

## 1.73 MUIbase/Record-editing

Record-editing  
\*\*\*\*\*

In this chapter you find:

Active object	Where your input goes.
Adding records	How to add new records to a table.
Changing records	How to edit record contents.
Deleting records	If you don't need some records any more.
Browsing records	How to view other records.

## 1.74 MUIbase/Active object

Active object  
=====

MUIbase uses a cursor for displaying which object is the active one. If the active object is a string object then a normal block cursor appears, other objects get a special frame around them. You can cycle through the active objects by pressing the Tab or S-Tab keys.

The table in which the active object resides is called the active table. The panel of a table can be set to the active object. This ensures that you can always set a table to be the active one, although the table may not contain any other activate-able objects.

## 1.75 MUIbase/Adding records

Adding records

=====

If you select menu item Table - New record a new record is allocated in the active table. The record is initialized with the initial values for all attributes. It is also possible to duplicate the current record of the active table by selecting menu item Table - Duplicate record.

If you have installed a trigger function for adding a new record (see Creating tables) then this trigger function is called for creating the record. For more information on this mechanism, See New trigger.

## 1.76 MUIbase/Changing records

Changing records

=====

To change the current record in a table you can activate any attribute object within the table's mask and enter a new value. For string, integer, real, date, time, and memo attributes you can use the usual editing commands.

An attribute object may have been configured as read-only. In this case you can't change its value (exception: string attribute with pop-up button).

String attributes with pop-up button

=====

If a string attribute has a pop-up button attached to it then you can press the pop-up button and get a requester to set the string contents, e.g. a file-requester for choosing a filename, or a list of strings to choose one from. The pop-up button can always be used to set the string attribute's value even if the attribute is set to read-only.

Right to the string field a V button might appear. Pressing this button calls an external viewer to display the file specified in the string field.

Entering bool values

=====

The checked state of bool attribute can be toggled with the left mouse button, or with the space bar if the object is the active one.

Entering choice values

=====

---

For choice attributes you can choose a value by clicking onto it, or by using the Cur-Up and Cur-Dn keys to browse through all choice labels.

#### Entering date values

=====

Date values can be entered in the format DD.MM.YYYY where DD, MM and YYYY are standing for two and four digit values representing the day, month and year of the date respectively. It is possible to omit the year value of a date. In this case the current year is appended to the input string.

By inserting a single integer value, a date value relatively to the current date can be specified, e.g. when entering 0 the today's date is used, or when entering -1 yesterday's date is used.

#### Entering time values

=====

Time values can be entered in the format HH:MM:SS where HH is a two digit value in the range of 0 to 23 representing the hours, MM a two digit value in the range of 0 to 59 representing the minutes, and SS a two digit value in the range of 0 to 59 representing the seconds.

It is possible to omit the number of hours and the number of minutes. In this case a value of 0 is used. E.g. if you enter 6:30 it is automatically expanded to 00:06:30.

#### Memo context menu

=====

Memo attributes have a context menu that offers further editing possibilities:

- \* Cut, Copy, and Paste allow exchanging data with the clipboard.
- \* Clear deletes all text in the memo.
- \* Undo and Redo allow going back and forth the changes you made to the memo contents.
- \* With Open text and Save text you can load and save the memo contents from/to a disk file.
- \* External editor launches an external editor for editing the memo. See External editor, for more information about the external editor.

#### Entering reference values

=====

For reference attributes there are several options to enter a value:

- \* To the right of a reference attribute you find a pop-up button which if pressed opens a list of records and two further buttons. Choose a record from the list to set the reference to this record, Initial to set the reference to the NIL value, or Current to set the reference to the current record of the referenced table.

- \* Use the context menu of the reference attribute to see further options to set the reference. You also find the corresponding shortcuts there.
  - The Previous and Next menu item set the reference to the previous or next record.
  - The Backward and Forward menu items will let you go 10 records backward or forward.
  - The Search for... menu item pops up a requester to enter a pattern for searching a record. If you don't check the All fields? box then the search is only done in the attribute which is first in the list of attributes that is used for ordering the referenced table. The reason why only this attribute is examined is because prefix searching can be done very efficient on this attribute as the attribute acts like an index. For more information about the search requester, see Search for.
  - Once you have entered a search pattern in the search requester, Search backward and Search forward can be used to find the previous or next matching record.
- \* If the reference attribute object is the active object, you can enter a character to search for a record that has its contents of the first order attribute of the referenced table starting with this character. The search is done case insensitive. E.g. if you enter l then a search pattern of l\* is used for searching the record. You can append further characters to the search buffer by typing them in uppercase, e.g. lASSIE would search for a record matching the string Lassie\*.

#### Entering NIL value

=====

To enter the NIL value enter any invalid string for the given attribute type, e.g. if you enter xyz in an integer attribute then the value of this attribute is set to NIL. Please note that not all attribute types support the NIL value. See Table of attribute types for an overview of all attribute types.

## 1.77 MUIbase/Deleting records

#### Deleting records

=====

To delete the current record chose menu item Table - Delete record. Before deleting the record a safety requester may appear asking you for confirmation. You can enable and disable this requester in the preferences settings (see Record delete requester).

If you have installed a trigger function for deleting records (see Creating tables) then this trigger function is called for deleting

---

the record. For more information on this mechanism, See Delete trigger.

It's also possible to delete all records of a table by choosing menu item Table - Delete all records. Only records matching the record filter of the corresponding table are deleted. Before deletion a safety requester appears, if enabled. No trigger function is called when deleting all records.

## 1.78 MUIbase/Browsing records

Browsing records

=====

To view other records than the currently displayed one, select one of the sub menu items in menu item Table - Goto record. You can go to the previous, next, first, or last record, jump several records backward or forward, or enter the record number of the record you want to view. The record number in this context is the number that is displayed in the corresponding panel for that record (see Panels). The panel may also include two arrow buttons for going to the previous and next record.

Record browsing can be easily done using the CurUp and CurDn keys in combination with the Shift, Alt, and Ctrl keys. All possible combinations are listed in menu item Table - Goto record and in the following table:

	Alt	Ctrl-Alt	S-Alt
CurUp	Previous record	First record	Jump backward
CurDn	Next record	Last record	Jump forward

## 1.79 MUIbase/Filter

Filter

\*\*\*\*\*

Filters can be used to hide records. This chapter describes what types of filters are available and how to use them.

MUIbase knows two types of filters, record filters and reference filters.

Record filter	Using a boolean expression as filter.
Reference filter	Using a record reference as filter.

## 1.80 MUIbase/Record filter

Record filter  
=====

A record filter can be installed into a table to filter out records that are not of interest to you. Records that are filtered out are excluded in the table mask and thus the user can't see or browse to them.

Filter expression	How a filter looks like.
Changing filters	How to specify filter expressions.
Filter examples	Some examples.

## 1.81 MUIbase/Filter expression

Filter expression  
-----

A filter is defined by specifying a boolean expression that may contain calls to MUIbase programming functions. For each record of the table the filter is specified for, this expression is evaluated. If it returns NIL then the record is filtered out, otherwise it is included in the table mask.

Each table can have its own filter expression.

## 1.82 MUIbase/Changing filters

Changing filters  
-----

To change the filter of the current table select menu item Table - Change filter. This will open a window containing

- \* the name of the table you install the filter for in the window title.
  - \* a list of all attributes of the table that can be used in the filter expression. This list is placed in the left part of the window. If you double click a name then the name will be inserted into the filter expression at the current cursor position.
  - \* a cascade of buttons displaying MUIbase programming functions and operators. It is placed in the right part of the window. Click on one of the buttons to enter the corresponding function in the filter expression. Please note that the presented list of functions and operators is not complete. Other MUIbase functions not shown in one of the buttons must be entered by hand. Only
-

those MUIbase functions can be used that don't have side effects, e.g. it is not possible to write data to a file within a filter expression.

- \* a string input field to enter the filter expression. The attribute and function/operator names are inserted here. You can also directly input your filter expression.
- \* two buttons Ok and Cancel to leave the window.

After you are done with the specification of the filter expression click on the Ok button to leave the window. The entered expression is now compiled and, if successful, the expression is evaluated for all records. Those records for which the boolean expression doesn't hold are then discarded in the table mask.

In case the expression doesn't compile you will get an error message displayed in the window title bar.

A filter can be turned on and off by clicking on the F button in the table's panel (if it is installed), or by pressing the F key when the panel is the active object. After you have specified a filter expression for a table the filter for this table is turned on automatically.

If you (re-)activate a filter then all records of the table are checked whether they match the filter or not.

If a filter is currently active and you change a (filter relevant) attribute in a record of this table then the match-filter state of this record is not recomputed and stays unchanged.

If you allocate a new record in a table with an activated filter then there is no check if the new record matches the filter and the new record has its match-filter state set to TRUE.

## 1.83 MUIbase/Filter examples

Filter examples

-----

Here are some examples for valid filter expressions:

- \* NIL filters out all records.
- \* TRUE doesn't filter out any record.
- \* 0 same as TRUE as for MUIbase all expressions not equal to NIL are considered as TRUE.
- \* (> Amount 100.0) only displays records where the Amount attribute is greater than 100.0 (we assume here that the table has an attribute Amount of type real).

\* (NOT (LIKE Name "\*x\*")) filters out all records that have the letter x in the Name attribute (a string attribute).

Please note that MUIbase' programming language uses a lisp-like syntax. For more information about the programming language, see Programming MUIbase.

## 1.84 MUIbase/Reference filter

Reference filter

=====

Reference attributes can also have a filter behavior. This is useful e.g. for building a hierarchy of tables (like it is hard-wired in AmigaBase). As an example see the project Albums.

If the filter of a reference attribute is turned on then the following features are activated:

1. The user can only access records in the attribute's table that have the reference set to the current record of the referenced table.
2. If the referenced table changes its current record then also a new current record is searched and set for the attribute's table.
3. When allocating a new record the reference is automatically set to the current record of the referenced table.

Note: Cascading delete (like it is automatically done in AmigaBase) has to be implemented by the user himself (by using a delete trigger function).

Do not use the reference filter on cyclic graphs, e.g. self-referencing tables. It doesn't make sense and will only confuse the user.

## 1.85 MUIbase/Order

Order

\*\*\*\*\*

For each table in your database you can specify in which order its records should be displayed. This chapter describes how you can specify an order and what consequences it has.

Empty order	If you don't want to order your records.
Order by attributes	Easy ordering by comparing attributes.
Order by a function	Customized ordering.

Changing orders            How to specify an order.  
 Reorder all records      If they ever become unsorted.

## 1.86 MUIbase/Empty order

Empty order  
 =====

By default each newly created table has an empty order. This means that if you enter a new record, the created record is inserted at the current position, that is, behind the current record of the table. If you change some fields of a record, the position of the record within the table leaves unchanged.

## 1.87 MUIbase/Order by attributes

Order by attributes  
 =====

Sometimes it is useful to have the records sorted by a certain attribute, e.g. by the Name attribute if the table has such one.

In MUIbase you can specify for each table a list of attributes by which its records should be sorted. All records are first sorted by the first attribute in this list. If two records are equal in an attribute then the next attribute in the list determines the order. For each attribute you can specify if the records should be sorted ascending or descending.

For determining the order of attributes the rules of the following table are used:

Type	Order relation
Integer Choice	NIL < MIN_INT < ... < -1 < 0 < 1 < ... < MAX_INT (Choice values are treated as integers)
Real	NIL < -HUGE_VAL < ... < -1.0 < 0.0 < 1.0 < ... < HUGE_VAL
String Memo	NIL < "" < ... < "a" < "AA" < "b" < ... (String comparison is done case insensitive)
Date	NIL < 1.1.0000 < ... < 31.12.9999
Time	NIL < 00:00:00 < ... < 23:59:59
Bool	NIL < TRUE
Reference	NIL < any_record

(Records itself are not comparable for ordering)

If you have specified an order for a table then the records are automatically rearranged whenever you add a new record or change an order relevant field of a record.

## 1.88 MUIbase/Order by a function

Order by a function

=====

Sometimes you want a more complex ordering scheme than a simple attribute list as described in the last section. For example, an attribute list can't hold reference attributes, so it is not possible to sort your records by a reference field. The reason for this is that with reference attributes MUIbase is not able to keep your records ordered at all times (see Comparison function in the MUIbase programming chapter for more details).

However, MUIbase also offers the possibility to specify a comparison function for sorting your records. You can specify any function that you have written using the MUIbase program editor. The function will be called with two record pointers and the returned value should reflect the order of the two records. The function can use any operations for comparing the records, thus it can e.g. compare records using reference attributes. For more information on this mechanism, see Comparison function.

If you decide to use a comparison function for ordering a table then please note that MUIbase cannot always detect when records of the table have to be reordered since the dependencies are unknown. Use menu item Table - Reorder all records when records are becoming unsorted.

## 1.89 MUIbase/Changing orders

Changing orders

=====

To specify an order for the current table select menu item Table - Change order. This will open a window containing

- \* the name of the table in the window title.
- \* a choice field Type where you specify whether the table should be ordered by using an Attribute list or by using a Comparison function. Depending on the state of Type you can enter the data into the following items.

For an order using an attribute list, the following items exist:

- \* a list of all attributes of the table that can be used in the order list. This list is placed in the left part of the window. If you double click a name then the name will be inserted into the order list at the current cursor position.
- \* the current list of attributes used for ordering. This list is placed in the right part of the window. The top item in this list is the first attribute of the order list. You can rearrange items by dragging them to other positions in the list. To add further attributes drag them from the attribute list to the order list. You can remove an attribute from the order list by dragging the item out of the order list and dropping it onto the attribute list.

Each entry in the order list has an arrow symbol pointing up or down on the left side. You can toggle the state of the arrow by double-clicking it. If an arrow is pointing up then the sorting order for this attribute is done ascending, if it is pointing down, it is done descending.

For an order using a comparison function, there is the following item:

- \* a field Function for comparing records where you can enter the name of a function that should be called for comparing two records of the table. You can use the pop-up button to the right of the string field for choosing a name from a list of all function names. If you leave the field empty then an empty order is used. For more information on how to use a comparison function, including the arguments that are passed to it, see Comparison function.

Furthermore, the following buttons exist:

- \* a Clear button that clears all fields for an empty order.
- \* two buttons Ok and Cancel for leaving the window.

To enter a new attribute order list, choose Attribute list in the Type field and press the Clear button. Then use drag & drop as described above to build up a new list of attributes. If you want to have an empty order then just don't add any attributes to the order list.

When you are done specifying the order, press the Ok button. MUIbase will then reorder all records of the table. As this can need some time, a busy mouse pointer will appear.

## 1.90 MUIbase/Reorder all records

Reorder all records

=====

If you ever think that some records of a table became unsorted, e.g. when using a comparison function for ordering then you can choose menu item Table - Reorder all records for sorting all records.

## 1.91 MUIbase/Search for

Search for  
\*\*\*\*\*

For record browsing you can use a search requester to search for a specific record. The search feature uses a search pattern that you provide and checks all records for a successful match with this pattern. If it finds one then this record will be displayed in the table mask.

Search requester	How to enter a search pattern.
Forward-backward search	Go to the next/previous matching record.
Examples	Some search pattern examples.

## 1.92 MUIbase/Search requester

Search requester  
=====

To open the search requester select menu item Table - Search for. This will open a window containing

- \* a string field for entering the search pattern. Use the characters \* and ? as jokers. The character \* matches any number of characters (including zero characters) whereas ? matches exactly one single character.
  - \* a field Case sensitive. If you check it, search for uses case sensitive string comparison, otherwise case insensitive string comparison is used.
  - \* a field All fields. If checked then all fields of a record are tested for a successful match with the search pattern. Otherwise only the attribute which has been the active one when the search requester has been opened is tested. In case the active object at the time when opening the search requester has not been an attribute object, this field is checked and disabled automatically.
  - \* two radio buttons for the search direction, Forward and Backward.
  - \* two radio buttons for specifying from which record the search should be started. First/last record will start the search from the first or last record depending on the search direction. Current record starts the search from the current record.
  - \* two buttons Search and Cancel for leaving the window.
-

After you have entered a search pattern and left the requester with the Search button MUIbase starts searching for a matching record. The comparison of an attribute field with the search pattern is always done string based, attribute fields of non-string types are therefore converted to strings first.

If a matching record is found, it is displayed as the current record in the table mask. Otherwise an Search pattern not found message appears.

If you search on an attribute that is used as the first attribute for ordering and your search pattern doesn't start with a joker (\* or ?) then an improved search algorithm (binary search) is used that makes use of the record order. This can speed up the search drastically.

### 1.93 MUIbase/Forward-backward search

Forward/backward search  
=====

Two further menu items allow to search for the next and the previous occurrence of the search pattern. Select menu item Table - Search forward for browsing to the next record that matches the search pattern, and Table - Search backward to go to the previous matching record.

### 1.94 MUIbase/Search pattern examples

Search pattern examples  
=====

Here are some search pattern examples:

- \* Lassie searches for records that have the string Lassie in one of the search attributes.
- \* \*x\* searches for records that have the letter x in one of the search attributes.
- \* ??? searches for records that have exactly three characters in one of the search attributes, e.g. a record with an entry UFO.

### 1.95 MUIbase/Import and Export

Import and Export  
\*\*\*\*\*

---

To share your records with other databases MUIbase offers a way to import and export records to and from other databases. Import and export is done by reading and writing ASCII files. Therefore the data you want to import must be in a special format described in the following section.

File format	How the format looks like.
Sample import file	An example.
Importing records	How to import records.
Exporting records	How to export records.

## 1.96 MUIbase/Import file format

File format  
=====

For importing records into MUIbase all records of a table must be available in a single ASCII file. If you want to import records of several tables you must have several import files, one for each table.

An import file consists of lines and columns. Lines are separated by a record delimiter, columns by a field delimiter. The delimiters can be specified in the import and export requesters. As the record fields itself may already contain these delimiters it is possible to use double quotes around all fields to protect them.

The import file must have the following structure:

- \* The first line contains the field names. For each name there must be an attribute with exactly the same name in the table where the records are imported to. If there is a name where no matching attribute can be found, an error message is generated.
- \* The following lines contain one record each. As all fields must be given as ASCII strings they are automatically converted to the type of the destination attribute. For attributes of type boolean, the field must be either NIL or TRUE (case insensitive), otherwise an error message is generated. For attributes of type choice the exact label string must be specified (case sensitive). For reference attributes you have to specify the record number starting with 1. For all other types a value of NIL is used if the field cannot be converted to the required type.
- \* If you decide to use double quotes then all record fields including the line with the field names must be surrounded by double quotes.

## 1.97 MUIbase/Sample import file

---

Sample import file

=====

The following sample import file uses `\n` and `\t` as record and field delimiter and double quotes around all fields. The file can be imported into a table with the following attributes:

- \* Name (string)
- \* NumChildrens (integer)
- \* Feminine (bool)
- \* Job (choice)
- \* Notes (memo)

```
"Name" "NumChildrens" "Feminine" "Job" "Notes"
"Janet Jackson" "???" "TRUE" "Musician" "Latest CD: The velvet rope"
"Bernt Schiele" "???" "NIL" "Scientist" "Research interests:
Robotics, Autonomy and Computer Vision"
"Gerhard" "0" "NIL" "Precision mechanic" ""
```

## 1.98 MUIbase/Importing records

Importing records

=====

To import records into the active table select menu item Table - Import records. This will open a window containing

- \* a string field for entering the import filename. Right to this field you find three buttons. Click on the first one to open a file requester for choosing the filename. The V button starts an external viewer to view the entered filename, and the E button starts an editor to edit the file contents.
- \* two string fields for entering the record and field delimiter. You can enter a single character here or an escape code by typing `\n`, `\t`, `\f`, `\???` (octal code), or `\x??` (hex code).
- \* a field Double quotes that can be checked to specify that the fields are surrounded by double quotes.
- \* two buttons Import and Cancel for leaving the window.

If you press the Import button, MUIbase will load the specified file and import all found records. If everything is going fine, MUIbase will ask you after the import process if you really want to add the imported records to the table. At this point you can still cancel the operation.

If an error occurs while reading the import file then an error message is generated.

If you need a more sophisticated import routine it is recommended to write your own import routine as a MUIbase program.

## 1.99 MUIbase/Exporting records

Exporting records

=====

To export records from the active table select menu item Table - Export records. This will open a window containing

- \* a string field for entering the export filename. Right to this field you find a pop-up button to open a file requester for choosing the filename.
- \* two string fields for entering the record and field delimiter. You can enter a single character here or an escape code by typing `\n`, `\t`, `\f`, `\???` (octal code), or `\x??` (hex code).
- \* a field Double quotes that can be checked to specify that the fields should be surrounded by double quotes.
- \* a field Filter. If checked only the records that match the currently installed record filter are written to the export file.
- \* two buttons Export and Cancel for leaving the window.

After you pressed the Export button, MUIbase will open the specified file and write out the records including a header line containing the attribute names. The export feature always writes all attribute fields of a table to the export file.

For a more customized export routine, you can use MUIbase' query editor (see Data retrieval) or you write your own export function as a MUIbase program.

## 1.100 MUIbase/Data retrieval

Data retrieval

\*\*\*\*\*

For data retrieval MUIbase offers two ways: the programming feature and the query editor.

The programming feature allows you to install buttons in table masks which, when pressed, call program functions. The usage of this feature is described in the structure editor chapter (see Structure editor)

---

and in the chapter about programming MUIbase (see Programming MUIbase).

This chapter describes the usage of the query editor, a requester where you can enter queries and view the output in a scrolling list-view.

Select-from-where queries	How a query looks like.
Query editor	How to enter and manage your queries.
Printing queries	Options when printing the result of a query.
Query examples	Some examples.

## 1.101 MUIbase/Select-from-where queries

Select-from-where queries

MUIbase offers a select-from-where query similar to the one in SQL database systems. The query allows you to list the record contents from one or more tables. Only records matching certain criteria are included in the output. The (incomplete) syntax of an select-from-where query is

```
SELECT EXPRLIST FROM TABLELIST [WHERE TEST-EXPR]
[ORDER BY ORDERLIST]
```

where EXPRLIST is a comma separated list of expressions to be printed (usually the attribute names) or a simple star \* matching all attributes of the specified tables, TABLELIST is a comma separated list of tables whose records are examined, TEST-EXPR is the expression that is tested for each set of records to be included in the output, and ORDERLIST is a comma separated list of attributes that defines the order for listing the output. Please note that the WHERE and ORDER BY fields are optional, denoted by the brackets [].

For example, the query

```
SELECT * FROM TABLE
```

lists the attribute contents of all records in the given table.

```
SELECT ATTR1 FROM TABLE WHERE (LIKE ATTR2 "*Madonna*")
```

lists the value of the ATTR1 field in all records of TABLE where the contents of field ATTR2 contain the word Madonna.

For more information about the select-from-where query including its full syntax, see Programming MUIbase, for more example see Query examples.

## 1.102 MUIbase/Query editor

Query editor

=====

For entering and running queries, open the query editor by choosing menu item Program - Queries. The query editor is able to manage several queries, however only one query is displayed at a time. The query editor window contains the following items:

- \* a string input field with an attached pop-up button. The edit-able string field displays the name of the current query. By pressing the pop-up button, a list with further query names together with several buttons appear. You can select one of the listed queries to make it the current one, press the New button to create a new query, press the Duplicate button to get a copy of the activated query, click on the Sort button to sort the list of queries, or press the Delete button to delete the selected query. For leaving the pop-up window without changing anything, click on the pop-up button again.
- \* a Run button that compiles and runs the query program and displays the output in the output list-view.
- \* a Print button that opens a requester (see Printing queries) for printing the results.
- \* two buttons Load and Save for loading and saving the current query program. If you save a query, the program text is written out in ASCII format and the first line holds the query name as a comment. The load function assumes the same format when reading a file. It is also possible to load and save queries by using the menu items in the context menu of the editor field for the query program. However, these menu items only save the program text and do not include the name of a query.
- \* an editor field for entering the query program. Here you usually enter a select-from-where query. However, it is also possible to enter any expression of MUIbase' programming language. This can be useful if you want to do simple computations or update some fields of a table by using a simple program. Please note that MUIbase automatically surrounds your program expression with a pair of parenthesis, thus you can omit the outermost ones.
- \* a list-view that displays the output after running the current query. The output is formatted into rows and columns. The title row holds the field names of the select-from-where query (usually the attribute names). The other rows hold the contents of the query result, one set of records per row. Each field entry is displayed in its own column. If you double click an entry in the list and this entry was generated by an attribute of type string or type reference then the selected record is displayed in the corresponding table mask. This is an easy way to jump to a certain record in a table mask.

The query editor is a non-modal requester. This means that you can

---

leave the query editor open and still work with the rest of the application. You can close the query editor at any time by clicking on the close button in the window title bar.

## 1.103 MUIbase/Printing queries

### Printing queries

=====

After you have run a query you can print the result to a file or printer by clicking on the Print button in the query editor. This will open a print requester containing the following items:

- \* a field Delimiter where you specify how the columns should be separated. Spaces pads the fields with space characters. Padding is done on the left or on the right side depending on the type of the field (numbers are padded on the left, text on the right side). Tabs inserts exactly one tab character between the columns. This can be useful if you want to use the print requester for exporting records (see below).
  - \* a field Font where you specify which font should be used for printing the output. NLQ stands for near letter quality which should print the output in better quality than Draft.
  - \* a field Size where you define the character size. Pica prints in a large font (10 cpi), Elite in a medium font (12 cpi) and Condensed in a small font (17 cpi).
  - \* a string field Init sequence where you can enter a string for initializing your printer. The contents of this field are written directly after opening the printer. For example you can use \33c as init sequence which resets your printer.
  - \* a field Indent where you can enter a number of spaces that are used for indenting each output line.
  - \* a field Headline that, if checked, prints the field names in the first line.
  - \* a field Escape codes. If not checked the output of all escape codes is suppressed which means that the settings of the fields Font and Size are ignored and the contents of Init sequence are not printed. Suppressing the output of all escape codes is useful if you want to generate an ASCII file, e.g. for exporting records.
  - \* a field Quotes that, if checked, surrounds all fields with double quotes.
  - \* a field After printing where you can specify how the output should be finished. Form feed prints a form feed character \f. Line feeds prints a number of line feed characters \n. The number of line feeds can be entered in the string field to the right of the Line feeds button. Nothing doesn't write anything to the printer.
-

- \* a string field Output with an attached pop-up button. You can use the pop-up button to open a file requester for choosing a filename or directly enter the filename into the string field. For writing the output to your printer enter PRT:, for printing to a window enter CON:////MUIbase output/CLOSE/WAIT into the string field.
- \* two buttons Ok and Cancel for leaving the print requester.

After you are done with all settings, click on the Ok button to start the print job.

The print requester can also be used for exporting records to an ASCII file. To do this, specify Tabs in the Delimiter field, set the number of spaces in the Indent field to 0, check Headline, un-check Escape codes to suppress the font, size and init sequence settings, optionally check Quotes if you want the field contents to be surrounded by double quotes, check Nothing in the After printing field, and enter the output filename in the Output field. Using the query editor together with the print requester for exporting records can be much more powerful than using MUIbase' import/export feature (see Import and Export) since in the query editor any query can be entered whereas the export requester only uses a fixed query.

## 1.104 MUIbase/Query examples

### Query examples

=====

To give you an impression of the power of the select-from-where queries this section gives you some sample queries.

Suppose we have two tables Person and Dog. Person has a string attribute Name, an integer attribute Age, and two reference attributes Father and Mother that refer to the father and mother records in table Person. The table contains the following records:

	Name	Age	Father	Mother
p1:	Steffen	26	p2	p3
p2:	Dieter	58	NIL	NIL
p3:	Marlies	56	NIL	NIL
p4:	Henning	57	NIL	NIL

Dog has a string attribute Name, a choice attribute Color and a reference attribute Owner that refers to the owner in the Person table. The table contains the following records:

	Name	Color	Owner
d1:	Boy	white	p3
d2:	Streuner	grey	NIL

Given these data the following sample select-from-where queries can

be run:

```
SELECT * FROM Person
```

results to:

Name	Age	Father	Mother
Steffen	26	Dieter	Marlies
Dieter	58		
Marlies	56		
Henning	57		

(For the reference attributes the Name field of the referenced record is printed.)

```
SELECT Name "Child", Age,
       Father.Name "Father", Father.Age "Age",
       Mother.Name "Mother", Mother.Age "Age"
FROM Person WHERE (AND Father Mother)
```

results to:

Child	Age	Father	Age	Mother	Age
Steffen	26	Dieter	58	Marlies	56

```
SELECT Name, Color,
       (IF Owner Owner.Name "No owner") "Owner"
FROM Dogs
```

results to:

Name	Color	Owner
Boy	white	Marlies
Streuner	grey	No owner

```
SELECT a.Name, a.Age, b.Name, b.Age FROM Person a, Person b
WHERE (> a.Age b.Age)
```

results to:

a.Name	a.Age	b.Name	b.Age
Dieter	58	Steffen	26
Marlies	56	Steffen	26
Henning	57	Steffen	26
Dieter	58	Marlies	56
Henning	57	Marlies	56
Dieter	58	Henning	57

## 1.105 MUIbase/Structure editor

Structure editor  
\*\*\*\*\*

MUIbase has two different operating modes, the record-editing mode, where you enter and browse records, and the structure-editing mode where you define the structure, that is, the tables, attributes and appearance of a project. This chapter describes the structure editor and explains how to manage the structure of a project.

To switch from record-editing to structure-editing mode select menu item Structure editor in the Project menu. This will close all windows and open the structure editor window. To switch back to record-editing mode select menu item Project - Exit structure editor or simply close the structure editor by clicking on the close button in the window title bar.

The structure editor window is divided into three parts: in the upper left part there is a field Tables for managing the tables of the project, the lower left part is occupied by a field Attributes for managing the attributes of a table, and the right part is used by a field Display for managing the project's gui elements.

Table management	How to add, change and delete tables.
Attribute management	Adding, changing, and deleting attributes.
Display management	Managing the project's gui elements.
Print structure	Getting an overview of all tables and attributes.

## 1.106 MUIbase/Table management

Table management  
=====

In the Tables field of the structure editor you can create, change, delete and sort tables.

Creating tables	How to add a table.
Changing tables	How to modify a table.
Deleting tables	How to remove a table.
Sorting tables	How to order the list of tables.

## 1.107 MUIbase/Creating tables

Creating tables  
-----

To create a new table press the New button in the Tables field. This will open the New table requester containing

- \* a string field for entering the name of the table. Each table must have a unique name that starts with an uppercase letter followed by further letters, digits or underscore characters. Non-ASCII characters like German umlauts are not allowed. Please note that in the user interface for the table it is still possible to display any strings including strings with non-ASCII characters.
- \* a field Number of records where you specify how many records the table is going to hold. Unlimited means that the table can hold any number of records, Exactly one means that the table can have only one record. The latter one is sometimes useful for controlling the project (see Tables).
- \* a field Trigger functions where you can enter the names of two functions. In the New string field you enter the name of the function that should be called whenever the user wants to create a new record, the Delete string field holds the name of the function that should be called whenever the user wants to delete a record. You can use the pop-up buttons to the right of the string fields for choosing a function name from a list of all names. If you leave a field empty then default actions are executed (records are created automatically and records are deleted after an optional safety requester). For more information on how to use the trigger functions, including the arguments that are passed to them, see Programming MUIbase.
- \* two buttons Ok and Cancel for leaving the requester.

When you are done with all settings, press the Ok button to create the new table. If you made an error somewhere, e.g. you entered an invalid name, a message window pops up giving you more information about the error you did. If everything goes fine, the New table requester will be closed and the new table is displayed in the structure editor's table list.

## 1.108 MUIbase/Changing tables

### Changing tables

-----

After you have created a new table you can still change it. Just double-click on the table's name and the Change table requester pops up. This requester is similar to the one when creating a table (see Creating tables) and allows you to make changes in any field by entering a new value.

When you are done with all changes, press the Ok button to leave the requester.

Please note that you can't change the number of records from Unlimited to Exactly one if the table already contains more than one record.

---

## 1.109 MUIbase/Deleting tables

Deleting tables  
-----

To delete a table, click on the table's name in the structure editor's table list, then press the Del button below the list. Before the table is actually deleted a safety requester pops up asking for confirmation. If you confirm this requester by pressing the Delete button, the table is deleted.

A problem occurs if the table is used somewhere in the project's program. In this case the table can't simply be deleted but all references to the table must be removed from the program. If the table you want to delete is used in the project's program then the program editor pops up and displays the first occurrence to the table. You should now modify the program such that no references to this table remain in the program. After you removed a reference you can jump to the next one by pressing the Compile button. At any point you can still cancel the whole operation by pressing the Undo button and closing the program editor.

## 1.110 MUIbase/Sorting tables

Sorting tables  
-----

For sorting the tables in the Tables field of the structure editor you have two choices. You can either order them by hand, that is, you use drag & drop to rearrange a table, or you use the Sort button below the list-view which orders the tables alphabetically.

## 1.111 MUIbase/Attribute management

Attribute management  
=====

In the Attributes field of the structure editor you can create, copy, change, delete and sort the attributes of the active table in the Tables field.

Creating attributes	How to add an attribute.
Type specific settings	Settings depending on the type of an attribute.
Label editor	Specifying label strings, e.g. for choice attributes.
Copying attributes	Making copies of an attribute.

Changing attributes	How to modify an attribute.
Deleting attributes	How to remove an attribute.
Sorting attributes	How to order the list of attributes.

## 1.112 MUIbase/Creating attributes

### Creating attributes

-----

To create a new attribute for the active table press the New button in the Attribute field. This will open the New attribute requester containing

- \* a string field for entering the name of the attribute. Each attribute in a table must have an unique name that starts with an uppercase letter followed by further letters, digits or underscore characters. Non-ASCII characters like German umlauts are not allowed. Please note that in the user interface it is still possible to display any strings including strings with non-ASCII characters for the attribute.
- \* a choice field Type where you specify the type of the attribute. For more information on attribute types, see Attribute types.
- \* a section below the Type field for specifying type specific settings. For more information about this section, see Type specific settings.
- \* a field Trigger where you can enter the name of a function that should be called whenever the user wants to change the contents of the attribute field in a record. You can use the pop-up button to the right of the string field for choosing a name from a list of all function names. If you leave the field empty then a default action is executed, that is, the entered value is simply stored in the attribute field. For more information on how to use the trigger function, including the arguments that are passed to it, see Programming MUIbase.
- \* two buttons Ok and Cancel for leaving the requester.

When you are done with all settings, press the Ok button to create the new attribute. If you made an error somewhere, e.g. you entered an invalid name, a message window pops up giving you more information about the error you did. If everything goes fine, the New attribute requester will be closed and the new attribute is displayed in the structure editor's attribute list.

## 1.113 MUIbase/Type specific settings

---

## Type specific settings

---

In the type specific section the following settings can be specified:

- \* For attributes of type string you have
  - an integer field Max length for entering the maximum string length for this attribute field.
  - a string field Initial value for specifying the value that is used for initializing the attribute field. Any string up to the specified maximum length can be entered here.
- \* For attributes of type integer, real, date, and time the type specific section offers
  - a field Initial value where you specify the value for initializing the attribute field. You can choose between NIL and other. If you select other then you should enter the initial value into the string field to the right of other.
  - a string field NIL string where you enter the string that should be displayed when the attribute field holds the NIL value.
- \* For bool attributes the type specific section contains a field Initial value where you can choose between NIL and TRUE for the initial value.
- \* The type specific section for choice attributes offers
  - a button Edit labels for opening the Edit labels window where you can enter the label strings for the choice field (see Label editor).
  - a choice field Initial value for specifying the value for initializing the attribute field.
- \* For reference attributes the type specific section contains
  - a list-view displaying all tables for specifying to which table the reference should be made. Click on the table the attribute is going to reference.
  - a field Filter. If checked the reference filter of this attribute is turned on. See Reference filter, for more information about this feature.

Reference attributes always have the NIL value as initial value.

- \* The type specific section for virtual attributes contains a string field Compute where you enter the name of a function that should be called for calculating the value of the attribute. You can use the attached pop-up button for selecting a name from a list of all function names.
-

- \* Memo and button attributes do not have any type specific settings. The initial value for memo attributes is an empty string.

## 1.114 MUIbase/Label editor

Label editor  
-----

Whenever you have to define a list of labels, e.g. the list of labels for a choice attribute, the label editor comes into place. The label editor is a window containing:

- \* a list-view displaying the current list of labels. You can click on a label to make it the active one. The active label is also displayed in the string field below the list-view. You can use drag & drop to rearrange the labels.
- \* a string field Label that displays the active label and allows changing it. The changes do only have effect after you pressed the Return key. If there is currently no active label then pressing Return inserts new labels into the list.
- \* a button New that deactivates the current label which allows entering new labels into the Label string field.
- \* a button Remove that removes the active label from the list.
- \* a button Sort for ordering the list of labels alphabetically.
- \* two buttons Ok and Cancel for leaving the label editor.

After you entered all labels or made changes to them, press the Ok button to leave the window.

## 1.115 MUIbase/Copying attributes

Copying attributes  
-----

In case you need a lot of similar attributes, it is possible to copy an attribute. Just select the desired attribute and press the Copy button below the attribute list. This opens the Copy attribute requester where the settings of the selected attribute are displayed. Change some of the fields, e.g. the name, and then press Ok for creating a copy of the attribute.

---

## 1.116 MUIbase/Changing attributes

### Changing attributes

-----

After you have created a new attribute it is still possible to change some settings of it. Just double-click on the attribute's name and the Change attribute requester pops up. This requester is similar to the one when creating an attribute (see Creating attributes) and allows you to make changes in some fields. The fields that cannot be changed, e.g. the attribute type, are displayed ghosted.

The following notes should be taken into account when changing an attribute.

- \* The type of an attribute cannot be changed. If you ever want to change the type of an attribute, it is best to create a new one of the desired type and copy the record contents from the old attribute to the new one by entering a simple MUIbase program in the query editor (see Query editor).
- \* If you change the initial value of an attribute then only new records will get the new value for initializing the record.
- \* For choice attributes you should be careful when changing its labels. The labels are only used for displaying the choice field contents, internally, numbers are stored that are used as an index into the list of labels. Thus, if you change the order of labels, you actually don't change the internal number but the label which is displayed for it! Therefore you should not change the order of labels after you created a choice attribute. Appending new labels to the end of the label list, however, doesn't make any problems. For a more flexible way of having a choice-like field where you can also change the order of labels, use a string attribute together with the List-view pop-up feature (see Attribute object editor).
- \* The referenced table of a reference attribute cannot be changed.

When you are done with all changes, press the Ok button to leave the requester.

## 1.117 MUIbase/Deleting attributes

### Deleting attributes

-----

To delete an attribute, click on it's name in the structure editor's attribute list and press the Del button below the list. Before the attribute is actually deleted a safety requester pops up asking for confirmation. If you confirm this requester by pressing the Delete button, the attribute is deleted.

---

A problem occurs if the attribute is used somewhere in the project's program. In this case the attribute can't simply be deleted but all references to it must be removed from the program. If the attribute you want to delete is used in the project's program then the program editor pops up and displays the first occurrence to this attribute. You should now modify the program such that no references to this attribute remain in the program. After you removed a reference you can jump to the next one by pressing the Compile button. At any point you can still cancel the whole operation by pressing the Undo button and closing the program editor.

## 1.118 MUIbase/Sorting attributes

Sorting attributes  
-----

For sorting the attributes in the Attributes field of the structure editor you have two choices. You can either order them by hand, that is, you use drag & drop to rearrange an attribute, or you use the Sort button below the list-view which orders the attributes alphabetically.

## 1.119 MUIbase/Display management

Display management  
=====

In the Display field of the structure editor you specify how the database elements should be arranged in the user interface. The field consists of a choice field, a list-view and several buttons.

Display field	Overview of the elements in the display field.
Panel editor	Settings for a panel object.
Attribute object editor	Settings for an attribute object.
Text editor	Settings for a text object.
Image editor	Settings for an image object.
Space editor	Settings for a space object.
Group editor	Settings for a group object.
Register group editor	Settings for a register group object.
Window editor	Settings for a window object.

## 1.120 MUIbase/Display field

Display field  
-----

The display field contains the following items:

---

- \* a choice item with two settings, Table mask and Root window. In Table mask you specify how the attributes of the active table are arranged in the user interface. In Root window you specify how the tables are arranged.
  - \* a list-view that displays the current specification of the user interface. The list is organized as a tree. Items that have an arrow to its left are composite gui objects and can be opened and closed by double-clicking on the arrow symbol. A double-click on the item itself opens a window for editing its settings. All gui objects that have the same parent object are layed out in the same way (either vertically or horizontally). How the layout is done is determined by the parent gui object: tables, panels and windows layout their elements vertically, groups layout their elements according to the settings in the group editor (see Group editor).
  - \* a button Panel for adding a panel to the table. See Panel editor, for more information about setting up a panel.
  - \* a button Add for adding the active table or the active attribute (depending on the state of the display choice field) to the display list-view. Usually tables and attributes are added to the display list-view automatically when you create them.
  - \* a button Rem for removing the active item from the display list-view. If you remove a table then the whole table form is removed from the user interface, thus you can't see the table in the project's gui. If you remove an attribute from the display list-view then the attribute doesn't appear in the project's gui. This is useful for hiding attributes.
  - \* two buttons Up and Down for moving the active item one field up, respectively down, in the display list-view.
  - \* two buttons In and Out for moving the active item one hierarchical level down or up in the display list-view.
  - \* a button Text for adding a text object to the display list-view. See Text editor, for more information about setting up a text object.
  - \* a button Image for adding an image object (see Image editor).
  - \* a button Space for putting space between the other objects (see Space editor).
  - \* a button Balance for adding a balance object to the display list-view. The balance object is useful for controlling the size of the other gui objects.
  - \* a button Group for adding a group object to the display list-view. Before you press the Group button you can multi-select the items in the list-view that should be moved into the new group. See Group editor, for more information about setting up a group object.
-

- \* a button Window for adding a new window to the display list-view. As for group objects, you can multi-select the gui objects that should be moved into the new window. For more information about setting up a window, see Window editor.

For more information about the gui elements, including their usage, see User interface.

## 1.121 MUIbase/Panel editor

Panel editor

-----

When you add a panel to a table's mask, or when you double click an existing panel object in the display list-view, the Panel window pops up. This window contains the following items:

- \* a string field Title for entering a title that should be displayed in the panel header.
- \* a string field Font with a pop-up button for selecting a font for the title. If you leave the field empty, a default font is used.
- \* a field Background with a check-mark field Default for specifying the background of the panel header. If you check the Default field then a default background is chosen. Otherwise you can click on the Background button to open a window for specifying the background setting (see the MUI docs for a description of this window).
- \* a field Num/All. If checked the number of the current record and the number of all records are displayed in the right part of the panel header.
- \* a field Filter that, if checked, adds a filter button to the panel header. With the filter button you can turn on and off the record filter of the table. If you don't check this field then the menu item Table - Change filter will also be disabled for this table, thus you can't enter a filter expression for the table. For more information about record filters, see Record filter.
- \* a field Arrows for adding two arrow buttons to the table mask. The arrow buttons allows you to browse through the records of the table. If you don't check this field then you can't browse the records of this table and all sub menu items of menu item Goto record and the menu items Search for, Search forward, and Search backward in menu Table are disabled.
- \* two buttons Ok and Cancel for leaving the window.

After you are done with all settings, click on the Ok button to close the window.

---

## 1.122 MUIbase/Attribute object editor

Attribute object editor  
-----

When you add an attribute to the display list-view, a default gui object is created for it. To change the settings of the attribute object, double click on it and the Display attribute window is opened. This window contains several items depending on the type of the attribute. The following items are included for most attribute types:

- \* a string field Title for entering a title that is displayed near the attribute object (or, for buttons, in the object itself). If you leave this field empty then no title is displayed.
  - \* a choice field Title position for specifying where the title (if any) is placed relative to the attribute object. You can choose between Left, Right, Top, and Bottom.
  - \* a string field Shortcut for entering a letter than can be used together with the Amiga key to activate the object.
  - \* a field Home. If checked, this attribute object becomes the home object. The home object is used as the object where the cursor is set on whenever a new record is allocated. This is quite useful if you always want to start entering data into the same attribute whenever you create a new record. If you mark an attribute object as the home object then all other attribute objects in the same table are unmarked.
  - \* a field Read only that, if checked, gives the object a read only status. This means that you can only read its contents but can't edit them. If you check this field then the settings of the fields Shortcut and Home are ignored.
  - \* a choice field Format for specifying how the attribute contents should be presented in the object. You can choose between Center, Left and Right for displaying the contents centered, flushed left, or flushed right.
  - \* a numerical field Weight for specifying the weight of the object. The value of this field determines how much space, relatively to the other objects, the object gets in the final window layout. Usually the value of this field only affects the horizontal size of the object as the vertical one is fixed for most attribute types.
  - \* a field Font for selecting the font used for displaying the attribute contents. If you leave the field empty, a default font is used.
  - \* a field Background with a check-mark field Default for specifying how the background of the attribute should look like. If you check the Default field then a default background is used, otherwise you can click on the Background field to open a window for specifying the background settings (see the MUI docs for a
-

description of this window).

- \* an editor field Bubble help where you can enter text that should be displayed as the bubble help information for this attribute object.
- \* two buttons Ok and Cancel for leaving the window.

If you are done with all settings, press the Ok button to leave the window.

#### Type specific settings

-----

Besides the above items, the following type-specific items are present:

- \* for attributes of type string there is an Extras page which contains:
    - a field Display picture that, if checked, attaches an image field to the string attribute for displaying the image whose filename is taken from the attribute contents. The image field is put above the string field. If you don't check this item then the settings of the fields Title at string field, Hide string field, and Size are meaningless.
    - a field Title at string field. If checked, the title of the attribute object is placed to the left of the string field, thus the image field gets more space in the window. If you don't check this field then the title is placed to the left of the image field.
    - a field Hide string field for removing the string field from the user interface. If checked, only the image field is displayed.
    - a field Size for specifying how size-handling is done for the image area. If Resize-able is active then the object can be resized and the object can become larger than the size of its image. Fixed sets the object's size to the size of the image. If the image sizes vary from record to record then the object is resized accordingly. Scroll-able adds two scrollers to the object allowing to view images that are larger than the visible region. If Scaled is activated then the image is scaled to the size of the display object.
    - a field File pop-up that, if checked, adds a pop-up button to the right of the string field. This buttons serves to open a file requester for choosing a filename.
    - a field Font pop-up for adding a pop-up button that opens a font requester. This feature has been added according to a suggestion from Ralphie. Blame him if you find this feature useless. :-)
    - a field List-view pop-up. If checked, a pop-up button is
-

attached to the right of the string field for opening a list-view pop-up where you can choose a string from a list of strings. The list of strings can be specified in the label editor which is opened after pressing the Edit labels button to the right of the List-view pop-up field. For more information about the label editor, see Label editor.

Only one of the fields File pop-up, Font pop-up and List-view pop-up can be made active (MUI programmers know why).

- a field View that, if checked, adds a button to the right of the string field for launching an external viewer with the attribute contents as argument. This can be useful if you store filenames in the attribute and want to display the contents of a file by starting an external viewer. The external viewer can be specified in menu item Preferences - External viewer (see External viewer).
  - \* for attributes of type choice there is a field Kind where you choose whether the attribute contents should be displayed by a Cycle button or by a set of Radio buttons. If you select Cycle button then you can set the title position to one of Left, Right, Top, or Bottom. If you select Radio buttons then two check mark items Frame and Horizontal allow drawing a border around the radio buttons and the specification of an horizontal layout.
  - \* for attributes of type real there is an integer field Num decimals where you can enter the number of decimals for displaying the floating point values.
  - \* for reference attributes there is an Extras page that contains the following items:
    - a list-view field Contents where you specify which contents of a referenced record should be displayed. You can multi-select several items in this list. If you select Record number then the record number of a referenced record is included in the display. The other items are the names of attributes in the referenced table.
    - a text field Items selected that displays how many items have been selected in the above list-view.
    - a field Show. If checked then the gui object for displaying the reference is created as a button. Clicking on this button will show you the referenced record in the table mask of the referenced table.
    - a field Filter that, if checked, adds a button to the right of the reference field for turning on and off the reference filter for this attribute. See Reference filter, for more information about reference filters.
  - \* for virtual attributes the attribute object editor contains:
    - a choice field Kind where you specify how the contents of the virtual attribute should be displayed. You can choose
-

between Bool that uses a check-mark field for displaying boolean values, Text that uses a text field for displaying one line of text (including date, time, and numerical values), and List which uses a list-view for displaying a list of lines (e.g. the result of a select-from-where query).

- if you set the Kind field to Text then two further fields are available: Format for specifying how the attribute contents should be presented and Num decimals for entering the number of decimals that should be used if the attribute contents are of type real.
- if the Kind field has been set to List then a field Show titles is available. If checked, the first line of the attribute contents is displayed as a title row in the list-view. Otherwise no title row is displayed and the first line is ignored.
- a field Immediate that, if checked, causes the virtual attribute to recompute its value whenever you switch the active record to another one. If not checked then the value of the virtual attribute is only computed when a MUIbase program needs its value, e.g. when you have installed a button somewhere in the user interface that reads the value of the virtual attribute when the button is pressed.

\* For buttons there are the following additional fields:

- a choice field Kind where you choose between Text button and Image button.
- if you set the button kind to Text button then further fields Title, Font, Background and Default appear for entering the text to be displayed in the button, the font used for displaying the text, and the background settings.
- if the button kind has been set to Image button then a field Image for specifying the image to be displayed and a field Size for specifying the size-handling of the image button are offered.

## 1.123 MUIbase/Text editor

Text editor

-----

When you add a text object to the display list-view, or when you change one by double clicking on it, a window Text is opened. This window contains the following items:

- \* a string field Title for entering the text that should be displayed.
  - \* a numerical field Weight for specifying the horizontal weight of
-

the text object.

- \* a choice field `Font` for specifying the font of the text. If you leave this field empty, a default font is used.
- \* two fields `Background` and `Default` for specifying the background settings of the text object.
- \* two buttons `Ok` and `Cancel` for leaving the window.

When you are done with all settings, press the `Ok` button to leave the window.

## 1.124 MUIbase/Image editor

Image editor  
-----

The image editor appears when you add a new image object or double click on an existing one. It contains the following items:

- \* a field `Weight` for specifying the weight of the image object in the final window layout.
- \* a field `Image` for specifying the image that should be displayed.
- \* a field `Size` where you specify how resize-handling should be done. If you select `Resize-able` then the image object can be resized. `Fixed` sets the object size to the size of the displayed image.
- \* two buttons `Ok` and `Cancel` for leaving the window.

When you are done with all settings, click on the `Ok` button to close the window.

## 1.125 MUIbase/Space editor

Space editor  
-----

After you have added a space object to the display list-view, you can change its default settings by double clicking on it. This will open the Space window containing the following items:

- \* a field `Delimiter` that, if checked, displays a vertical or horizontal bar (depending on the layout of the parent object) in the center of the space object. This is useful for separating parts in the window layout.
  - \* a numerical field `Weight` for specifying the weight of the object.
-

- \* two fields Background and Default for specifying the background settings.
- \* two buttons Ok and Cancel for leaving the window.

After you are done with all settings, press the Ok button to close the window.

## 1.126 MUIbase/Group editor

Group editor  
-----

After you have added a group object to the display list-view, you can change its settings by double clicking on it. This will open the Group window offering the following items:

- \* a string field Title for entering a title string that should be displayed centered above the group. If you leave this field empty then no title is displayed.
- \* a numerical field Weight for specifying the weight of this object.
- \* two fields Background and Default for specifying the background settings.
- \* a field Border that, if activated, draws a border frame around the group.
- \* a field Horizontal. If checked, the layout of the group is done horizontally and the group is listed in the display list-view as HGroup. Otherwise the group is organized vertically and the display list-view will show a VGroup for this group.
- \* a field Spacing that, if checked, adds some space between the group's child objects. Otherwise no space is put between the objects.
- \* two buttons Ok and Cancel for leaving the window.

When you are done with all settings, press the Ok button to leave the window.

## 1.127 MUIbase/Register group editor

Register group editor  
-----

Double click on a register group object in order to change its

---

settings. This will open the Register group window offering the following items:

- \* a numerical field Weight for specifying the weight of this object.
- \* two fields Background and Default for specifying the background settings.
- \* an area Labels for specifying the label of each register page. You should specify exactly the same number of labels as there are elements in the register group. For more information on how to enter and edit the labels, see Label editor.
- \* two buttons Ok and Cancel for leaving the window.

When you are done with all settings, press the Ok button to leave the window.

## 1.128 MUIbase/Window editor

Window editor  
-----

To change the settings of a window object, double click on it. This will open the window editor containing the following items:

- \* a string field Title where you enter a string that should be displayed in the window title bar and in the window button.
- \* a string field Shortcut where you enter the shortcut for activating the window button.
- \* a numerical field Weight for specifying the weight of the window button.
- \* a field Font for selecting the font of the window button. If you leave the field empty, a default font is used.
- \* two fields Background and Default for specifying the background settings of the window button.
- \* a field Disabled that, if checked, creates the window button with a disabled state, that is, you can't click on it and therefore cannot use it to open or close the window. This can be useful if you don't want the user to open windows by himself but want to open them from within a MUIbase program.
- \* two buttons Ok and Cancel for leaving the window.

When you are done with all settings, press the Ok button to close the window.

---

## 1.129 MUIbase/Print structure

Print structure  
=====

Sometimes it is useful to get an overview of all tables and attributes of a project, e.g. when you want to write a MUIbase program. You can do this by selecting menu item Project - Print structure. This will ask you for a filename where the list of tables and attributes should be written to.

The output will first list the project name, followed by all tables in this project. For each table all attributes including their types are listed.

## 1.130 MUIbase/Programming MUIbase

Programming MUIbase  
\*\*\*\*\*

This chapter describes the programming language of MUIbase, including all available functions. This chapter, however, is not intended as a general guide about programming. You should be familiar with the basics about programming and should already have written some small programs.

Program editor	Where to enter a MUIbase program.
Preprocessing	Includes, conditional compilation, and constants.
Programming language	The syntax of expressions.

### Functions

Defining commands	Function and variable definitions.
Program control functions	Loops, conditional expressions, and more.
Type predicates	Examining the type of an expression.
Type conversion functions	Converting types.
Boolean functions	AND, OR, and NOT.
Comparison functions	Comparing values of expressions.
Mathematical functions	Adding, Multiplying, etc.
String functions	Useful stuff for strings.
Memo functions	Useful stuff for memos.
List functions	List processing commands.
Input requesting functions	Asking the user for input.
I-O functions	File input and output commands.
Record functions	Useful stuff for records.
Attribute functions	Manipulating attributes.
Table functions	Manipulating tables.
Gui functions	Manipulating the user interface.
Project functions	Getting information about the project.
System functions	Operating system related functions.
Variables	Pre-defined variables.
Constants	Pre-defined constant values.

## Useful information

Functional parameters	Using functions as arguments in function calls.
Type specifiers	Types for local variables.
Semantics of expressions	The meaning of an expression.
Function triggering	How to use trigger functions.
Function index	Index of all functions.

**1.131 MUIbase/Program editor**Program editor  
=====

To enter a program for a project, open the program editor by selecting menu item Program - Edit. This will open the Edit program window containing:

- \* a text editor field where you edit the project program.
- \* a button Compile & close for compiling the program and, on successful compilation, leaving the program editor.
- \* a button Compile to compile the program. If your program contains an error somewhere then an error description is put into the window title and the cursor is set on the faulty location.
- \* a button Undo that undos all changes since the last successful compilation.

The program editor is an asynchronous requester. You can leave the window open and still work with the rest of the application. You can close the editor at any time by clicking into its window close button. If you have done changes since the last successful compilation then a safety requester pops-up asking for confirmation to close the window.

You can also compile a project's program without opening the program editor by choosing menu item Program - Compile. This can be useful if you e.g. do changes to an external include file and want to incorporate these changes in the project's program.

**1.132 MUIbase/Preprocessing**Preprocessing  
=====

MUIbase programs are pre-processed similar to a C compiler preprocessing a C source file. This section describes how to use the preprocessing directives.

---

All directives start with a hash symbol # which should be the first character on a line. Space or tab characters can appear after the initial #.

#define	Defining constants.
#undef	Un-defining constants.
#include	Including external files.
#if	Conditional compilation.
#ifdef	Conditional compilation.
#ifndef	Conditional compilation.
#elif	Conditional compilation.
#else	Conditional compilation.
#endif	Conditional compilation.

### 1.133 MUIbase/#define

```
#define  
-----
```

```
#define NAME STRING
```

Defines a new symbol with the given name and contents. The symbol STRING can be any text including spaces and ends at the end of line. If STRING does not fit on one line you can use further lines by using a backslash character \ at the end of each line (except for the last one). If the symbol NAME does occur in the remaining source code then it will be replaced by the contents of STRING.

Example: (PRINTF "X is %i" X) prints X is 1 (Occurrences of NAME in strings are not altered.)

The replacement of defined symbols is done syntactically which means that you can replace symbols with any text, e.g. you can define your own syntax like in the following example:

```
#define BEGIN (  
#define END )  
  
BEGIN defun test ()  
    ...  
END
```

The replacement string of a definition may contain another symbol that has been defined with the #define directive to allow nested definitions. However there is an upper limit of 16 nested definitions.

See also #undef, #ifdef, #ifndef.

## 1.134 MUIbase/#undef

#undef  
-----

#undef NAME

Removes the definition of symbol NAME. If NAME has not been defined, nothing happens.

See also #define, #ifdef, #ifndef.

## 1.135 MUIbase/#include

#include  
-----

#include FILENAME

Reads in the contents of FILENAME (a string surrounded with double quotes) at this location. MUIbase searches in the current directory and in the directory specified in the preferences (see Program include directory) for loading the file. The file contents are processed by the compiler as if they were part of the current source code.

An external file may include one or more other external files. However there is a limit of 16 nested #include directives. To protect files from including them more than once, you can use conditional compilation.

Be careful when moving source code to external files! Debugging and tracking down errors is much harder for external files. Move only well tested and project independent code to external files.

## 1.136 MUIbase/#if

#if  
---

#if CONST-EXPR

If the given constant expression results to non-NIL then the text up to the matching #else, #elif, or #endif is used for compilation, otherwise (the expression results to NIL) the text up to the matching #else, #elif, or #endif is discarded for compilation.

Currently you can only use TRUE and NIL as constant expressions.

See also #ifdef, #ifndef, #elif, #else, #endif.

---

### 1.137 MUIbase/#ifdef

#ifdef

-----

#ifdef NAME

If the given symbol has been defined with a #define directive then the text up to the matching #else, #elif, or #endif is used for compilation, otherwise it is discarded.

See also #if, #ifndef, #elif, #else, #endif.

### 1.138 MUIbase/#ifndef

#ifndef

-----

#ifndef NAME

If the given symbol has not been defined with a #define directive then the text up to the matching #else, #elif, or #endif is used for compilation, otherwise it is discarded.

See also #if, #ifdef, #elif, #else, #endif.

### 1.139 MUIbase/#elif

#elif

-----

#elif CONST-EXPR

Any number of #elif directives may appear between an #if, #ifdef, or #ifndef directive and a matching #else or #endif directive. The lines following the #elif directive are used for compilation only if all of the following conditions hold:

- \* The constant expression in the preceding #if directive evaluated to NIL, the symbol name in the preceding #ifdef is not defined, or the symbol name in the preceding #ifndef directive was defined.
  - \* The constant expression in all intervening #elif directives evaluated to NIL.
  - \* The current constant expression evaluates to non-NIL.
-

If the above conditions hold then subsequent `#elif` and `#else` directives are ignored up to the matching `#endif`.

See also `#if`, `#ifdef`, `#ifndef`, `#else`, `#endif`.

### 1.140 MUIbase/#else

```
#else
-----
```

```
    #else
```

This inverts the sense of the conditional directive otherwise in effect. If the preceding conditional would indicate that lines are to be included, then lines between the `#else` and the matching `#endif` are ignored. If the preceding conditional indicates that lines would be ignored, subsequent lines are included for compilation.

Conditional directives and corresponding `#else` directives can be nested. However there is a maximum nesting count limit of 16 nested conditional directives

See also `#if`, `#ifdef`, `#ifndef`, `#elif`, `#endif`.

### 1.141 MUIbase/#endif

```
#endif
-----
```

```
    #endif
```

Ends a section of lines begun by one of the conditional directives `#if`, `#ifdef`, or `#ifndef`. Each such directive must have a matching `#endif`.

See also `#if`, `#ifdef`, `#ifndef`, `#elif`, `#else`.

### 1.142 MUIbase/Programming language

```
Programming language
=====
```

MUIbase uses a programming language with a lisp-like syntax. Indeed several constructs and functions have been adopted from standard lisp. However, MUIbase is not fully compatible to standard lisp. Many functions are missing (e.g. destructive commands) and the meaning of

---

some commands is different (e.g. the return command).

Why lisp?	The advantages of lisp.
Lisp syntax	Syntax of programming language.
Kinds of programs	Different kinds of programs used in MUIbase ↔
.	.
Name conventions	Name conventions for program symbols.
Accessing record contents	Using attributes and tables in programs.
Data types for programming	Available data types for programming.
Constants	Constant expressions.
Command syntax	Syntax for describing all commands.

### 1.143 MUIbase/Why lisp?

Why lisp?

-----

The advantage of a lisp-like language is that you can program in both, a functional and an imperative way. Functional languages are getting quite popular in mathematical applications. The basic concept in functional languages is the use of expressions. Functions are defined in a mathematical way and recursion is used heavily.

Imperative programming languages (e.g. C, Pascal, Modula) use an imperative description on how to compute things. Here, the state is the basic concept (e.g. variables) and a program computes its output by going from one state to another (e.g. by assigning values to variables).

Lisp combines both techniques and therefore you can choose in which way you want to implement things. Use the one which is more appropriate for the specific problem or which you like more.

### 1.144 MUIbase/Lisp syntax

Lisp syntax

-----

A lisp expression is either a constant, a variable, or a function application. For calling functions, lisp uses a prefix notation. The function and its arguments are surrounded by parenthesis. For example, to add two values a and b, you write

```
(+ a b)
```

All expressions return a value, e.g. in the above example the sum of a and b returned. Expressions can be nested, that is, you can place an expression as a sub-expression into another one.

Function evaluation is done by using a call-by-value scheme, this

means that the arguments are evaluated first before a function is called.

If not stated otherwise, all functions are strict, that is, all arguments of a function are evaluated before the function is called. Some functions, however, are non-strict, e.g. IF, AND and OR. These functions may not evaluate all arguments.

## 1.145 MUIbase/Kinds of programs

Kinds of programs  
-----

MUIbase knows three kinds of programs. The first one is the project program kind. In a program of this kind you can define functions and global variables. The functions can be used as trigger functions for attributes. You define a project program in the program editor (see Program editor).

The second kind is the query program kind. For this kind you can enter expressions only. An expression is allowed to contain global variables and calls to functions that are defined in the project program. However, you cannot define new global variables or functions in a query program. Query programs are entered in the query editor (see Query editor).

The third kind of programs are filter expressions. Here you can only enter expressions that contain calls to pre-defined MUIbase functions. Not all pre-defined functions are available, only those that don't have a side effect, e.g. you cannot use a function that writes data to a file. Filter expressions are entered in the change filter requester (see Changing filters).

## 1.146 MUIbase/Name conventions

Name conventions  
-----

In a MUIbase program you can define symbols like functions and local or global variables. The names of these symbols must follow the following conventions:

- \* the first character of a name must be a lowercase letter. This distinguishes program symbols from table and attribute names.
  - \* The following characters can be any letters, digits or underscore characters. Other characters like German umlauts are not allowed.
-

## 1.147 MUIbase/Accessing record contents

Accessing record contents  
-----

To access tables and attributes in a MUIbase program, you have to specify a path to them. A path is a dot separated list of components where each component is the name of a table or an attribute.

Paths can either be relative or absolute. Absolute paths are specified with a table name as the first component, followed by a list of attributes that lead to the final attribute you want to access. E.g. the absolute path `Person.Name` accesses the `Name` attribute in the current record of table `Person`, or the absolute path `Person.Father.Name` accesses the `Name` attribute in the record referenced by the `Father` field (a reference attribute to table `Person`).

Relative paths already have a current table to which they are relative. For example in a filter expression the current table is the table for which you write the filter expression. The relative path for an attribute in the current table is simply the attribute name itself. For attributes that are not directly accessible from the current table but indirectly via a reference attribute, the same rules as for absolute paths apply.

It is not always clear if a specified path is a relative or an absolute one, e.g. suppose you are writing a filter expression for a table `Foo` that has an attribute `Bar` and there also exists a table `Bar`. Now entering `Bar` would be ambiguous, what is meant, the table or the attribute? Therefore all paths are first treated as relative ones. If no attribute is found for the specified path than the path is treated as global. In our example the attribute would be preferred.

But now, what if you want to access the table in the above example? Therefore the path must be given absolute. To indicate that a path is global you have to insert two colons in front of the path. In our example you would have to type `::Bar` for accessing the table.

To give you a better understanding of paths and their semantics, consider in our example that the `Bar` attribute in the `Foo` table is a reference to the `Bar` table, and the `Bar` table has an attribute `Name`. Now you can access the `Name` attribute by typing `Bar.Name` or `::Bar.Name`. Both expressions have a different meaning. `::Bar.Name` means to take the current record of table `Bar` and return the value of the `Name` attribute of this record, whereas `Bar.Name` takes the current record of table `Foo`, extracts the record reference of the `Bar` field and uses this record for getting the value of the `Name` attribute.

To make a more complete example, consider that table `Bar` has two records. One that contains `Mats` and one that contains `Steffen` in the `Name` field. The first record should be the current one. Furthermore table `Foo` has one record (the current one) whose `Bar` field refers to the second record of table `Bar`. Now `::Bar.Name` results to `Mats` and `Bar.Name` to `Steffen`.

---

## 1.148 MUIbase/Data types for programming

Data types for programming  
-----

The programming language of MUIbase knows of the following data types:

Type	Description
Bool	all expressions. Non-NIL expressions are treated as TRUE.
Integer	long integer, 32 bit, choice values are automatically converted to integers
Real	double, 64 bit
String	strings of arbitrary length
Memo	like strings but line oriented format
Date	date values
Time	time values
Record	pointer to a record
File	file descriptor for reading/writing
List	list of items, NIL is empty list.

All programming types support the NIL value.

## 1.149 MUIbase/Constants

Constants  
-----

The programming language of MUIbase can handle constant expressions which can be entered depending on the type of the expression:

Type	Description
Integer	Integer constants in the range of -2147483648 to 2147483647 can be specified as usual.
Real	Floating point constants in the range of -3.59e308 to 3.59e308 can be specified as usual, in scientific or non-scientific format. If you omit the decimal point then the number may not be treated as a real number but as an integer instead.
String	String constants are any character strings surrounded

---

by double quotes, e.g. "sample string". Within the double quotes you can enter any characters except control characters or new lines. However there are special escape codes for entering such characters:

<code>\n</code>	new line (nl)
<code>\t</code>	horizontal tabulator (ht)
<code>\v</code>	vertical tabulator (vt)
<code>\b</code>	backspace (bs)
<code>\r</code>	carriage return (cr)
<code>\f</code>	form feed (ff)
<code>\\</code>	backslash character itself
<code>\"</code>	double quote
<code>\e</code>	escape code 033
<code>\NNN</code>	character with octal code NNN
<code>\xNN</code>	character with hex code NN

Memo There are no constant values, just use a constant string instead.

Date Constant date values can be specified in one of the formats DD.MM.YYYY, MM/DD/YYYY, or YYYY-MM-DD, where DD, MM and YYYY are standing for two and four digit values representing the day, month and year of the date respectively.

Time Constant time values can be entered in the format HH:MM:SS, where HH is a two digit value in the range of 0 to 23 representing the hours, MM a two digit value in the range of 0 to 59 representing the minutes, and SS a two digit value in the range of 0 to 59 representing the seconds.

For some other pre-defined constant values, see Pre-defined constants.

## 1.150 MUIbase/Command syntax

Command syntax

-----

In the remainder of this chapter, you will find the description of all commands and functions available for programming MUIbase. The following syntax is used for describing the commands:

- \* text written in brackets [] is optional. If you omit the text inside the brackets then a default value is assumed.
- \* text separated by a vertical bar | denotes several options, e.g. a | b means that you can specify either a or b.
- \* text written in a font like VAR indicates a place-holder that can be filled with other expressions.

- \* dots ... indicate that further expressions can follow.
- \* all other text is obligatory.

## 1.151 MUIbase/Defining commands

Defining commands

=====

This section lists commands for defining functions and global variables. The commands are only available for project programs.

DEFUN	Function definitions.
DEFUN*	Function definitions.
DEFVAR	Variable definitions.

## 1.152 MUIbase/DEFUN

DEFUN

-----

DEFUN defines a function with the specified name, a list of arguments that are passed to the function, and a list of expressions to evaluate.

```
(DEFUN NAME (VARLIST) EXPR ...)
```

The name of a function must start with a lowercase character, followed by further characters, digits or underscore characters (see Name conventions).

The parameters VARLIST specifies the arguments of the function:

```
VARLIST: VAR1 ...
```

where VAR1 ... are the names for the arguments. The names must follow the same rules as the function name.

It is also possible to add type specifiers to the arguments (see Type specifiers).

The function executes the expressions EXPR, ... one by one and returns the value of the last expression. The function may call further functions including itself. A self defined function can be called like calling a pre-defined function.

For example to count the number of elements of a list, you can define the following function:

```
(DEFUN len (l)
```

```

      (IF (= 1 NIL)
        0
        (+ 1 (len (REST 1)))
      )
    )

```

Functions defined with DEFUN are listed in the pop-up list-views of table and attribute requesters (see Creating tables and Creating attributes).

This command is only available for project programs.

See also DEFUN\*, DEFVAR.

### 1.153 MUIbase/DEFUN\*

DEFUN\*

-----

DEFUN\* is the star version of DEFUN and has the same effect as DEFUN (see DEFUN). The only difference is that functions defined with DEFUN\* are not listed in the pop-up list-views when creating or changing tables and attributes. However, it is still possible to enter the function name in the corresponding string fields.

This command is only available for project programs.

See also DEFUN, DEFVAR.

### 1.154 MUIbase/DEFVAR

DEFVAR

-----

```
(DEFVAR VAR [EXPR])
```

Defines a global variable with initial value of EXPR or NIL if EXPR is missing. The name of the variables must start with a lowercase letter followed by further letters, digits or underscore characters (see Name conventions).

You can also add a type specifier to the variable name (see Type specifiers).

DEFVAR is only available for project programs. All DEFVAR commands should be placed on top of all function definitions.

Example: (DEFVAR x 42) defines a global variable x with value 42.

There are some pre-defined variables in MUIbase (see

Pre-defined variables).

See also DEFUN, DEFUN\*, LET.

## 1.155 MUIbase/Program control functions

Program control functions

=====

This section lists functions for program control, e.g. functions for defining local variables, loop functions, conditional program execution, loop control functions and more.

PROGN	Compound statement, returns last expression.
PROG1	Compound statement, returns first expression.
LET	Defining local variables.
SETQ	Setting the value of variables, attributes and tables.
SETQ*	Setting the value of variables, attributes and tables.
FUNCALL	Calling a function.
IF	If-then-else conditional program execution.
CASE	Switch-case conditional program execution.
COND	Powerful conditional program execution.
DOTIMES	Loop over a range of integer values.
DOLIST	Loop over a list.
DO	Generic loop.
FOR ALL	Loop over sets of records.
NEXT	Jumping to next loop run.
EXIT	Exiting a loop.
RETURN	Returning from a function.
HALT	Stopping program execution.
ERROR	Aborting program execution with an error message.

## 1.156 MUIbase/PROGN

PROGN

-----

To evaluate several expressions one by another the PROGN construction can be used.

```
(([EXPR ...])
```

executes EXPR ... one by one. Returns the result of the last expression (or NIL if no expression has been specified). In Lisp this construct is known as (PROGN [EXPR ...]).

Example: (1 2 3 4) results to 4.

See also PROG1.

## 1.157 MUIbase/PROG1

PROG1

-----

Another way, besides the PROGN function, to evaluate several expressions one by another is the PROG1 expression.

```
(PROG1 [EXPR ...])
```

executes EXPR ... and returns the value of the first expression (or NIL if no expression has been specified).

Example: (PROG1 1 2 3 4) results to 1.

See also PROGN.

## 1.158 MUIbase/LET

LET

---

LET defines a new block of local variables. This is useful, e.g., for defining local variables of a function. The syntax is

```
(LET (VARLIST) EXPR ...)
```

where VARLIST is a list of local variables.

```
VARLIST: VARSPEC ...
```

```
VARSPEC: (VAR EXPR) | VAR
```

Here VAR is the name of the variable and must start with a lowercase character, followed by further characters, digits or underscore characters (see Name conventions).

In the case of a (VAR EXPR) specification, the new variable is initialized with the given expression. In the other case the new variable is set to NIL.

It is also possible to add type specifiers to the variables. (see Type specifiers).

After initializing all variables the list of expressions EXPR ... are evaluated and the value of the last one is returned.

For example, the following LET expression

```
(LET ((x 0) y (z (+ x 1)))
      (+ x z)
    )
```

results to 1.

See also DOTIMES, DOLIST, DO, DEFVAR.

## 1.159 MUIbase/SETQ

SETQ

----

The SETQ function sets values to variables, attributes and tables.

```
(SETQ LVALUE1 EXPR ...)
```

Sets LVALUE1 to the value of EXPR. The dots indicate assignments for further lvalues. An lvalue is either a variable, an attribute in a table, or a table. In case of a variable, the variable must have been previously defined (e.g. by a LET expression).

Setting the value of a table means setting its program or gui record pointer: (SETQ TABLE EXPR) sets the program record pointer of TABLE to the value of EXPR, (SETQ TABLE\* EXPR) sets the gui record pointer of it and updates the display. For more information about program and gui record pointers, see Tables.

SETQ returns the value of the last expression.

Example: (SETQ a 1 b 2) assigns 1 to the variable a, 2 to the variable b and returns 2.

See also SETQ\*, LET, DEFVAR, Tables, Semantics of expressions.

## 1.160 MUIbase/SETQ\*

SETQ\*

-----

SETQ\* is the star version of SETQ (see SETQ) and has similar effects. The difference is that when assigning a value to an attribute, SETQ\* calls the trigger function of that attribute (see Attribute trigger) instead of directly assigning the value. In case no trigger function has been specified for an attribute, SETQ\* behaves like SETQ and simply assigns the value to the attribute.

Example: (SETQ\* Table.Attr 0) calls the trigger function of Table.Attr with an argument of 0.

Warning: With this function it is possible to write endless loops, e.g. if you have defined a trigger function for an attribute and this function calls SETQ\* to set a value to itself.

See also SETQ\*, LET, DEFVAR.

## 1.161 MUIbase/FUNCALL

FUNCALL

-----

FUNCALL is used to call a function.

(FUNCALL FUN-EXPR [EXPR ...])

Calls the function FUN-EXPR with the given arguments. The expression FUN-EXPR can be any expression whose value is a pre-defined or user-defined function, e.g. a variable holding the function to call. If the number of arguments is not correct, an error message is generated.

FUNCALL returns the return value of the function call or NIL if FUN-EXPR is NIL.

For more information about functional expressions, see Functional parameters.

## 1.162 MUIbase/IF

IF

--

IF is a conditional operator.

(IF EXPR1 EXPR2 [EXPR3])

The expression EXPR1 is tested. If it results to non-NIL then the value of EXPR2 is returned else the value of EXPR3 (or NIL if not present) is returned.

This function is not strict, that is, only one expression of EXPR2 or EXPR3 will be evaluated.

See also CASE, COND.

## 1.163 MUIbase/CASE

CASE

----

CASE is similar to the switch statement in the C language.

```
(CASE EXPR [CASE ...])
```

Here EXPR is the selection expression and CASE ... are pairs consisting of:

```
CASE: (VALUE [EXPR ...])
```

where VALUE is a single expression or a list of expressions and EXPR ... are the expressions to be executed if one of the case expression matches.

The CASE expression first evaluates EXPR. Then each case pair is checked whether it (or one of the expressions in the list) matches the evaluated expression. If a matching case expression is found then the corresponding expressions are executed and the value of the last expression is returned. If no case matches, NIL is returned.

Example: (CASE 1 ((2 3 4) 1) (1 2)) returns 2.

See also IF, COND.

## 1.164 MUIbase/COND

COND

----

COND is, like IF, a conditional operator.

```
(COND [(TEST-EXPR [EXPR ...]) ...])
```

COND test the first expression of each list one by one. For the first one that does not result to NIL, the corresponding expressions EXPR ... are evaluated and the value of the last expression is returned.

If all tested expressions result to NIL then NIL is returned.

Example

-----

```
(COND ((> 1 2) "1 > 2")
      ((= 1 2) "1 = 2")
      ((< 1 2) "1 < 2")
      )
```

results to "1 < 2".

See also IF, CASE.

## 1.165 MUIbase/DOTIMES

DOTIMES

-----

For simple loops the DOTIMES command can be used.

```
(DOTIMES (NAME INT-EXPR [RESULT-EXPR ...]) [LOOP-EXPR ...])
```

Here NAME is the name of a new variable which will be used in the loop. The name must start with a lowercase character, followed by further characters, digits or underscore characters (see Name conventions).

The number of times the loop is executed is given in INT-EXPR. In RESULT-EXPR ... expressions can be specified that are executed after terminating the loop. In LOOP-EXPR you specify the body of the loop, that is, the expressions that are evaluated in each loop run.

Before executing the loop, DOTIMES computes the value of INT-EXPR to determine the number of times the loop gets executed. Here INT-EXPR is evaluated only once at the start of the loop and must result to an integer value. Then DOTIMES sets the loop variable to the values of 0 to INT-EXPR-1 one by one for each loop run. First, the variable is initialized with zero and checked if it is already greater or equal to the value of EXPR. If INT-EXPR is negative or NIL or if the variable is greater or equal to the value of EXPR then the loop is terminated and the result expressions are evaluated. Otherwise the loop expressions are evaluated and the variable is incremented by one. Then the execution returns to the termination test and, possibly, does further loop runs.

The DOTIMES expression returns the value of the last result expression or NIL if no result expression has been given.

Example

-----

```
(DOTIMES (i 50 i) (PRINT i))
```

Prints the numbers from 0 to 49 and returns the value 50.

See also DOLIST, DO, FOR ALL, LET.

## 1.166 MUIbase/DOLIST

DOLIST

-----

For loops through lists the DOLIST expression can be used.

```
(DOLIST (NAME LIST-EXPR [RESULT-EXPR ...]) [LOOP-EXPR ...])
```

---

Here NAME is the name of a new variable which will be used in the loop. The name must start with a lowercase character, followed by further characters, digits or underscore characters (see Name conventions).

In LIST-EXPR you specify the list over which the loop should be executed, RESULT-EXPR ... are expressions which are evaluated after terminating the loop, end LOOP-EXPR ... build the body of the loop.

Before executing the loop, DOLIST computes the value of LIST-EXPR. This expression is evaluated only once at the start of the loop and must result to a list value. Then DOTIMES sets the loop variable to the nodes of the list one by one for each loop run. First the loop variable is initialized to the first node of the list. If the list is already empty (NIL) then the loop is terminated and the result expressions are evaluated. Otherwise the loop expressions are evaluated and the variable is set to the next node of the list. Then the execution returns to the termination test and, possibly, does further loop runs.

The DOLIST expression returns the value of the last result expression or NIL if no result expression has been given.

Example

-----

```
(DOLIST (i (SELECT * FROM Accounts)) (PRINT i))
```

Prints all records of table Accounts and returns NIL.

See also DOTIMES, DO, FOR ALL, LET.

## 1.167 MUIbase/DO

DO

--

With the DO expression arbitrary loops can be programmed.

```
(DO ([BINDING ...]) (TERM-EXPR [RESULT-EXPR ...]) [LOOP-EXPR ...])
```

where BINDING ... are the variable bindings, each of which is either:

- \* a new name for a variable (which is initialized to NIL)
- \* a list of the form: (NAME INIT [STEP]) where NAME is a name for the new variable, INIT is the initial value of the variable, and STEP is a step expression.

Furthermore, TERM-EXPR is the termination test expression, RESULT-EXPR ... are the result expressions (the default is nil) and LOOP-EXPR ... build the body of the loop.

The DO loop first initializes all local variables with the init

expressions, then tests the termination expression. If it results to TRUE, the loop is terminated and the result expressions are evaluated. The value of the last result expression is returned. Otherwise the body of the loop (LOOPEXPR ...) is executed and each variable is updated by the value of its step expression. Then the execution returns to test the terminating expression and so on.

Example

-----

```
(DO ((i 0 (+ i 1))) ((>= i 5) i) (PRINT i))
```

Prints the values 0, 1, 2, 3 and 4 and returns the value 5. Of course this is a quite complicated way to build a simple FOR loop. Therefore a simpler version exists, the DOTIMES expression.

See also DOTIMES, DOLIST, FOR ALL, LET.

## 1.168 MUIbase/FOR ALL

FOR ALL

-----

The FOR ALL expression is used to loop over a list of records.

```
(FOR ALL TABLE-LIST [WHERE WHERE-EXPR] [ORDER BY ORDER-LIST] DO EXPR ...)
```

Here TABLE-LIST is a comma-separated list of tables, WHERE-EXPR an expression to test each set of records, ORDER-LIST a comma-separated list of expressions by which the record sets are ordered, and EXPR ... are the expressions that are executed for each record set.

FOR ALL first generates a list of all record sets for which the loop body should be executed. This is done like in the SELECT expression. Please see SELECT, for more information about how this list is generated. For each element of this list the loop body EXPR ... is executed.

For example, summing up an attribute of a table can be done in the following way:

```
(SETQ sum 0)
(FOR ALL Accounts DO
  (SETQ sum (+ sum Accounts.Amount))
)
```

The FOR ALL expression returns NIL.

See also SELECT, DOTIMES, DOLIST, DO.

## 1.169 MUIbase/NEXT

NEXT  
-----

NEXT can be used for controlling DOTIMES, DOLIST, DO and FOR ALL loops.

Calling NEXT in the body of a loop will jump to the next loop iteration. This can be used for skipping non-interesting loop runs, like e.g. in the following example:

```
(FOR ALL TABLE DO
  (IF NOT-INTERESTED-IN-THE-CURRENT-RECORD (NEXT))
  ...
)
```

See also EXIT, DOTIMES, DOLIST, DO, FOR ALL.

## 1.170 MUIbase/EXIT

EXIT  
-----

EXIT can be used to terminate a loop.

```
(EXIT [EXPR ...])
```

EXIT within a loop body will terminate the loop, execute the optional expressions EXPR ..., and return the value of the last expression (or NIL in case of no expression) as the return value of the loop. Possible return expressions of the loop as for example in

```
(DOTIMES (x 10 RET-EXPR ...) ...)
```

are not executed.

You may use the EXIT function for example to end a FOR ALL loop when you found the record you are interested in:

```
(FOR ALL TABLE DO
  (IF INTERESTED-IN-THE-CURRENT-RECORD (EXIT TABLE))
  ...
)
```

See also NEXT, RETURN, HALT, DOTIMES, DOLIST, DO, FOR ALL.

## 1.171 MUIbase/RETURN

**RETURN**

-----

Within a function definition you can return to the caller by using the RETURN command.

```
(RETURN [EXPR ...])
```

terminates the function, executes the optional expressions EXPR ..., and returns the value of the last expression (or NIL in case of no expression).

**Example**

-----

```
(DEFUN find-record (name)
  (FOR ALL Table DO
    (IF (= Name name) (RETURN Table))
  )
)
```

The example searches a record whose Name attribute matches the given name. The function returns the first record found or NIL in case of no record found.

See also HALT, EXIT.

**1.172 MUIbase/HALT****HALT**

-----

HALT can be used to terminate program execution.

```
(HALT)
```

stops program execution silently.

See also ERROR, EXIT, RETURN.

**1.173 MUIbase/ERROR****ERROR**

-----

For aborting program execution with an error message the ERROR function can be used.

```
(ERROR FMT [ARG ...])
```

---

stops program execution and pops up a requester with an error message. The error message is generated from FMT and the optional arguments ARG ... like in the SPRINTF function (see SPRINTF).

See also HALT, SPRINTF.

## 1.174 MUIbase/Type predicates

Type predicates

=====

For each type a predicate is defined that returns TRUE if the supplied expression is of the specified type and NIL otherwise. The predicates are:

Predicate	Description
(STRP EXPR)	TRUE if EXPR is of type string, NIL otherwise.
(MEMOP EXPR)	TRUE if EXPR is of type memo, NIL otherwise.
(INTP EXPR)	TRUE if EXPR is of type integer, NIL otherwise.
(REALP EXPR)	TRUE if EXPR is of type real, NIL otherwise.
(DATEP EXPR)	TRUE if EXPR is of type date, NIL otherwise.
(TIMEP EXPR)	TRUE if EXPR is of type time, NIL otherwise.
(NULLP EXPR)	TRUE if EXPR is NIL (an empty list), NIL otherwise.
(CONSP EXPR)	TRUE if EXPR is a non-empty list, NIL otherwise.
(LISTP EXPR)	TRUE if EXPR is a list (may be NIL), NIL otherwise.
(RECP TABLE EXPR)	TRUE if EXPR is a record pointer of the given table. If EXPR is NIL then TRUE is returned (initial record). If TABLE is NIL then a check is done whether EXPR is a record pointer of any table.

## 1.175 MUIbase/Type conversion functions

Type conversion functions

=====

This section lists functions for converting values from one type to another one.

STR	Conversion to string.
MEMO	Conversion to memo.
INT	Conversion to integer.
REAL	Conversion to real.
DATE	Conversion to date.
TIME	Conversion to time.

## 1.176 MUIbase/STR

STR  
---

STR can be used to convert an expression into a string representation.

(STR EXPR)

converts EXPR into a string representation. The type of EXPR determines the conversion:

Type	Return string
String	The string itself.
Memo	The whole memo text in one string.
Integer	String representation of integer value.
Real	String representation of real value. If EXPR is an attribute then the number decimals specified for this attribute are used, otherwise 2 decimals are used.
Choice	Label string of the choice attribute.
Date	String representation of date value.
Time	String representation of time value.
Bool	The string "TRUE"
NIL	User defined nil string if EXPR is an attribute, the string "NIL" otherwise.
Record	String representation of record number.
Others	String representation of internal address pointer.

See also MEMO, SPRINTF.

## 1.177 MUIbase/MEMO

MEMO  
----

MEMO can be used to convert an expression into a memo.

(MEMO EXPR)

converts EXPR into a memo representation. It treats the expression like the STR function (see STR) but returns a memo text instead of a string.

See also STR.

## 1.178 MUIbase/INT

INT  
---

INT is used to convert an expression into an integer.

(INT EXPR)

converts EXPR into an integer value. Possible conversions are:

Type	Return value
String	If the whole string represents an integer value then the string is converted into an integer. Leading and following spaces are ignored. If it does not represent an integer value, NIL is returned.
Memo	Same as for string.
Integer	The value itself.
Real	If the value lies within the integer range than the real value is rounded and returned, otherwise NIL is returned.
Choice	The internal number (starting with 0) of the current label.
Date	Number of days since 01.01.0000.
Time	Number of seconds since 00:00:00.
Record	Record number.
NIL	NIL
Others	An error message is generated and program execution is aborted.

See also REAL, ASC.

---

## 1.179 MUIbase/REAL

REAL

----

REAL is used to convert an expression into a value of type real.

(REAL EXPR)

converts EXPR into a real. It treats the expression like the INT function (see INT) but returns a value of type real instead of an integer.

See also INT.

## 1.180 MUIbase/DATE

DATE

----

DATE is used to convert an expression into a date (with your best girl friend :-).

(DATE EXPR)

converts the given expression into a date (only one date guys, needs two good girl friends for getting more dates!) Possible conversions are:

Type	Return value
String	If the whole string represents a date value then the string is converted into a date value. Leading and following spaces are ignored. If it does not represent a date value, NIL is returned.
Memo	Same as for string.
Integer	A date value is generated where the given integer represents the number of days since 01.01.0000. If the integer value is too great (date value would be greater than 31.12.9999) or negative then NIL is returned.
Real	Same as for integer.
Date	The value itself.
NIL	NIL
others	An error message is generated and program execution is aborted.

## 1.181 MUIbase/TIME

TIME

----

TIME is used to convert an expression into a time value.

(TIME EXPR)

converts the given expression into a time value. Possible conversions are:

Type	Return value
String	If the whole string represents a time value then the string is converted into a time value. Leading and following spaces are ignored. If it does not represent a time value, NIL is returned.
Memo	Same as for string.
Integer	A time value is generated where the given integer represents the number of seconds since 00:00:00. The integer value is taken modulo the number of seconds per day, e.g. (TIME 86400) results to 00:00:00.
Real	Same as for integer.
Time	The value itself.
NIL	NIL
others	An error message is generated and program execution is aborted.

## 1.182 MUIbase/Boolean functions

Boolean functions

=====

This section lists boolean operators.

AND	Conjunction of boolean values.
OR	Disjunction of boolean values.
NOT	Inverting a boolean value.

## 1.183 MUIbase/AND

AND  
---

AND checks if all of its arguments are TRUE.

```
(AND [EXPR ...])
```

checks EXPR ... one by another until one expression evaluates to NIL. If all expressions evaluate to non-NIL then the value of the last expression is returned. Otherwise NIL is returned.

This function is non-strict which means that arguments of AND may not be evaluated, e.g. in (AND NIL (+ 1 2)) the expression (+ 1 2) is not evaluated since a NIL value has already been processed, however in (AND (+ 1 2) NIL) the expression (+ 1 2) gets evaluated.

See also OR, NOT.

## 1.184 MUIbase/OR

OR  
--

OR checks if all of its arguments are NIL.

```
(OR [EXPR ...])
```

checks EXPR ... one by another until one expression evaluates to non-NIL. Returns the value of the first non-NIL expression or NIL if all expressions evaluate to NIL.

This function is non-strict which means that arguments of OR may not be evaluated, e.g. in (OR TRUE (+ 1 2)) the expression (+ 1 2) is not evaluated since a non-NIL value (here TRUE) has already been processed, however in (OR (+ 1 2) TRUE) the expression (+ 1 2) gets evaluated.

See also AND, NOT.

## 1.185 MUIbase/NOT

NOT  
---

NOT is used to invert the value of a boolean expression.

```
(NOT EXPR)
```

results to TRUE if EXPR is NIL, NIL otherwise.

---

See also AND, OR.

## 1.186 MUIbase/Comparison functions

Comparison functions  
=====

In this section you will find functions for comparing values.

Relational operators	=, <>, <, >, <=, >= and their star versions.
CMP	Returns an integer representing the order.
CMP*	Extended comparison, e.g. case insensitive.

## 1.187 MUIbase/Relational operators

Relational operators  
-----

For comparing two values in a MUIbase program, use

(OP EXPR1 EXPR2)

where OP is in {=, <>, <, >, >=, <=, =\*, <>\*, <\*, >\*, >=\*, <=\*}. The star is used for special comparison (strings are compared case insensitive, records are compared by using the order defined by the user).

The following table shows all rules for comparing two values in a MUIbase program.

Type	Order relation
Integer	NIL < MIN_INT < ... < -1 < 0 < 1 < ... < MAX_INT
Real	NIL < -HUGE_VAL < ... < -1.0 < 0.0 < 1.0 < ... < HUGE_VAL
String:	NIL < "" < "Z" < "a" < "aa" < "b" < ... (case sensitive) NIL < * "" < * "a" < * "AA" < * "b" < ... (case insensitive)
Memo:	same as string
Date	NIL < 1.1.0000 < ... < 31.12.9999
Time	NIL < 00:00:00 < ... < 23:59:59
Bool	NIL < TRUE
Record:	NIL < any_record (records itself are not comparable with <)

NIL <\* rec1 <\* rec2 (order specified by user)

See also CMP, CMP\*, LIKE.

## 1.188 MUIbase/CMP

CMP

---

CMP returns an integer representing the order of its arguments.

(CMP EXPR1 EXPR2)

returns a value less than 0 if EXPR1 is smaller than EXPR2, 0 if EXPR1 is equal to EXPR2, and a value greater than 0 if EXPR1 is greater than EXPR2. For determining the order the simple (non-star) order relation as for relational operators (see Relational operators) is used.

Do not assume that the returned value will always be -1, 0, or 1!

Example: (CMP "Bike" "bIKE") results to -32.

See also CMP\*, Relational operators.

## 1.189 MUIbase/CMP\*

CMP\*

----

CMP\* is the star version of CMP. The difference is that CMP\* uses the extended ordering as defined for relational operators (see Relational operators) where strings are compared case insensitive and records are compared using the user defined record order.

Example: (CMP\* "Bike" "bIKE") results to 0.

See also CMP, Relational operators.

## 1.190 MUIbase/Mathematical functions

Mathematical functions

=====

Here, some mathematical functions are listed.

+	Adding values.
-	Subtracting values.
1+	Increasing value.
1-	Decreasing value.
*	Floating point multiplication.
/	Floating point division.
DIV	Integer division.
MOD	Integer modulo.
MAX	Maximum expression.
MIN	Minimum expression.
ABS	Absolute value of an expression.
TRUNC	Truncating decimals of a real value.
ROUND	Rounding a real value.
RANDOM	Random number generator.

## 1.191 MUIbase/add

Adding values  
-----

For adding values, use

(+ EXPR ...)

Returns the sum of the arguments EXPR .... If any argument value is NIL then the result is NIL. If the values are of type real or integer then a real (or integer) value is the result.

You may also add strings or memos. In this case the result is the concatenation of the strings/memos.

If EXPR is of type date and the rest of the arguments are integers/reals then the sum of integers/reals are interpreted as a number of days and added to EXPR. If the resulting date is out of range (smaller than 1.1.0000 or greater than 31.12.9999) then NIL is the result.

If EXPR is of type time and the rest of the arguments are integers/reals then the sum of integers/reals are interpreted as a number of seconds and added to EXPR. However the rest of the arguments can also consist of expressions of type time. The resulting time value is computed modulo 24:00:00.

Examples  
-----

Expression	Value
(+ 1 2 3)	6
(+ 5 1.0)	6.0
(+ "Hello" " " "world!")	"Hello world!"
(+ 28.11.1968 +365 -28 -9)	22.10.1969
(+ 07:30:00 3600)	08:30:00
(+ 03:00:00 23:59:59)	02:59:59

---

See also -, 1+, \*, CONCAT, CONCAT2.

## 1.192 MUIbase/sub

Subtracting values

-----

For subtracting values, use

```
(- EXPR1 EXPR2 ...)
```

Subtracts the sum of EXPR2 ... from EXPR1. Here the same rules apply as for adding values (see +), except that strings and memos can't be subtracted.

(- EXPR) has a special meaning, it returns the negative value of EXPR (integer or real), e.g. (- (+ 1 2)) results to -3.

See also +, 1-.

## 1.193 MUIbase/1+

1+

--

1+ increases an integer or real expression by one.

```
(1+ EXPR)
```

Returns the value of EXPR (integer or real) plus one. If EXPR is NIL then NIL is returned.

See also +, 1-.

## 1.194 MUIbase/1-

1-

--

1- decreases an integer or real expression by one.

```
(1- EXPR)
```

Returns the value of EXPR (integer or real) minus one. If EXPR is NIL then NIL is returned.

---

See also -, 1+.

## 1.195 MUIbase/mul

Multiplying values  
-----

For multiplying integer/real values, use

```
(* EXPR ...)
```

Returns the multiplication of the integer/real values EXPR .... If all arguments are integers then an integer is returned, otherwise the result is a value of type real.

See also +, /.

## 1.196 MUIbase/fdiv

Dividing values  
-----

For dividing integer/real values, use

```
(/ EXPR1 [EXPR2 ...])
```

Divides EXPR1 by the multiplication of the rest of the arguments. Returns a real value. On division by zero, NIL is returned.

See also \*, DIV, MOD.

## 1.197 MUIbase/DIV

DIV  
---

DIV is used for integer division.

```
(DIV INT1 INT2)
```

Returns the integer division of INT1 with INT2. For example, (DIV 5 3) results to 1.

See also /, MOD.

---

## 1.198 MUIbase/MOD

MOD  
---

MOD is used for modulo calculation.

(MOD INT1 INT2)

Returns INT1 modulo INT2. For example, (MOD 5 3) results to 2.

See also DIV.

## 1.199 MUIbase/MAX

MAX  
---

MAX returns the argument that has the largest value.

(MAX EXPR ...)

Returns the maximum value of the arguments EXPR ... (all integers or reals). If one of the expressions is NIL then NIL is returned.

See also MIN.

## 1.200 MUIbase/MIN

MIN  
---

MIN returns the argument that has the smallest value.

(MIN EXPR ...)

Returns the minimum value of the arguments EXPR ... (all integers or reals). If one of the expressions is NIL then NIL is returned.

See also MAX.

## 1.201 MUIbase/ABS

ABS  
---

---

ABS computes the absolute value of an expression.

(ABS EXPR)

Returns the absolute value of EXPR (integer or real). If EXPR is NIL then NIL is returned.

## 1.202 MUIbase/TRUNC

TRUNC

-----

TRUNC truncates decimals of a real value.

(TRUNC REAL)

Returns the largest integer (as a real number) not greater than the specified real number. If REAL is NIL then NIL is returned.

Examples: (TRUNC 26.1) results to 26, (TRUNC -1.2) results to -2.

See also ROUND.

## 1.203 MUIbase/ROUND

ROUND

-----

ROUND rounds a real value.

(ROUND REAL DIGITS)

Returns the specified real number rounded to DIGITS decimal digits. If REAL or DIGITS are NIL then NIL is returned.

Examples: (ROUND 70.70859 2) results to 70.71, (ROUND 392.36 -1) results to 392.0.

See also TRUNC.

## 1.204 MUIbase/RANDOM

RANDOM

-----

RANDOM can be used to generate random numbers.

---

(RANDOM EXPR)

Returns a random number. On the first call the random number generator is initialized with a value generated from the current time. RANDOM generates a random number in the range of 0 ... EXPR, excluding the value of EXPR itself. The type of EXPR (integer or real) is the return type. If EXPR is NIL then NIL is returned.

Examples:

-----

Example	Meaning
(RANDOM 10)	returns a value from 0 to 9,
(RANDOM 10.0)	returns a value from 0.0 to 9.99999...

## 1.205 MUIbase/String functions

String functions

=====

This section deals with functions useful for strings.

LEN	String length.
LEFTSTR	Left sub string.
RIGHTSTR	Right sub string.
MIDSTR	Individual sub string.
SETMIDSTR	Replacing a sub string.
INSMIDSTR	Inserting a string.
INDEXSTR	First occurrence of sub string, case-sensitive.
INDEXSTR*	First occurrence of sub string, case-insensitive.
INDEXBRK	First occurrence of characters, case-sensitive.
INDEXBRK*	First occurrence of characters, case-insensitive.
RINDEXSTR	Last occurrence of sub string, case-sensitive.
RINDEXSTR*	Last occurrence of sub string, case-insensitive.
RINDEXBRK	Last occurrence of characters, case-sensitive.
RINDEXBRK*	Last occurrence of characters, case-insensitive.
REPLACESTR	Replacing sub strings.
REMCHARS	Removing characters from string.
TRIMSTR	Removing leading and trailing spaces.
WORD	Extracting word in a string.
WORDS	Number of words in a string.
CONCAT	Concatenating strings.
CONCAT2	Concatenating strings.
COPYSTR	Creates copies of a string.
UPPER	Upper case string.
LOWER	Lower case string.
ASC	ASCII value of character.
CHR	Character of ASCII value.
LIKE	Comparing strings.
SPRINTF	String formatting.

## 1.206 MUIbase/LEN

LEN  
---

LEN computes the length of a string.

(LEN STR)

Returns the length of the given string or NIL if STR is NIL.

See also WORDS, LINES, MAXLEN.

## 1.207 MUIbase/LEFTSTR

LEFTSTR  
-----

LEFTSTR extracts a sub string out of a string.

(LEFTSTR STR LEN)

Returns the left part of the given string with at most LEN characters. If STR or LEN are NIL or if LEN is negative then NIL is returned.

Example: (LEFTSTR "Hello world!" 5) results to "Hello".

See also RIGHTSTR, MIDSTR, WORD, LINE.

## 1.208 MUIbase/RIGHTSTR

RIGHTSTR  
-----

RIGHTSTR extracts a sub string out of a string.

(RIGHTSTR STR LEN)

Returns the right part of the given string with at most LEN characters. If STR or LEN are NIL or if LEN is negative then NIL is returned.

Example: (RIGHTSTR "Hello world!" 6) results to "world!".

See also LEFTSTR, MIDSTR, WORD, LINE.

---

## 1.209 MUIbase/MIDSTR

MIDSTR

-----

MIDSTR extracts a sub string out of a string.

(MIDSTR STR POS LEN)

Returns a part of the given string with at most LEN characters. If LEN is NIL then the number of returning characters is not restricted. The sub string starts at the POS-th position (starting with zero). If STR or LEN are NIL or if LEN is negative then NIL is returned. If POS is out of range, that is, negative or greater than the string length, NIL is returned.

Example: (MIDSTR "Hello world!" 3 5) results to "lo wo".

See also LEFTSTR, RIGHTSTR, WORD, LINE, SETMIDSTR, INSMIDSTR.

## 1.210 MUIbase/SETMIDSTR

SETMIDSTR

-----

SETMIDSTR replaces a sub string in a string.

(SETMIDSTR STR INDEX SET)

Returns a copy of string STR where the sub string starting at INDEX is overwritten with string SET. The length of the returned string is greater or equal to the length of STR. If one of the arguments is NIL or if INDEX is out of range then NIL is returned.

Example: (SETMIDSTR "Hello world!" 6 "Melanie!") results to "Hello Melanie!".

See also INSMIDSTR, REPLACESTR.

## 1.211 MUIbase/INSMIDSTR

INSMIDSTR

-----

INSMIDSTR is used to insert a sub string into a string.

(INSMIDSTR STR INDEX INSERT)

Returns a copy of string STR where the string INSERT has been inserted at the given index. If one of the arguments is NIL or if

INDEX is out of range then NIL is returned.

Example: (INSMIDSTR "Hello world!" 6 "MUIbase-") results to "Hello MUIbase-world!".

See also SETMIDSTR, REPLACESTR.

## 1.212 MUIbase/INDEXSTR

INDEXSTR

-----

INDEXSTR searches a string for the first occurrence of a sub string.

(INDEXSTR STR SUBSTR)

Searches for the first occurrence of SUBSTR in STR. String comparison is done case-sensitive. Returns the index (starting with 0) of the sub string in STR or NIL if the sub string is not present. If one of the arguments is NIL then NIL is returned.

Example: (INDEXSTR "Hello world!" "world") returns 6.

See also INDEXSTR\*, RINDEXSTR, RINDEXSTR\*, INDEXBRK, INDEXBRK\*.

## 1.213 MUIbase/INDEXSTR\*

INDEXSTR\*

-----

INDEXSTR\* has the same effect as INDEXSTR (see INDEXSTR) except that string comparison is done case-insensitive.

See also INDEXSTR, RINDEXSTR, RINDEXSTR\*, INDEXBRK, INDEXBRK\*.

## 1.214 MUIbase/INDEXBRK

INDEXBRK

-----

INDEXBRK searches for the first occurrence of a character in a string.

(INDEXBRK STR BRKSTR)

Searches for the first occurrence of a character from BRKSTR in STR. String comparison is done case-sensitive. Returns the index (starting with 0) of the first character found in STR or NIL if no character is

---

found. If one of the arguments is NIL then NIL is returned.

Example: (INDEXBRK "Hello world!" "aeiou") returns 1.

See also INDEXBRK\*, RINDEXBRK, RINDEXBRK\*, INDEXSTR, INDEXSTR\*.

## 1.215 MUIbase/INDEXBRK\*

INDEXBRK\*

-----

INDEXBRK\* has the same effect as INDEXBRK (see INDEXBRK) except that string comparison is done case-insensitive.

See also INDEXBRK, RINDEXBRK, RINDEXBRK\*, INDEXSTR, INDEXSTR\*.

## 1.216 MUIbase/RINDEXSTR

RINDEXSTR

-----

RINDEXSTR searches a string for the last occurrence of a sub string.

(RINDEXSTR STR SUBSTR)

Searches for the last occurrence of SUBSTR in STR. String comparison is done case-sensitive. Returns the index (starting with 0) of the sub string in STR or NIL if the sub string is not present. If one of the arguments is NIL then NIL is returned.

Example: (RINDEXSTR "Do itashimashite." "shi") returns 11.

See also RINDEXSTR\*, INDEXSTR, INDEXSTR\*, RINDEXBRK, RINDEXBRK\*.

## 1.217 MUIbase/RINDEXSTR\*

RINDEXSTR\*

-----

RINDEXSTR\* has the same effect as RINDEXSTR (see RINDEXSTR) except that string comparison is done case-insensitive.

See also RINDEXSTR, INDEXSTR, INDEXSTR\*, RINDEXBRK, RINDEXBRK\*.

## 1.218 MUIbase/RINDEXBRK

RINDEXBRK  
-----

RINDEXBRK searches for the last occurrence of a character in a string.

(RINDEXBRK STR BRKSTR)

Searches for the last occurrence of a character from BRKSTR in STR. String comparison is done case-sensitive. Returns the index (starting with 0) of the last character found in STR or NIL if no character is found. If one of the arguments is NIL then NIL is returned.

Example: (RINDEXBRK "Konnichiwa" "chk") returns 6.

See also RINDEXBRK\*, INDEXBRK, INDEXBRK\*, RINDEXSTR, RINDEXSTR\*.

## 1.219 MUIbase/RINDEXBRK\*

RINDEXBRK\*  
-----

RINDEXBRK\* has the same effect as RINDEXBRK (see RINDEXBRK) except that string comparison is done case-insensitive.

See also RINDEXBRK, INDEXBRK, INDEXBRK\*, RINDEXSTR, RINDEXSTR\*.

## 1.220 MUIbase/REPLACESTR

REPLACESTR  
-----

REPLACESTR replaces sub strings by other strings.

(REPLACESTR STR SUBSTR REPLACESTR)

Replaces all occurrences of SUBSTR in STR by REPLACESTR. If any of the strings are NIL or SUBSTR is empty then NIL is returned.

Example: (REPLACESTR "From Freiburg to San Francisco" "Fr" "X") results to "Xom Xeiburg to San Xancisco".

See also SETMIDSTR, INSMIDSTR, REMCHARS.

## 1.221 MUIbase/REMCHARS

---

REMCHARS

-----

REMCHARS removes characters from a string.

(REMCHARS STR CHARS-TO-REMOVE)

Returns a copy of STR where all characters of CHARS-TO-REMOVE are removed from. If STR or CHARS-TO-REMOVE are NIL then NIL is returned.

Example: (REMCHARS YOUR-STRING " \t\n") removes all spaces, tabs and newline characters from YOUR-STRING.

See also REPLACESTR, TRIMSTR.

## 1.222 MUIbase/TRIMSTR

TRIMSTR

-----

TRIMSTR removes leading and trailing spaces from a string.

(TRIMSTR STR)

Returns a copy of STR where all leading and trailing spaces have been removed.

Example: (TRIMSTR " I wrecked Selma's bike. ") results to "I wrecked Selma's bike."

See also REMCHARS.

## 1.223 MUIbase/WORD

WORD

-----

WORD returns a word of a string.

(WORD STR NUM)

Returns the NUM-th word (starting with zero) of the given string. Words in a string are non-empty sub strings separated by space-like characters (e.g. space, tab or newline characters).

If STR or NUM are NIL, or if NUM is out of range, that is, less than zero or greater or equal to the number of words, then NIL is returned.

Example: (WORD "Therefore, I lend Selma my bike." 3) results to

---

"Selma".

See also WORDS, LINE, LEFTSTR, RIGHTSTR, MIDSTR.

## 1.224 MUIbase/WORDS

WORDS

-----

WORDS counts the number of words in a string.

(WORDS STR)

Returns the number of words of the given string or NIL if STR is NIL. Words are sub strings separated by space-like characters (e.g. space, tab, or newline characters).

Example: (WORDS "Actually, it wasn't really my bike.") results to 6.

See also WORD, LINES, LEN.

## 1.225 MUIbase/CONCAT

CONCAT

-----

CONCAT concatenates strings.

(CONCAT [STR ...])

Returns the concatenation of the given list of strings where space characters have been inserted between the strings. If one of the strings is NIL, or the list is empty, then NIL is returned.

Example: (CONCAT "I" "thought" "it" "was" "an" "abandoned" "bike.") results to "I thought it was an abandoned bike."

See also CONCAT2, +, COPYSTR, SPRINTF.

## 1.226 MUIbase/CONCAT2

CONCAT2

-----

CONCAT2 concatenates strings.

(CONCAT2 INSERT [STR ...])

Returns the concatenation of the given list of strings. The strings will be separated with the given INSERT string. If INSERT is NIL, one of the strings is NIL, or the list is empty, then NIL is returned.

Example: (CONCAT2 "! " "But" "it" "wasn't!") results to "But! it! wasn't!".

See also CONCAT, +, COPYSTR, SPRINTF.

## 1.227 MUIbase/COPYSTR

COPYSTR

-----

COPYSTR creates copies of a string.

(COPYSTR STR NUM)

Returns a string consisting of NUM times the string STR. If STR is NIL, NUM is NIL or less than zero, NIL is returned.

Example: (COPYSTR "+-" 5) results to "+-+-+-+--"

See also CONCAT, CONCAT2, +, SPRINTF.

## 1.228 MUIbase/UPPER

UPPER

-----

UPPER converts a string to upper case.

(UPPER STR)

Returns a copy of the given string where all characters are converted to upper case. If STR is NIL then NIL is returned.

Example: (UPPER "Selma found a letter attached to my bike.") results to "SELMA FOUND A LETTER ATTACHED TO MY BIKE.".

See also LOWER.

## 1.229 MUIbase/LOWER

LOWER

-----

LOWER converts a string to lower case.

(LOWER STR)

Returns a copy of the given string where all characters are converted to lower case. If STR is NIL then NIL is returned.

Example: (LOWER "The letter was from Silke.") results to "the letter was from silke.".

See also UPPER.

### 1.230 MUIbase/ASC

ASC

---

ASC converts a character to its ASCII code.

(ASC STR)

Returns the ASCII code of the first character of STR. If STR is empty, 0 is returned. If STR is NIL, NIL is returned.

Example: (ASC "A") results to 65.

See also CHR, INT.

### 1.231 MUIbase/CHR

CHR

---

CHR converts an ASCII code to a character.

(CHR INT)

Returns a string containing the character with ASCII code INT. If INT is 0, an empty string is returned. If INT is NIL or not in the range of ASCII characters (0..255) then NIL is returned.

Example: (CHR 67) results to "C".

See also ASC, STR.

---

## 1.232 MUIbase/LIKE

LIKE  
-----

LIKE compares strings.

(LIKE STR1 STR2)

Returns TRUE if STR1 matches STR2, NIL otherwise. The string STR2 may contain the joker characters '?' and '\*' where '?' matches exactly one character and '\*' matches a string of any length. String comparison is done case insensitive.

Example: (LIKE "Silke has been in France for one year." "\*France\*") results to TRUE.

See also Comparison functions.

## 1.233 MUIbase/SPRINTF

SPRINTF  
-----

SPRINTF formats a string with various data.

(SPRINTF FMT [EXPR ...])

SPRINTF takes a series of arguments, converts them to strings, and returns the formatted information as one string. The string FMT determines exactly what gets written to the return string and may contain two types of items: ordinary characters which are always copied verbatim and conversion specifiers which direct SPRINTF to take arguments from its argument list and format them. Conversion specifiers always begin with a % character.

Conversion specifiers always take the following form:

%[FLAGS][WIDTH][.PRECISION]TYPE

where

- \* The optional FLAGS field controls output justification, sign character on numerical values, decimal points, and trailing blanks.
  - \* The optional WIDTH field specifies the minimum number of characters to print (the field width), with padding done with blanks or zeros.
  - \* The optional PRECISION field specifies either the minimum number of digits to be printed for types integer, string, bool, date and time, or the number of characters after the decimal point to be printed for values of type real.
-

- \* The TYPE field specifies the actual type of the argument that `SPRINTF` will be converting, such as string, integer, real, etc.

Note that all of the above fields are optional except for TYPE. The following tables list the valid options for these fields.

#### Flags field

-----

- : The result is left justified, with padding done on the right using blanks. By default when - is not specified, the result is right justified with padding on the left with 0's or blanks.
- +: The result will always have a - or + character prepended to it if it is a numeric conversion.
- space: Positive numbers begin with a space instead of a + character, but negative values still have a prepended -.

#### Width field

-----

- N: A minimum of N characters are output. If the conversion has less than N characters, the field is padded with blanks.
- \*: The width specifier is supplied in the argument list as an integer or real value, before the actual conversion argument. This value is limited to the range of 0 to 999.

#### Precision field

-----

- .N: For integer, string, bool, date and time values, N is the number of characters written from the converted item. For conversions of real values, N specifies the number of digits after the decimal point.
- .\*: The precision is supplied in the argument list as an integer or real value, before the actual conversion argument. This value is limited to the range of 0 to 999.

#### Type field

-----

- b: Converts a bool parameter to "TRUE" or "NIL".
- i: Converts an integer value.

- e:** Converts a real number using the format [-]d.ddde+dd. Exactly one digit is before the decimal point, followed by an e, followed by an exponent. The number of digits after the decimal point is determined by the precision field, or is 2 if precision is not specified. The decimal point will not appear if precision is 0.
- f:** Converts a real value using the format [-]ddd.ddd. The number of digits after the decimal point is determined by the precision field, or is 2 if precision is not specified. The decimal point will not appear if precision is 0.
- s:** Writes a string value until either the end of string is reached or the number of characters written equals the precision field.
- d:** Converts a date value.
- t:** Converts a time value.
- %:** The character % is written, and no argument is converted.

SPRINTF returns the formatted string or NIL in case FMT is NIL.

#### Examples

-----

Call	Result
(SPRINTF "Hello")	"Hello"
(SPRINTF "%s" "Hello")	"Hello"
(SPRINTF "%10s" "Hello")	" Hello"
(SPRINTF "%-10.10s" "Hello")	"Hello "
(SPRINTF "%010.3s" "Hello")	" Hello"
(SPRINTF "%-5.3b" TRUE)	"TRU "
(SPRINTF "%i" 3)	"3"
(SPRINTF "%03i" 3)	"003"
(SPRINTF "% 0+- 5.3i" 3)	" 003 "
(SPRINTF "%f" 12)	"12.00"
(SPRINTF "%10e 12.0)	" 1.20e+1"
(SPRINTF "%+-10.4f" 12.0)	"+12.0000 "
(SPRINTF "%10.5t" 12:30:00)	" 12:30"
(SPRINTF "%d" 28.11.1968)	"28.11.1968"
(SPRINTF "He%s %5.5s!" "llo" "world champion ship")	"Hello world!"

See also PRINTF, FPRINTF, STR, +, CONCAT, CONCAT2, COPYSTR.

## 1.234 MUIbase/Memo functions

Memo functions

=====

This section deals with functions useful for memos.

LINE	Extracting a line in a memo.
LINES	Number of lines in a memo.
MEMOTOLIST	Converting memo to list.
LISTTOMEMO	Converting list to memo.
FILLMEMO	Filling in a memo.
FORMATMEMO	Formatting a memo.
INDENTMEMO	Indenting a memo.

## 1.235 MUIbase/LINE

LINE

----

LINE extracts a line in a memo.

(LINE MEMO NUM)

Returns the NUM-th line (starting with zero) of the given memo. The line string will not have a terminating newline character. If MEMO or NUM are NIL or if NUM is out of range, that is, less than zero or greater or equal to the number of lines, then NIL is returned.

See also LINES, WORD.

## 1.236 MUIbase/LINES

LINES

-----

LINES returns the number of lines in a memo.

(LINES MEMO)

Returns the number of lines of the given memo or NIL if MEMO is NIL.

See also LINE, WORDS, LEN.

## 1.237 MUIbase/MEMOTOLIST

## MEMOTOLIST

-----

MEMOTOLIST converts a memo to a list of strings.

(MEMOTOLIST MEMO)

Converts the given memo to a list. If MEMO is NIL then NIL is returned, otherwise a list is generated where each element contains one line of the memo.

Example: (MEMOTOLIST "My insurance\npays for\nthe wrecked bike.") results to ( "My insurance" "pays for" "the wrecked bike." ).

See also LISTTOMEMO.

## 1.238 MUIbase/LISTTOMEMO

## LISTTOMEMO

-----

LISTTOMEMO converts a list to a memo.

(LISTTOMEMO LIST)

Converts the given list to a memo. If LIST is NIL then NIL is returned, otherwise a memo is generated where each line consists of the string representation of the corresponding list element.

Example: (LISTTOMEMO (LIST "Silke" "lends me" "'my' bike" "till" "01.09.1998)) results to: "Silke\nlends me\n'my' bike\ntill\n01.09.1998".

See also MEMOTOLIST.

## 1.239 MUIbase/FILLMEMO

## FILLMEMO

-----

FILLMEMO fills in a memo with the results of expressions.

(FILLMEMO MEMO)

Creates a copy of the given memo where all substrings of the form \$(EXPR) are replaced by their results after evaluation.

Example: (FILLMEMO "(+ 1 1) is \$(+ 1 1).") results to "(+ 1 1) is 2."

Please use only small expressions in the memo as debugging and tracking down errors is not easy here.

See also `FORMATMEMO`, `INDENTMEMO`.

## 1.240 MUIbase/FORMATMEMO

`FORMATMEMO`

-----

`FORMATMEMO` formats a memo.

(`FORMATMEMO` `MEMO` `WIDTH` `FILL`)

Formats `MEMO` to a memo with lines not longer than `WIDTH` characters. If `FILL` is non-NIL then spaces are used to pad the lines up to exactly `WIDTH` characters. The memo is processed section-wise. A section starts at the first non-white-space character. The line holding this character and all following lines up to the first line whose first character is a white-space are counted to this section. The whole section is then formatted word-wise, that is, as many words are put in one line as there is space for.

See also `FILLMEMO`, `INDENTMEMO`.

## 1.241 MUIbase/INDENTMEMO

`INDENTMEMO`

-----

`INDENTMEMO` indents a memo by putting spaces to the left.

(`INDENTMEMO` `MEMO` `INDENT`)

Returns a copy of the given memo where each line is indented by `INDENT` space characters. If `MEMO` or `INDENT` is NIL then NIL is returned. If `INDENT` is negative, a value of 0 is used.

See also `FILLMEMO`, `FORMATMEMO`.

## 1.242 MUIbase/List functions

List functions

=====

This section lists functions for processing lists.

---

CONS	Elementary list constructor.
LIST	Generating a list of elements.
LENGTH	Getting number of list elements.
FIRST	Extracting first element of a list.
REST	Getting remainder of a list.
LAST	Extracting last element of a list.
NTH	Extracting n-th element of a list.
APPEND	Concatenating lists.
REVERSE	Reversing a list.
MAPFIRST	Applying a function to all list elements.
SORTLIST	Sorting elements of a list.
SORTLISTGT	Sorting elements of a list.

## 1.243 MUIbase/CONS

CONS

----

CONS builds a pair of expressions.

```
(CONS ELEM LIST)
```

Constructs a new list. The first element of the new list is ELEM, the rest are the elements of LIST (which should be a list or NIL). The list LIST is not copied, only a pointer is used to reference it!

Example: (CONS 1 (CONS 2 NIL)) results to ( 1 2 ).

The elements of a list can be of any type, e.g. it's also possible to have a list of lists (e.g. see SELECT). The CONS constructor can also be used to build pairs of elements, e.g. (CONS 1 2) is the pair with the two integers 1 and 2.

See also LIST, FIRST, REST.

## 1.244 MUIbase/LIST

LIST

----

LIST generates a list out of its arguments.

```
(LIST [ELEM ...])
```

takes the arguments ELEM ... and generates a list of it. This is equivalent to calling (CONS ELEM (CONS ... NIL)). Note that NIL alone stands for an empty list.

See also CONS, LENGTH.

## 1.245 MUIbase/LENGTH

LENGTH

-----

LENGTH determines the length of a list.

(LENGTH LIST)

returns the length of the given list.

Example: (LENGTH (LIST "a" 2 42 3)) results to 4.

See also LIST.

## 1.246 MUIbase/FIRST

FIRST

-----

FIRST extracts the first element in a list.

(FIRST LIST)

returns the first element of the given list. If LIST is empty (NIL) then NIL is returned.

See also REST, LAST, NTH, CONS.

## 1.247 MUIbase/REST

REST

-----

REST returns the sub-list after the first element of a list.

(REST LIST)

returns the rest of the given list (the list without the first element). If LIST is empty (NIL) then NIL is returned.

Example: (REST (LIST 1 2 3)) results to ( 2 3 ).

See also FIRST, CONS.

---

## 1.248 MUIbase/LAST

LAST  
----

LAST extracts the last element in a list.

(LAST LIST)

Returns the last element of the given list or NIL if LIST is NIL.

See also FIRST, NTH.

## 1.249 MUIbase/NTH

NTH  
---

NTH extracts the n-th element of a list.

(NTH N LIST)

Returns the N-th element of the given list (starting with 0) or NIL if the element doesn't exist.

See also FIRST, LAST.

## 1.250 MUIbase/APPEND

APPEND  
-----

APPEND concatenates lists.

(APPEND [LIST ...])

returns the concatenation of LIST ....

Example: (APPEND (list 1 2) (list 3 4) (list 5)) results to ( 1 2 3 4 5 ).

See also LIST.

## 1.251 MUIbase/REVERSE

---

## REVERSE

-----

REVERSE reverses a list.

```
(REVERSE LIST)
```

returns the reverse list.

Example: (REVERSE (list 1 2 3)) results to ( 3 2 1 ).

## 1.252 MUIbase/MAPFIRST

## MAPFIRST

-----

MAPFIRST applies a function to all list elements.

```
(MAPFIRST FUNC LIST [...])
```

Builds a list whose elements are the result of the specified function called with the arguments of the given list elements one by one. The length of the returned list is as long as the length of the longest specified list. If one of the specified lists is too short then the list is padded with NIL elements.

## Examples

-----

Expression	Value
(MAPFIRST 1+ (LIST 1 2 3))	( 2 3 4 )
(MAPFIRST + (LIST 1 2 3) (LIST 2 3))	( 3 5 NIL )

## 1.253 MUIbase/SORTLIST

## SORTLIST

-----

SORTLIST sorts the elements of a list.

```
(SORTLIST FUNC LIST)
```

Returns a copy of the specified list that has been sorted using the function FUNC for ordering. The order function must take two arguments one for each element and return an integer value less than zero if the first element is smaller than the second one, a value greater than zero if the first element is greater than the second one, and a value of zero if the two elements are equal.

Example for a string comparing function usable for sorting:

```
(DEFUN cmp_str (x y)
  (COND
    ((< x y) -1)
    (> x y) 1)
  (TRUE 0)
  )
)
```

Now you can sort a list by calling:

```
(SORTLIST cmp_str (LIST "hi" "fine" "great" "ok"))
```

which results to ( "fine" "great" "hi" "ok" ).

See also SORTLISTGT, MAPFIRST.

## 1.254 MUIbase/SORTLISTGT

SORTLISTGT

-----

SORTLISTGT sorts the elements of a list.

```
(SORTLISTGT GTFUNC LIST)
```

Like SORTLIST but here you specify an order function that returns a value not equal to NIL if the first element is greater then the second one, and NIL otherwise.

Example: (SORTLISTGT > (LIST "hi" "fine" "great" "ok")) result to ( "fine" "great" "hi" "ok" ).

See also SORTLIST, MAPFIRST.

## 1.255 MUIbase/Input requesting functions

Input requesting functions

=====

For requesting information from the user, the following functions can be used.

ASKFILE	Requesting a filename.
ASKDIR	Requesting a directory name.
ASKSTR	Requesting a string.
ASKINT	Requesting an integer.

ASKCHOICE     Requesting one item out of many items.  
ASKCHOICESTR   Requesting a string, offering predefined ones.  
ASKOPTIONS     Requesting some items out of many items.  
ASKBUTTON     Requesting the user to press a button.  
ASKMULTI       Requesting several informations.

## 1.256 MUIbase/ASKFILE

ASKFILE

-----

ASKFILE prompts the user for entering a filename.

(ASKFILE TITLE OKTEXT DEFAULT SAVEMODE)

Pops up a file-requester for entering a filename. The window title can be set in `TITLE`, the text of the Ok button in `OKTEXT`, and the initial filename in `DEFAULT`. You may specify `NIL` for one of them to use default values. The last argument `SAVEMODE` (bool) allows setting the file-requester to a save mode. This mode should be used when asking for a filename to write something to.

ASKFILE returns the entered filename as string or `NIL` in case the user canceled the requester.

See also `ASKDIR`, `ASKSTR`.

## 1.257 MUIbase/ASKDIR

ASKDIR

-----

ASKDIR prompts the user for entering a directory name.

(ASKDIR TITLE OKTEXT DEFAULT SAVEMODE)

Pops up a file-requester for entering a directory name. The arguments are used in the same way as in `ASKFILE` (see `ASKFILE`).

ASKDIR returns the entered directory name as string or `NIL` in case the user canceled the requester.

See also `ASKFILE`, `ASKSTR`.

## 1.258 MUIbase/ASKSTR

ASKSTR

-----

ASKSTR prompts the user for entering a string.

(ASKSTR TITLE OKTEXT DEFAULT MAXLEN)

Pops up a requester asking for a string to enter. The window title, the text of the Ok button, and the initial value can be set in TITLE, OKTEXT, and DEFAULT respectively (strings or NIL for default values), MAXLEN determines the maximum characters the user can enter.

ASKSTR returns the entered string or NIL in case the user canceled.

See also ASKFILE, ASKDIR, ASKCHOICESTR, ASKINT.

## 1.259 MUIbase/ASKINT

ASKINT

-----

ASKINT prompts the user for entering an integer value.

(ASKINT TITLE OKTEXT DEFAULT MIN MAX)

Pops up a requester asking for an integer to enter. The window title and the text of the Ok button can be specified in TITLE and OKTEXT (strings or NIL for default values). In DEFAULT you pass the initial integer value or NIL to start with an empty editing field. In MIN and MAX you can set the integer range. Entered values outside this range are not accepted by the requester. Use NIL for default min and max values.

ASKINT returns the entered integer or NIL if the user canceled the requester.

See also ASKSTR.

## 1.260 MUIbase/ASKCHOICE

ASKCHOICE

-----

ASKCHOICE prompts the user to select one item out of many items.

(ASKCHOICE TITLE OKTEXT CHOICES DEFAULT)

Pops up a requester allowing the user to choose one item out of many items. You can set the window title and the text of the Ok button in

---

TITLE and OKTEXT (strings or NIL for default values). In CHOICES you specify a list of choices from which the user can choose one. The list items can be any expressions that can be converted to strings. The initial choice value can be set in DEFAULT which must be an integer index into the choice list (starting with index 0 for the first item). Use NIL for no initial choice.

ASKCHOICE returns the index of the chosen item or NIL if the user canceled the requester.

Example

-----

```
(LET ((items (LIST "First Entry" 2 3.14 "Last entry"))) index)
  (SETQ index (ASKCHOICE "Choose one item" "Ok" items NIL))
  (IF index
    (PRINTF "User chose item num %i with contents <%s>\n"
      index (STR (NTH index items))
    )
  )
)
```

See also ASKCHOICESTR, ASKOPTIONS.

## 1.261 MUIbase/ASKCHOICESTR

ASKCHOICESTR

-----

ASKCHOICESTR prompts the user for entering a string value offering several pre-defined ones.

```
(ASKCHOICESTR TITLE OKTEXT STRINGS DEFAULT)
```

Pops up a requester allowing the user to choose one string out of many strings or to enter any other string in a separate string field. You can set the window title and the text of the Ok button in TITLE and OKTEXT (strings or NIL for default values). In STRINGS you specify a list of strings from which the user can choose one. The initial value of the string field can be set in DEFAULT (string or NIL for an empty string field).

ASKCHOICESTR returns the chosen string or NIL if the user canceled the requester.

Example

-----

```
(LET ((strings (LIST "Claudia" "Mats" "Ralphie"))) likebest)
  (SETQ likebest
    (ASKCHOICESTR "Who do you like the best?" "Ok" strings "My collies!")
  )
  (IF likebest (PRINTF "User chose <%s>\n" likebest))
)
```



```
(ASKBUTTON TITLE TEXT BUTTONS CANCELTEXT)
```

Pops up a requester with the specified window title (string or NIL for a default title) and specified description text (string or NIL for no text). The function waits until the user presses one of the buttons specified in BUTTONS (list of strings) or the Cancel button. The text of the cancel button can be set in CANCELTEXT. If you specify NIL here then a default text based on the number of buttons you specified is used.

ASKBUTTON returns the number of the pressed button (starting with 0 for the first (leftmost) button) or NIL if the user pressed the Cancel button.

Examples

-----

```
(LET ((buttons (LIST "At home" "In bed" "In front of my Amiga"))) index)
  (SETQ index (ASKBUTTON "Please answer:"
    "Where do you want to be tomorrow?" buttons "Don't know"))
  )
(IF index
  (PRINTF "User chose: <%s>\n" (NTH index buttons))
  )
)
```

```
(ASKBUTTON "Info"
  "You have been playing with your\nAmiga for five hours now.\nGo to bed!"
  NIL NIL
)
```

See also ASKCHOICE.

## 1.264 MUIbase/ASKMULTI

ASKMULTI

-----

ASKMULTI prompts the user to enter various kinds of information.

```
(ASKMULTI TITLE OKTEXT ITEMLIST)
```

ASKMULTI is a multi-purpose requester. It opens a window with the specified title, a set of gui objects for editing data, and two buttons (Ok and Cancel) for terminating the requester. The text of the Ok button can be set in OKTEXT (string or NIL for default text). The set of gui objects are specified in ITEMLIST which is a list of items where each item has one of the following forms:

```
(LIST TITLE "String" INITIAL [HELP])    for editing one line of text,
(LIST TITLE "Memo" INITIAL [HELP])      for editing multi-line text,
(LIST TITLE "Integer" INITIAL [HELP])    for editing an integer,
(LIST TITLE "Real" INITIAL [HELP])       for editing a real,
```

```

(LIST TITLE "Date" INITIAL [HELP])      for editing a date,
(LIST TITLE "Time" INITIAL [HELP])     for editing a time,
(LIST TITLE "Bool" INITIAL [HELP])     for a boolean field,
(LIST TITLE "Choice" INITIAL
  (LIST CHOICE ...) [HELP]
)                                         for a choice field.
(LIST TITLE "ChoiceList" INITIAL
  (LIST CHOICE ...) [HELP]
)                                         for selecting one item of a list.
(LIST TITLE "Options" INITIAL
  (LIST OPTION ...) [HELP]
)                                         for selecting several items of a list.
NON-LIST-EXPR                            for static text

```

The title (string or NIL for no title) will be displayed to the left of the gui object. If the initial value is NIL then a default value is used (e.g. an empty text field). For choice fields the initial value must be the index (starting with 0) for the initial active entry, for choice list fields the initial value may be NIL (no item is activated), and for options fields the initial value must be a list of integers representing the indices (starting with 0) of the items that are initially selected. The optional help field (string) can be used for giving more information to the user about the usage of the field.

ASKMULTI returns a list of result values which the user has edited and acknowledged by pressing the Ok button. Each result value of a field has the same format as the one for the initial value, e.g. for a choice list field the result value is the index of the selected item (or NIL if no item has been selected), or for an options field the result value is a list of integers representing the indices of the selected items. For static text a value of NIL is returned.

E.g. if you have specified a date field, a static text field, a choice field, an options field, and a string field with initial value "world", and the user has entered 11.11.1999, selected the choice entry with index number 2, selected the 3rd and 4th item in the options field, and left the string field untouched then the function returns the list ( 11.11.1999 NIL 2 ( 3 4 ) "world" ).

If the user cancels the requester, NIL is returned.

#### Example

-----

```

(ASKMULTI "Please edit:" NIL (LIST
  (LIST "N_ame" "String" "")
  (LIST "_Birthday" "Date" NIL)
  (LIST "_Sex" "Choice" 0 (LIST "male" "female"))
  (LIST "_Has car?" "Bool" NIL)
  (LIST "_Likes", "Options" (LIST 0 2)
    (LIST "Beer" "Wine" "Whisky" "Wodka" "Schnaps"))
  ))
)

```

Please see also the project AskDemo.mb for further examples.

## 1.265 MUIbase/I-O functions

I/O functions

=====

This sections lists functions and variables for input and output (e.g. printing) of data.

FOPEN	Opening a file for reading/writing.
FCLOSE	Closing a file.
stdout	Standard output file handle.
PRINT	Printing an expression to stdout.
PRINTF	Formatted printing to stdout.
FPRINTF	Formatted printing to a file.
FERROR	Error checking of a file.
FEOF	Detecting end of file.
FSEEK	Setting position in a file.
FTELL	Getting current position in a file.
FGETCHAR	Reading the next input character from a file.
FGETCHARS	Reading several input characters from a file.
FGETSTR	Reading a string from a file.
FGETMEMO	Reading a memo from a file.
FPUTCHAR	Writing a character to a file.
FPUTSTR	Writing a string to a file.
FPUTMEMO	Writing a memo to a file.
FFLUSH	Flushing a file.

## 1.266 MUIbase/FOPEN

FOPEN

-----

FOPEN opens a file for reading/writing.

(FOPEN FILENAME MODE)

Opens the file specified by FILENAME (string). The MODE parameter (string) controls the access mode. Use "w" to open a file for writing, "a" to append to a file, and "r" for reading from a file. You may also use other flags (or combination of flags) like "r+" for reading and writing. There is no check done that tests if you have specified a valid flag string. However if the file can't be opened, NIL is returned.

FOPEN returns a file handle on success. On failure NIL is returned. If FILENAME or MODE are NIL then NIL is returned.

Example: (FOPEN "PRT:" "w") opens and returns a file handle to the printer.

See also FCLOSE, stdout, FFLUSH.

## 1.267 MUIbase/FCLOSE

FCLOSE

-----

FCLOSE closes a file.

(FCLOSE FILE)

Closes the given file. Returns 0 on success, NIL if an error has occurred. If FILE is NIL then 0 is returned (no error). After closing a file accessing the file handle is an illegal operation and results in aborting program execution with an error message.

See also FOPEN, FFLUSH.

## 1.268 MUIbase/stdout

stdout

-----

The global variable stdout holds the file handle to MUIbase's standard output file. The output filename can be set in menu item Program - Output file. You may use PRT: here for sending the output to your printer, or CON:////Output/CLOSE/WAIT for printing into a console window.

On the first access of this variable (either directly, e.g. by calling (FPRINTF stdout ...), or indirectly, e.g. by calling (PRINTF ...), this file is opened. The file is not pre-opened on program execution. This avoids opening the file when no output is generated, e.g. when you simply want to do some calculations and change some record contents.

If MUIbase can't open the program output file then execution is aborted and an error message is generated.

See also FOPEN, PRINTF.

## 1.269 MUIbase/PRINT

PRINT

-----

PRINT converts an expression to a string and prints it.

(PRINT ELEM)

Converts the value of ELEM to a readable string and prints it to stdout. This function mainly exists for debug purposes.

---

See also `PRINTF`, `stdout`.

## 1.270 MUIbase/PRINTF

`PRINTF`

-----

`PRINTF` prints a formatted string.

```
(PRINTF FORMAT [EXPR ...])
```

Formats a string using the given format string and arguments and prints it to `stdout`. Formatting is done like in `SPRINTF` (see `SPRINTF`).

`PRINTF` returns the number of output characters or `NIL` on failure. If `FORMAT` is `NIL` then `NIL` is returned.

Example: `(PRINTF "%i days and %i week" 7 1)` prints the string "7 days and 1 week" to `stdout` and returns 17.

See also `PRINT`, `FPRINTF`, `stdout`.

## 1.271 MUIbase/FPRINTF

`FPRINTF`

-----

`FPRINTF` prints a formatted string to a file.

```
(FPRINTF FILE FORMAT [EXPR ...])
```

Formats a string using the given format string and arguments and prints it to the specified file. Formatting is done like in `SPRINTF` (see `SPRINTF`).

`FPRINTF` returns the number of output characters or `NIL` on failure. If `FILE` is `NIL` then `FPRINTF` still returns the number of potentially written characters but no output is actually written. If `FORMAT` is `NIL` then `NIL` is returned.

See also `PRINTF`, `FOPEN`.

## 1.272 MUIbase/FERROR

FERROR

-----

FERROR checks if an file I/O error has occurred.

(FERROR FILE)

returns TRUE if an error for the given file has occurred, NIL otherwise. If file is NIL, NIL is returned.

See also FEOF, FOPEN, FCLOSE.

### 1.273 MUIbase/FEOF

FEOF

----

FEOF checks for an end of file status.

(FEOF FILE)

Tests the end-of-file indicator for the given file and returns TRUE if it is set. Otherwise NIL is returned. If file is NIL, NIL is returned.

See also FERROR, FTELL, FOPEN, FCLOSE.

### 1.274 MUIbase/FSEEK

FSEEK

-----

FSEEK sets the read/write position in a file.

(FSEEK FILE OFFSET WHENCE)

Sets the read/write position for the given file. The new position, measured in bytes, is obtained by adding OFFSET bytes to the position specified by WHENCE. If WHENCE is set to SEEK\_SET, SEEK\_CUR, or SEEK\_END, the OFFSET is relative to the start of the file, the current position, or end-of-file, respectively.

On success, FSEEK returns 0. Otherwise NIL is returned and the file position stays unchanged. If FILE, OFFSET, or WHENCE is NIL, or if WHENCE is not one of the constant values SEEK\_SET, SEEK\_CUR, or SEEK\_END then NIL is returned.

See also FTELL, Pre-defined constants.

---

## 1.275 MUIbase/FTELL

FTELL  
-----

FTELL returns the read/write position of a file.

(FTELL FILE)

Determines the current read/write position of the given file relative to the beginning of the file and returns it as an integer. If an error occurs or if file is NIL, then NIL is returned.

See also FSEEK, FEOF.

## 1.276 MUIbase/FGETCHAR

FGETCHAR  
-----

FGETCHAR reads a character from a file.

(FGETCHAR FILE)

Returns the next character from the given file as a string or NIL if FILE is NIL, end of file has been reached, or an error has happened. If the next character is a null character, an empty string is returned.

See also FGETCHARS, FGETSTR, FPUTCHAR.

## 1.277 MUIbase/FGETCHARS

FGETCHARS  
-----

FGETCHARS reads characters from a file.

(FGETCHARS NUM FILE)

returns a string containing the next NUM characters from the given file. If end of file has been reached before reading NUM characters or if a null character has been read then only these characters are returned. If NUM or FILE are NIL, NUM is negative, end of file has been reached before reading the first character, or a read error has happened then NIL is returned.

See also FGETCHAR, FGETSTR.

---

## 1.278 MUIbase/FGETSTR

FGETSTR

-----

FGETSTR reads a string from a file.

(FGETSTR FILE)

returns the next line of the given file as a string or NIL if FILE is NIL, end of file has been reached, or an error happened. The end of a line is detected if either a newline character or a null character is read, or if end of file is detected. In either case the string does not contain any newline characters.

See also FGETCHAR, FGETCHARS, FGETMEMO, FPUTSTR.

## 1.279 MUIbase/FGETMEMO

FGETMEMO

-----

FGETMEMO reads a memo from a file.

(FGETMEMO FILE)

returns a memo that contains the contents of the given file up to the next null character or up to the end of file. If FILE is NIL, end of file has been reached before reading any characters, or an error occurred then NIL is returned.

See also FGETSTR, FPUTMEMO.

## 1.280 MUIbase/FPUTCHAR

FPUTCHAR

-----

FPUTCHAR writes a character to a file.

(FPUTCHAR STR FILE)

Writes the first character of STR to the given file. If STR is empty, a null character is written, if STR or FILE are NIL, nothing happens. Returns STR or NIL in case an output error occurred.

See also FPUTSTR, FGETCHAR.

---

## 1.281 MUIbase/FPUTSTR

FPUTSTR  
-----

FPUTSTR writes a string to a file.

(FPUTSTR STR FILE)

Prints STR together with a newline character to the given file. If STR or FILE are NIL, nothing happens. Returns STR or NIL in case an output error occurred.

See also FPUTCHAR, FPUTMEMO, FGETSTR.

## 1.282 MUIbase/FPUTMEMO

FPUTMEMO  
-----

FPUTMEMO writes a memo to a file.

(FPUTMEMO MEMO FILE)

Prints MEMO to the given file. If MEMO or FILE are NIL, nothing happens. Returns MEMO or NIL in case an output error occurred.

See also FPUTSTR, FGETMEMO.

## 1.283 MUIbase/FFLUSH

FFLUSH  
-----

FFLUSH flushes pending data to a file.

(FFLUSH FILE)

Flushes all pending output for the given file. Returns 0 on success, NIL if an error occurred. If FILE is NIL then 0 is returned (no error).

See also FOPEN, FCLOSE.

## 1.284 MUIbase/Record functions

---

## Record functions

=====

This section lists functions that deal with records.

NEW	Allocating new record.
NEW*	Allocating new record by calling trigger function.
DELETE	Deleting a record.
DELETE*	Deleting a record by calling trigger function.
DELETEALL	Deleting all records of a table.
GETMATCHFILTER	Getting the match-filter state of a record.
SETMATCHFILTER	Setting the match-filter state of a record.
GETISSORTED	Checking if a record is in the right order.
SETISSORTED	Telling a record its order state.
RECNUM	Getting the record number of a record.
COPYREC	Copying the contents of a record.

**1.285 MUIbase/NEW**

NEW

---

NEW allocates a new record for a table.

(NEW TABLE INIT)

Allocates a new record in the given table. The parameter INIT specifies the record which should be used for initializing the new record. A value of NIL stands for the initial record.

NEW returns a record pointer to the new record.

The NEW function has the side effect of setting the program record pointer of the given table (see Tables) to the new record.

Example: (NEW table NIL) allocates a new record in the given table and initializes it with the initial record.

See also NEW\*, DELETE, Tables.

**1.286 MUIbase/NEW\***

NEW\*

-----

NEW\* is the star version of NEW (see NEW).

(NEW\* TABLE INIT)

NEW\* checks if you have specified a New trigger function for the given table (see New trigger). If so then this trigger function is called for allocating the record and its result is returned. The INIT parameter can be used to specify a record with which the new record should be initialized (use NIL for the initial record).

If no trigger function has been specified, the function behaves like the NEW function.

Warning: With this function it is possible to write endless loops, e.g. if you have defined a New trigger function for a table and this function calls NEW\* to allocate the record.

See also NEW, DELETE\*.

## 1.287 MUIbase/DELETE

DELETE

-----

DELETE deletes a record in a table.

(DELETE TABLE REQUESTER)

Deletes the current program record of the given table after an optional delete requester. The first argument specifies the table for which the current program record should be deleted, the second argument is a boolean expression. If it is NIL then the record is deleted silently, if it is not NIL then the state of preferences menu item Record delete requester is checked. If it is not set, the record is deleted silently, otherwise the standard delete requester appears asking for confirmation. If the users cancels the delete operation then the record will not be deleted.

The return code of the DELETE function reflects the selected action. If it returns TRUE then the record has been deleted, otherwise (the user has canceled the operation) NIL is returned.

On deletion, DELETE sets the program record pointer (see Tables) of the specified table to NIL.

Example: (DELETE table NIL) deletes the current record in the given table silently.

See also DELETE\*, DELETEALL, NEW, Tables.

## 1.288 MUIbase/DELETE\*

DELETE\*

-----

DELETE\* is the star version of DELETE (see DELETE).

(DELETE\* TABLE REQUESTER)

DELETE\* checks if you have specified a Delete trigger function for the given table (see Delete trigger). If so then this trigger function is called for deleting the record and its result is returned. The REQUESTER parameter can be used to specify if the trigger function should pop up a confirmation requester before deleting the record.

If no trigger function has been specified, the function behaves like the DELETE function.

Warning: With this function it is possible to write endless loops, e.g. if you have defined a Delete trigger function for a table and this function calls DELETE\* to delete the record.

See also DELETE, DELETEALL, NEW\*.

## 1.289 MUIbase/DELETEALL

DELETEALL

-----

DELETEALL deletes all records of a table.

(DELETEALL TABLE[\*])

Deletes all records of the specified table. If you add a star behind the table name then only those records that match the current filter of the table are deleted. There is no safety requester before deleting the records.

DELETEALL returns TRUE on successful deletion of all records, otherwise NIL is returned. If TABLE is NIL then NIL is returned.

Example: (DELETEALL table\*) deletes all records in the given table that match the filter of the table.

See also DELETE, Tables.

## 1.290 MUIbase/GETMATCHFILTER

GETMATCHFILTER

-----

---

GETMATCHFILTER returns the match-filter state of a record.

(GETMATCHFILTER REC)

Returns TRUE if the specified record matches the filter of its table, NIL otherwise. If the filter of the table is currently not active then TRUE is returned. If REC is NIL (the initial record) then NIL is returned.

See also SETMATCHFILTER, GETISSORTED, GETFILTERSTR, SETFILTERSTR.

## 1.291 MUIbase/SETMATCHFILTER

SETMATCHFILTER  
-----

SETMATCHFILTER sets the match-filter state of a record.

(SETMATCHFILTER REC ON)

Changes the match-filter state of the specified record to the value of ON. SETMATCHFILTER returns the new match-filter state of the given record. The new state may be different from the expected one because setting the match-filter state to NIL does only work when the filter of the corresponding table is currently active, otherwise TRUE is returned. Calling SETMATCHFILTER with a value of NIL for REC (the initial record) will always return NIL.

See also GETMATCHFILTER, SETISSORTED, GETFILTERSTR, SETFILTERSTR.

## 1.292 MUIbase/GETISSORTED

GETISSORTED  
-----

GETISSORTED returns the sorted state of a record.

(GETISSORTED REC)

Returns TRUE if the specified record is sorted in the order that has been defined for its table, NIL otherwise. If REC is NIL then NIL is returned.

See also SETISSORTED, GETMATCHFILTER, REORDER, GETORDERSTR, SETORDERSTR, Comparison function.

## 1.293 MUIbase/SETISSORTED

## SETISSORTED

-----

SETISSORTED sets the sorted state of a record.

(SETISSORTED REC ON)

Changes the sorted state of the specified record to the value of ON. Use this function if you think that a certain record is in the right order (ON = TRUE), or it it should be reordered (ON = NIL). Reordering all unsorted records can be done by calling the REORDER function (see REORDER).

SETISSORTED returns the new sorted state of the given record. Calling SETISSORTED with a value of NIL for REC (the initial record) will return NIL.

For an example on how to use this function, see Comparison function.

See also GETISSORTED, SETMATCHFILTER, REORDER, GETORDERSTR, SETORDERSTR, Comparison function.

## 1.294 MUIbase/RECNUM

## RECNUM

-----

RECNUM returns the record number of a record.

(RECNUM RECORD)

Returns the record number of the given record. Please note that the numbering for records is different than e.g. for lists. For lists, strings and others the counting begins with zero, however for records it begins with 1 for the first record. The number 0 is reserved for the initial record. This seems to be inconsistent with the rest of the MUIbase programming functions, but it does really makes sense here as the record numbers are also used in the window display.

See also RECORDS, INT.

## 1.295 MUIbase/COPYREC

## COPYREC

-----

COPYREC copies records.

(COPYREC REC SOURCE)

---

Copies the contents of record SOURCE to record REC. If SOURCE is NIL then REC is set to the values of the initial record. If REC is NIL then an error message is generated.

COPYREC returns REC.

See also NEW.

## 1.296 MUIbase/Attribute functions

Attribute functions

=====

This section lists functions that work on attributes of a table.

ATTRNAME	Getting the name of an attribute.
MAXLEN	Maximum number of characters for a string attribute.
GETLABELS	Getting the labels of a choice or string attribute.
SETLABELS	Setting the list-view labels of a string attribute.

## 1.297 MUIbase/ATTRNAME

ATTRNAME

-----

ATTRNAME returns the name of an attribute.

(ATTRNAME ATTR)

Returns a string containing the name of the specified attribute.

See also TABLENAME

## 1.298 MUIbase/MAXLEN

MAXLEN

-----

MAXLEN returns the maximum size of a string attribute.

(MAXLEN STRING-ATTR)

Returns the maximum number of characters that the given string attribute can hold.

---

See also LEN.

## 1.299 MUIbase/GETLABELS

GETLABELS

-----

GETLABELS returns all labels of a choice or string attribute.

(GETLABELS ATTR)

Returns the labels of the given choice or string attribute. In case of a choice attribute, the labels you entered in the attribute requester (see Type specific settings) are returned, in case of a string attribute, the labels you entered for the list-view pop-up (see Attribute object editor) are returned.

The labels are returned in one single string and are separated by newline characters.

For example, consider you have a choice attribute with the labels Car, House, and Oil. Then calling GETLABELS on that attribute will result to the string "Car\nHouse\nOil".

Note: you can easily convert the result string to a list by calling MEMOTOLIST (see MEMOTOLIST) on the result string.

See also SETLABELS.

## 1.300 MUIbase/SETLABELS

SETLABELS

-----

SETLABELS is used to set the labels of a string attribute.

(SETLABELS ATTR STR)

Sets the labels of the string attribute ATTR to the labels listed in the STR argument. The STR argument consists of lines each of which holds one label. The labels replace the ones you have entered for the pop-up list-view in the attribute object editor (see Attribute object editor).

SETLABELS returns the value of its STR argument.

Example: (SETLABELS Table.String "My house\nis\nyour house") sets the list-view labels of the specifies string attribute to My house, is, and your house.

Note: you can easily convert a list of labels to the required string format by calling LISTTOMEMO on the list.

See also GETLABELS.

### 1.301 MUIbase/Table functions

Table functions

=====

TABLENAME	Getting the name of a table.
GETORDERSTR	Getting record order.
SETORDERSTR	Setting record order.
REORDER	Reordering unsorted records.
REORDERALL	Reorder all records of a table.
GETFILTERACTIVE	Getting record filter state.
SETFILTERACTIVE	Setting record filter state.
GETFILTERSTR	Getting record filter expression.
SETFILTERSTR	Setting record filter expression.
RECORDS	Number of records.
RECORD	Getting pointer to a record.
SELECT	Select-from-where queries.

### 1.302 MUIbase/TABLENAME

TABLENAME

-----

TABLENAME returns the name of a table.

(TABLENAME TABLE)

Returns a string containing the name of the specified table.

See also ATTRNAME

### 1.303 MUIbase/GETORDERSTR

GETORDERSTR

-----

GETORDERSTR returns the record order of a table.

(GETORDERSTR TABLE)

Returns the current order expression of the given table. If the

table uses an attribute list for ordering then the returned string contains the attribute names separated by spaces. Each attribute name is prepended by a + or a - sign indicating ascending or descending order.

If the table is ordered by a comparison function then the name of this function is returned.

An empty string means no ordering.

Example

-----

Consider a table Person which is ordered by the Attributes Name (ascending), Town (ascending), and Birthday (descending). Then (ORDERSTR Person) will result to the string "+Name +Town -Birthday".

See also SETORDERSTR, REORDER, REORDERALL, GETISSORTED, SETISSORTED, Order, Comparison function.

## 1.304 MUIbase/SETORDERSTR

SETORDERSTR

-----

SETORDERSTR sets the record order of a table.

(SETORDERSTR TABLE ORDER)

Sets the order of the given table according to the ORDER string. The ORDER string can either hold a list of attribute names or the name of a comparison function.

For sorting using an attribute list, the ORDER string must contain the attribute names separated by any number of spaces, tabs or newlines. Each attribute name may be prepended by a + or a - sign indicating ascending or descending order. If you omit this sign then ascending ordering is assumed.

For sorting using a comparison function, the ORDER string must hold the name of the function.

SETORDERSTR returns TRUE if it has been able to set the new order, NIL otherwise, e.g. if an unknown attribute has been specified or the type of the attribute is not allowed for ordering. If you specify NIL for the ORDER argument then nothing happens and NIL is returned.

Note: For building the order string you should not directly write the attribute names into the string because when you change an attribute name then the order string will not be updated. Better use the ATTRNAME function (see ATTRNAME) to copy the attribute's name into the order string.

Example

-----

Consider a table Person with the attributes Name, Town, and Birthday. Then (SETORDERSTR Person (SPRINTF "+%s" (ATTRNAME Person.Name))) will set the order of table Person using Name as (ascending) order attribute.

See also GETORDERSTR, REORDER, REORDERALL, GETISSORTED, SETISSORTED, Order, Comparison function.

### 1.305 MUIbase/REORDER

REORDER

-----

REORDER brings all unsorted records back into the right order.

(REORDER TABLE)

Examines all records of the given table for unsorted ones and reinserts them to their right positions. After reinserting an unsorted record, the sorted state of the record is set to TRUE, thus on the return of REORDER the sorted state of all records is TRUE.

REORDER returns NIL.

Usually you only need to call this function, when you are using a comparison function for defining the order of a table. Orders defined by an attribute list are automatic, that is, a record is reordered automatically when needed.

For an example on how to use this function, see Comparison function.

See also REORDERALL, GETORDERSTR, SETORDERSTR, GETISSORTED, SETISSORTED, Order, Comparison function.

### 1.306 MUIbase/REORDERALL

REORDERALL

-----

REORDERALL reorders all records of a table.

(REORDERALL TABLE)

Reorders all records of the given table by setting the sorted state of all records to NIL and then calling REORDER for reordering everything.

REORDERALL returns NIL.

---

See also REORDER, GETORDERSTR, SETORDERSTR, GETISSORTED, SETISSORTED, Order, Comparison function.

### 1.307 MUIbase/GETFILTERACTIVE

GETFILTERACTIVE  
-----

GETFILTERACTIVE returns the filter state of a table.

(GETFILTERACTIVE TABLE)

Returns TRUE if the filter of the specified table is currently activated and NIL otherwise.

See also SETFILTERACTIVE, GETFILTERSTR, GETMATCHFILTER.

### 1.308 MUIbase/SETFILTERACTIVE

SETFILTERACTIVE  
-----

SETFILTERACTIVE sets the filter state of a table.

(SETFILTERACTIVE TABLE BOOL)

Sets the filter state of the specified table. If BOOL is non-NIL then the filter is activated, otherwise it is deactivated.

SETFILTERACTIVE returns the new state of the filter. The new state may not be the expected one in case you activate the filter but an error condition occurs and the filter can't be activated. However deactivating the filter always succeeds.

See also GETFILTERACTIVE, SETFILTERSTR, SETMATCHFILTER.

### 1.309 MUIbase/GETFILTERSTR

GETFILTERSTR  
-----

GETFILTERSTR returns the record filter expression of a table.

(GETFILTERSTR TABLE)

Returns the record filter expression of the specified table as a string. An empty string means that no filter expression has been set

---

for this table.

See also SETFILTERSTR, GETFILTERACTIVE, GETMATCHFILTER.

## 1.310 MUIbase/SETFILTERSTR

SETFILTERSTR

-----

SETFILTERSTR sets the record filter expression of a table.

```
(SETFILTERSTR TABLE FILTER-STR)
```

Sets the record filter expression of the specified table to the expression in the FILTER-STR argument. If the filter of the given table is currently active then the new filter expression is directly applied to all records and the match-filter state of all records are recomputed.

SETFILTERSTR returns TRUE if it has been able to compile the given filter string expression, otherwise NIL is returned. Note that you only get the result of the compilation. If the filter of the given table is currently active and recomputing all match-filter states of the corresponding records fails then you will not notice that from the result of this function. The recommended way to set a new filter expression is like follows:

```
(SETFILTERACTIVE Table NIL) ; always succeeds.
(IF (NOT (SETFILTERSTR Table filter-string))
  (ERROR "Can't set filter string for %s!" (TABLENAME Table))
)
(IF (NOT (SETFILTERACTIVE Table TRUE))
  (ERROR "Can't activate filter for %s!" (TABLENAME Table))
)
```

If SETFILTERSTR is called with a value of NIL for the FILTER-STR argument then nothing happens and NIL is returned.

Example: (SETFILTERSTR Table "> Value 0.0").

See also GETFILTERSTR, SETFILTERACTIVE, SETMATCHFILTER.

## 1.311 MUIbase/RECORDS

RECORDS

-----

RECORDS returns the number of records in a table.

```
(RECORDS TABLE)
```

Returns the number of records in the given table. You may append a star to the table name for counting the number of records that match the filter of the table.

See also RECORD, RECNUM.

### 1.312 MUIbase/RECORD

RECORD

-----

RECORD returns a record pointer for a given record number.

(RECORD TABLE NUM)

Returns the record pointer to the NUM-th record in the given table or NIL if a record with this number doesn't exist. You may add a star to the table name to get the NUM-th record that matches the record filter of the table.

Please note that record numbers start with 1 and record number 0 is used for the initial record.

See also RECORDS, RECNUM.

### 1.313 MUIbase/SELECT

SELECT

-----

SELECT extracts and returns various data from records.

(SELECT [DISTINCT] EXPRLIST FROM TABLELIST  
[WHERE WHERE-EXPR] [ORDER BY ORDERLIST])

where EXPRLIST is either a simple star \* or a list of expressions with optional titles separated by commas:

EXPRLIST:        \* | EXPR "title", ...

and TABLELIST is a list of table names:

TABLELIST:        TABLE[\*] [ident], ...

For each table in the table list you can specify an identifier. This can be very useful if a table occurs more than once in the table list (see example of comparing ages below). If you add a star to a table then only the records matching the currently defined filter of that table will be examined.

The orderlist has the following syntax:

```
ORDERLIST:      EXPR [ASC | DESC], ...
```

where EXPR, ... can be arbitrary expressions or field numbers. For example (SELECT Name FROM ... ORDER BY 1) will sort the result by the Name attribute. You may specify ASC or DESC for an ascending or descending order. If none of them are present then an ascending order is assumed.

How it works

-----

The select-from-where query builds the (mathematical) cross product of all tables in the table list (it examines all sets of records in TABLE, ...) and checks the where-expression (if any). If the where-expression results to TRUE (or there is no where-expression) then a list is build whose elements are calculated by the expression list in the select-part. If you have specified a single star for the expression list then the list contains the values of all attributes belonging to tables in the table list (except virtual attributes and buttons).

The result of the query is a list of lists. The first list entry contains the title strings, the remaining ones contain the values of the from-list in the matching records.

Examples

-----

See Query examples for some examples using the SELECT function.

See also FOR ALL.

## 1.314 MUIbase/Gui functions

Gui functions

=====

This section describes functions for manipulating gui elements.

SETCURSOR	Placing the cursor on a gui element.
GETDISABLED	Getting the disabled state of a gui element.
SETDISABLED	Setting the disabled state of a gui element.
GETWINDOWDISABLED	Getting the disabled state of a window.
SETWINDOWDISABLED	Setting the disabled state of a window.
GETWINDOWOPEN	Getting open/close state of a window.
SETWINDOWOPEN	Opening/closing of a window.

### 1.315 MUIbase/SETCURSOR

SETCURSOR

-----

SETCURSOR sets the cursor on a gui element.

(SETCURSOR ATTR-OR-TABLE)

Sets the cursor on the given attribute or table gui object. The function also opens the window where the attribute/table resides in if the window was not open.

SETCURSOR returns TRUE if everything went ok (window could be opened) or NIL on failure.

### 1.316 MUIbase/GETDISABLED

GETDISABLED

-----

GETDISABLED returns the disabled state of an attribute.

(GETDISABLED ATTR)

Returns the disabled state of the specified attribute in the current record.

See also SETDISABLED, GETWINDOWDISABLED.

### 1.317 MUIbase/SETDISABLED

SETDISABLED

-----

SETDISABLED sets the disabled state of an attribute.

(SETDISABLED ATTR BOOL)

Sets the disabled state of the specified attribute in the current record to the value of BOOL. Returns the new value of the disabled state.

See also GETDISABLED, SETWINDOWDISABLED.

### 1.318 MUIbase/GETWINDOWDISABLED

GETWINDOWDISABLED  
-----

GETWINDOWDISABLED returns the disabled state of a window.

(GETWINDOWDISABLED ATTR-OR-TABLE)

Returns the state of the disabled flag of the window in which the specified attribute or table resides. If the attribute or table resides in the root window then NIL is returned.

See also SETWINDOWDISABLED, GETWINDOWOPEN, GETDISABLED.

### 1.319 MUIbase/SETWINDOWDISABLED

SETWINDOWDISABLED  
-----

SETWINDOWDISABLED sets the disabled state of a window.

(SETWINDOWDISABLED ATTR-OR-TABLE DISABLED)

Sets the state of the disabled flag of the window button in which the specified attribute or table resides to the value of DISABLED. If you disable a window, the window is closed and the corresponding window button is disabled. You can't disable the root window of a project.

SETWINDOWDISABLED returns the new state of the window's disabled flag.

See also GETWINDOWDISABLED, SETWINDOWOPEN, SETDISABLED.

### 1.320 MUIbase/GETWINDOWOPEN

GETWINDOWOPEN  
-----

GETWINDOWOPEN returns the open state of a window.

(GETWINDOWOPEN ATTR-OR-TABLE)

Returns the open state of the window where the given attribute/table resides.

See also SETWINDOWOPEN, GETWINDOWDISABLED.

---

### 1.321 MUIbase/SETWINDOWOPEN

SETWINDOWOPEN  
-----

SETWINDOWOPEN opens and closes a window.

(SETWINDOWOPEN ATTR-OR-TABLE OPEN)

Opens or closes the window in which the given attribute/table resides. If OPEN is non-NIL then the window is opened, otherwise it is closed. You cannot close the root window of a project.

SETWINDOWOPEN returns the new open state of the window.

See also GETWINDOWOPEN, SETWINDOWDISABLED.

### 1.322 MUIbase/Project functions

Project functions  
=====

This section lists functions dealing with projects.

PROJECTNAME    Getting the project name.  
CHANGES       Getting number of changes made to current project.

### 1.323 MUIbase/PROJECTNAME

PROJECTNAME  
-----

PROJECTNAME returns the project name.

(PROJECTNAME)

PROJECTNAME returns the name of the current project as a string or NIL if no name has been defined yet.

See also CHANGES.

### 1.324 MUIbase/CHANGES

CHANGES  
-----

---

CHANGES returns the number of changes in the current project.

(CHANGES)

Returns an integer containing the number of changes since the last save operation of the current project.

See also PROJECTNAME.

## 1.325 MUIbase/System functions

System functions

=====

This section lists functions accessing the operating system.

EDIT	Launching external editor asynchronously.
EDIT*	Launching external editor synchronously.
VIEW	Launching external viewer asynchronously.
VIEW*	Launching external viewer synchronously.
SYSTEM	Calling external commands.
STAT	Examining a file.
TACKON	Creating pathname from dirname and filename.
DIRNAME	Getting directory name from a path.
FILENAME	Getting last component of a path.
TODAY	Getting current date.
NOW	Getting current time.
MESSAGE	Showing messages to the user.
GC	Forcing garbage collection.

## 1.326 MUIbase/EDIT

EDIT

----

EDIT launches the external editor.

(EDIT FILENAME)

Starts the external editor for editing the specified file. The external editor can be set in menu item Preferences - External editor (see External editor). EDIT starts the external editor asynchronously, that is, the function returns to the caller immediately.

EDIT returns TRUE if it was successful in starting the editor, else it returns NIL.

See also EDIT\*, VIEW, SYSTEM.

### 1.327 MUIbase/EDIT\*

EDIT\*  
-----

EDIT\* is the star version of EDIT and has the same effect as EDIT (see EDIT). The only difference is that EDIT\* starts the external editor synchronously and waits until the user exits the editor.

See also EDIT, VIEW\*, SYSTEM.

### 1.328 MUIbase/VIEW

VIEW  
-----

VIEW launches the external viewer.

(VIEW FILENAME)

Starts the external viewer for displaying the specified file. The external viewer can be set in menu item Preferences - External viewer (see External viewer). VIEW starts the external viewer asynchronously, that is, the function returns to the caller immediately.

VIEW returns TRUE if it was successful in starting the viewer, else it returns NIL.

See also VIEW\*, EDIT, SYSTEM.

### 1.329 MUIbase/VIEW\*

VIEW\*  
-----

VIEW\* is the star version of VIEW and has the same effect as VIEW (see VIEW). The only difference is that VIEW\* starts the external viewer synchronously and waits until the user exits the viewer.

See also VIEW, EDIT\*, SYSTEM.

---

### 1.330 MUIbase/SYSTEM

SYSTEM

-----

SYSTEM calls an external program.

(SYSTEM FMT [ARG ...])

Calls an external program. The command line to call the program is generated from FMT and the optional arguments like in the SPRINTF function (see SPRINTF). SYSTEM waits until the called program exits. If you don't want SYSTEM to wait then use a command line that starts the program in the background.

SYSTEM returns TRUE on success and NIL on failure, e.g. when the command line could not be executed or the called command returned an error code.

Example: (SYSTEM "run %s %s" "clock" "digital") launches the system's clock in digital mode.

See also EDIT, EDIT\*, VIEW, VIEW\*.

### 1.331 MUIbase/STAT

STAT

----

STAT examines a filename.

(STAT FILENAME)

Examines if the specified filename exists in the file system. STAT returns NIL if the filename could not be found, 0 if the filename exists and is a directory, and an integer value greater than 0 if the filename exists and is a regular file.

### 1.332 MUIbase/TACKON

TACKON

-----

TACKON creates a pathname.

(TACKON DIRNAME FILENAME)

Joines DIRNAME and FILENAME to a pathname. TACKON knows how to deal with colons and slashes in DIRNAME. It returns the pathname as a string or NIL if DIRNAME or FILENAME is NIL.

---

Example: (TACKON "Sys:System" "CLI") results to "Sys:System/CLI".

See also FILENAME, DIRNAME.

### 1.333 MUIbase/FILENAME

FILENAME

-----

FILENAME extracts the filename part of a path name.

(FILENAME PATH)

Extracts the last component of the given path name. There is no check whether PATH actually refers to a file, thus it is also possible to use FILENAME to get the name of a sub-directory. FILENAME returns its result as a string or NIL if PATH is NIL.

Example: (FILENAME "Sys:System/CLI") results to "CLI".

See also DIRNAME, TACKON.

### 1.334 MUIbase/DIRNAME

DIRNAME

-----

DIRNAME extracts the directory part of a path name.

(DIRNAME PATH)

Extracts the directory part of the given path name. There is no check whether PATH actually refers to a file, thus it is also possible to use DIRNAME to get the name of a parent directory. DIRNAME returns its result as a string or NIL if PATH is NIL.

Example: (DIRNAME "Sys:System/CLI") results to "Sys:System".

See also FILENAME, TACKON.

### 1.335 MUIbase/TODAY

TODAY

-----

TODAY returns the current date.

---

(TODAY)

Returns the current date as a date value.

See also NOW.

### 1.336 MUIbase/NOW

NOW

---

NOW returns the current time.

(NOW)

Returns the current time as a time value.

See also TODAY.

### 1.337 MUIbase/MESSAGE

MESSAGE

-----

MESSAGE displays a message to the user.

(MESSAGE FMT [ARG ...])

Sets the window title of the pause/abort window (if it is open). The title string is generated from FMT and the optional arguments like in the SPRINTF function (see SPRINTF).

MESSAGE returns the formatted title string.

Example: (MESSAGE "6 \* 7 = %i" (\* 6 7)).

See also PRINT, PRINTF.

### 1.338 MUIbase/GC

GC

--

GC forces garbage collection.

(GC)

---

Forces garbage collection and returns NIL. Normally garbage collection is done automatically from time to time.

### 1.339 MUIbase/Pre-defined variables

Pre-defined variables

=====

MUIbase knows some pre-defined global variables.

In the current version there exists only one global variable: stdout (see stdout).

### 1.340 MUIbase/Pre-defined constants

Pre-defined constants

=====

The following pre-defined constants can be used in any expression for programming.

Name	Type	Value	Comment
NIL	any	NIL	
TRUE	bool	TRUE	
RESET	string	"\33c"	
NORMAL	string	"\33[0m"	
ITON	string	"\33[3m"	
ITOFF	string	"\33[23m"	
ULON	string	"\33[4m"	
ULOFF	string	"\33[24m"	
BFON	string	"\33[1m"	
BFOFF	string	"\33[22m"	
ELITEON	string	"\33[2w"	
ELITEOFF	string	"\33[1w"	
CONDON	string	"\33[4w"	
CONDOFF	string	"\33[3w"	
WIDEON	string	"\33[6w"	
WIDEOFF	string	"\33[5w"	
NLQON	string	"\33[2\"z"	
NLQOFF	string	"\33[1\"z"	
INT_MAX	integer	2147483647	Maximum integer value
INT_MIN	integer	-2147483648	Minimum integer value
HUGE_VAL	real	1.797693e+308	Largest absolute real value
PI	real	3.14159265359	
OSVER	integer	<OS version>	
OSREV	integer	<OS revision>	
MBVER	integer	<MUIbase version>	
MBREV	integer	<MUIbase revision>	

LANGUAGE	string	depends	Default language.
SEEK_SET	integer	see stdio.h	Seek from beg. of file.
SEEK_CUR	integer	see stdio.h	Seek from current pos.
SEEK_END	integer	see stdio.h	Seek from end of file.

See Constants, for more information about constants. For defining your own constants, use the #define preprocessing directive (see #define).

## 1.341 MUIbase/Functional parameters

Functional parameters  
=====

You can pass a function as an argument to another function. This is useful for defining functions of higher order, e.g. for sorting or mapping a list.

To call a function that has been passed in an argument you have to use the FUNCALL function (see FUNCALL).

Example:

-----

```
(DEFUN map (l fun)                # arguments: list and function
  (LET (res)                      # local variable res, initialized with NIL
    (DOLIST (i l)                 # for all items one by one
      (SETQ res
        (CONS (FUNCALL fun i) res) # apply function and
      )                               # build new list
    )
    (REVERSE res)                 # we need to reverse the new list
  )
)
```

You can now use the map function for example to increment all items of a list of integers:

```
(map (LIST 1 2 3 4) 1+) results to ( 2 3 4 5 ).
```

See also FUNCALL, MAPFIRST.

## 1.342 MUIbase/Type specifiers

Type specifiers  
=====

It is possible to specify the type of a local variable by adding a type specifier behind the name. The following type specifiers exist:

Specifier	Description
:INT	for integers
:REAL	for reals
:STR	for strings
:MEMO	for memos
:DATE	for dates
:TIME	for times
:LIST	for lists
:FILE	for file handles
:FUNC	for functions of any type
:TABLE	for record pointers to TABLE

The type specifier appends the variable name as in the following example:

```
(LET (x:INT (y:REAL 0.0) z) ...)
```

The example defines three new variables *x*, *y* and *z*, where *x* is of type integer and initialized with *NIL*, *y* is of type real and initialized with 0.0, and *z* is an untyped variable initialized with *NIL*.

The advantage of the type specifiers is that the compiler can detect more type errors, e.g. if you have a function:

```
(DEFUN foo (x:INT) ...)
```

and call it with (foo "bar") then the compiler generates an error message. However, if you call foo with an untyped value, e.g. (foo (FIRST list)) then no error checking can be done at compile time because the type of (FIRST list) is unknown.

For reasons of speed no type checking is currently done at run time. It could be implemented but then would add a little overhead which is not really necessary as the wrong type will result in a type error sooner or later anyway.

Type specifiers for record pointers have another useful feature. If you tag a variable as a record pointer to a table then you can access all attributes of this table by using the variable name instead of the table name in the attribute path. E.g. if you have a table *Foo* with an attribute *Bar*, and you define a variable *foo* as:

```
(LET (foo:Foo)
```

then you can print the *Bar* attribute of the third record by using:

```
(SETQ foo (RECORD Foo 3)) (PRINT foo.Bar)
```

Note that in a select-from-where expression, variables defined in the from list automatically have a type of record pointer to the corresponding table.

### 1.343 MUIbase/Semantics of expressions

Semantics of expressions  
=====

The semantics of expressions are very important for understanding what a program does. This section lists the semantics depending on syntactical expressions.

(FUNC [EXPR ...])

Evaluates EXPR ... and then calls the function FUNC (call by value). Returns the return value of the called function. In MUIbase there exists some non-strict functions, e.g. AND, OR and IF. These functions may not evaluate all expressions. For more information about non-strict functions, see Lisp syntax, AND, OR, and IF.

([EXPR ...])

Evaluates EXPR ... and returns the value of the last expression (see PROGN). An empty expression () evaluates to NIL.

TABLE

Returns the program record pointer of the given table.

TABLE\*

Returns the gui record pointer of the given table.

ATTRPATH

Returns the contents of the specified attribute. The attribute path specifies which record is used to extract the attribute value. For example Table.Attribute uses the program record of Table to extract the value of the attribute, Table.ReferenceAttribute.Attribute uses the program record of Table to extract the value of the reference attribute (which is a record pointer) and then uses this record to extract the value of Attribute.

LOCALVAR

Returns the contents of the local variable. Local variables can be defined e.g. with LET (see LET).

LOCALVAR.ATTRPATH

Uses the record pointer of LOCALVAR to determine the value of the specified attribute.

### 1.344 MUIbase/Function triggering

Function triggering  
=====

For automatic execution of MUIbase programs you can specify trigger functions for projects, tables and attributes which are called in specific cases. This section lists all available trigger possibilities.

---

onOpen	Trigger after opening a project.
onClose	Trigger when closing a project.
onChange	Trigger when changing a project.
New trigger	Trigger for allocating a new record.
Delete trigger	Trigger for deleting a record.
Comparison function	For comparing records of a table.
Attribute trigger	Trigger for changing an attribute.
Virtual attributes	How to write functions for virtual attributes.

### 1.345 MUIbase/onOpen

onOpen

-----

After opening a project, MUIbase searches the project's program for a function called onOpen. If such a function exists then this function is called without any arguments.

Example

-----

```
(DEFUN onOpen ()
  (ASKBUTTON NIL "Thank you for opening me!" NIL NIL)
)
```

See also onClose, onChange, demo Trigger.mb.

### 1.346 MUIbase/onClose

onClose

-----

Before closing a project, MUIbase searches the project's program for a function called onClose. If such a function exists then this function is called without any arguments. In the current version the result of the trigger function is ignored and the project is closed regardless of its return value.

If you do changes to the project in the onClose function then MUIbase will ask you for saving the project first before it is actually closed. If you use menu item Project - Save & Close for closing the project, the trigger function is called before saving the project, thus the changes are saved automatically.

Example

-----

```
(DEFUN onClose ()
```

```
(ASKBUTTON NIL "Good bye!" NIL NIL)
)
```

See also `onOpen`, `onChange`, `demo Trigger.mb`.

### 1.347 MUIbase/onChange

`onChange`

-----

Whenever the user makes changes to a project or after saving a project, MUIbase searches the project's program for a function called `onChange`. If such a function exists then this function is called without any arguments. This can be used to count the changes a user does to a project.

Example

-----

```
(DEFUN onChange ()
  (SETQ Control.NumChanges (CHANGES))
)
```

In the above example `Control.NumChanges` could be a virtual attribute somewhere in an exactly-one-record table for displaying the project's number of changes.

See also `onOpen`, `onClose`, `demo Trigger.mb`.

### 1.348 MUIbase/New trigger

`New trigger`

-----

When the user wants to allocate a new record by selecting one of the menu items `New record` or `Duplicate record` and the `New trigger` of that table has been set to a MUIbase program function then this trigger function is executed. The `New trigger` function can be set in the table requester (see `Creating tables`).

The trigger function receives `NIL` or a record pointer as the first and only argument. `NIL` means that the user wants to allocate a new record, a record pointer means that the user wants to duplicate this record. If the trigger function has more than one argument then these are initialized with `NIL`. The trigger function should allocate the new record by calling the `NEW` function (see `NEW`). The result returned by the trigger function will be examined. If it returns a record pointer then this record will be displayed.

The `New trigger` is also called when a MUIbase program calls the `NEW*`

function (see NEW\*).

Sample New trigger function

-----

```
(DEFUN newRecord (init)
  (PROG1
    (NEW Table init)
    ...
  )
)
```

See also NEW\*, Delete trigger.

### 1.349 MUIbase/Delete trigger

Delete trigger

-----

When the user wants to delete a record by selecting menu item Delete record and the Delete trigger of that table has been set to a MUIbase program function then this trigger function is executed. The Delete trigger function can be set in the table requester (see Creating tables).

The trigger function receives a boolean argument as its only argument. If the argument is non-NIL then the function should ask the user if he really wants to delete the record. If so, the function should call DELETE (see DELETE) for deleting the record.

The Delete trigger is also called when a MUIbase program calls the DELETE\* function (see DELETE\*).

Sample Delete trigger function

-----

```
(DEFUN deleteRecord (requester)
  (DELETE Table requester)
)
```

See also DELETE\*, New trigger.

### 1.350 MUIbase/Comparison function

Comparison function

-----

For defining an order on the records of a table you can use a comparison function that takes two record pointers as arguments and returns an integer value reflecting the order relation of the two

records. The comparison function should return a value smaller than 0 if its first argument is smaller than the second, 0 if they are equal and a value larger than 0 if the first argument is greater than the second one.

For example, if you have a table Persons with a string attribute Name then you could use the following function for comparing two records:

```
(DEFUN cmpPersons (rec1:Persons rec2:Persons)
  (CMP rec1.Name rec2.Name)
)
```

This will order all records according to the Name field using case sensitive string comparison. Note that by using an attribute list you could not achieve the same ordering as string comparison is done case insensitive for attribute lists.

Using a comparison function you can define very complex order relations. Be careful not to create recursive functions that will call itself. MUIbase will stop program execution and prompt you with an error message if you try to do so. Also you should not use commands that cause side effects, e.g. setting a value to an attribute.

When using a comparison function, MUIbase does not always know when it has to reorder its records. E.g. consider in the above example another table Toys with a string attribute Name and a reference attribute Owner referring to Persons, and the following function for comparing records:

```
(DEFUN cmpToys (rec1:Toys rec2:Toys)
  (CMP* rec1.Owner rec2.Owner)
)
```

This function is using the order of Persons for determining the order of the records, thus the records of Toys are sorted according to the order of Persons.

Now if the user changes one record in the Persons table and this record gets a new position then also records in Toys that refer to this record have to be reordered. MUIbase, however, does not know of this dependency.

Besides using menu item Table - Reorder all records on the Toys table for rearranging the order, you can implement an automatic reordering yourself by specifying the following trigger function for the Name attribute of Persons:

```
(DEFUN setName (newValue)
  (SETQ Persons.Name newValue)
  (FOR ALL Toys WHERE (= Toys.Owner Persons) DO
    (SETISSORTED Toys NIL)
  )
  (REORDER Toys)
)
```

The function clears the sorted flag for all records that refer to

the current record of table Persons and then reorders all unsorted records of table Toys.

See also Order, GETISSORTED, SETISSORTED, REORDER, REORDERALL, GETORDERSTR, SETORDERSTR, demo Order.mb.

## 1.351 MUIbase/Attribute trigger

Attribute trigger  
-----

In the attribute requester (see Creating attributes) you can define a trigger function that is called whenever the user wants to change the attribute contents.

If you have defined such a trigger function for an attribute and the user changes the value of that attribute then the record contents are not automatically updated with the new value. Instead the value is passed to the trigger function as first argument. The trigger function can now check the value and may refuse it. To store the value in the record you have to use the SETQ function.

The trigger function should return the result of the SETQ call or the old value of the attribute if it decides to refuse the new one.

The trigger function is also called when a MUIbase program calls the SETQ\* function (see SETQ\*) for setting a value to the attribute.

Sample attribute trigger function  
-----

```
(DEFUN setAmount (amount)
  (IF SOME-EXPRESSION
    (SETQ Table.Amount amount)
    (ASKBUTTON NIL "Invalid value!" NIL NIL)
  )
  Table.Amount ; return current value
)
```

See also SETQ\*

## 1.352 MUIbase/Programming virtual attributes

Programming virtual attributes  
-----

In MUIbase virtual attributes are special attributes that calculate their value on the fly whenever it is needed. E.g. if you go to another record by clicking on one of the arrow buttons in a table's panel bar and a virtual attribute in that table has its Immediate flag

---

(see Attribute object editor) set then the value of that attribute is computed and displayed. For computing the value the attribute's Compute trigger function is called. This trigger function can be specified in the attribute requester (see Type specific settings). The return value of this function defines the value of the virtual attribute. If you don't specify a Compute trigger function for a virtual attribute then the attribute's value is NIL.

You can also trigger the calculation of a virtual attribute by simply accessing it in an MUIbase program, so e.g. if you have a button that should compute the value of the virtual attribute, you only need to specify a function for the button like the following one:

```
(DEFUN buttonHook ()  
  VIRTUAL-ATTR  
)
```

You can also set a virtual attribute to any value by using the SETQ function:

```
(SETQ VIRTUAL-ATTR EXPR)
```

However if you access the virtual attribute after the SETQ call then the value of the virtual attribute is recomputed.

There is no caching of the value of a virtual attribute because it's not easy to know when the value has to be recomputed and when not. So you should access virtual attributes rarely and cache the value in local variables for further use yourself.

For an example on how to use virtual attributes please check the Movie.mb demo.

See also Virtual, demo Movie.db.

### 1.353 MUIbase/ABConvert

ABConvert  
\*\*\*\*\*

To convert an AmigaBase project to a MUIbase project a small utility called ABConvert exists. It loads an AmigaBase project created with AmigaBase version 2.0 or higher and saves it as an MUIbase project. For projects created by older AmigaBase versions, please load and save them with AmigaBase version 2.4 and then do the conversion.

A Solaris binary of ABConvert can be found in the solaris directory. If you have ever problems with low memory when using the Amiga version, run ABConvert on a Solaris box and your memory problems should be gone.

To run the conversion utility type ABConvert AB-FILE MB-FILE where AB-FILE is an existing AmigaBase project and MB-FILE is a new MUIbase project to be created.

---

Since MUIbase is a completely new database application, only the structures and datasets of AmigaBase are converted. Programs, filters, orders, etc. must be redone using MUIbase. For converting AmigaBase programs to MUIbase ones it is best to print all programs of the AmigaBase project and then use the guidelines described in the text file PortingABPrograms for converting the programs.

Please note, that the terms used in AmigaBase have been renamed in MUIbase, e.g. an AmigaBase record is now called table, a variable is now called attribute and a dataset is now called record.

MUIbase is relational. Therefore the hierarchy of an AmigaBase project has to be converted into tables. This is done by adding reference attributes to each table (except for the "root" table) which point to the "parent" record.

## 1.354 MUIbase/Menus

Menus

\*\*\*\*\*

Menu Project

Info	Information about a project.
Clear - Project	Start a new project.
Clear - Records	Delete all records.
Open new	Opens new root window.
Open - Project	Loads project.
Open - Structure	Loads project without records.
Save	Saves project to disk.
Save & Reorg	Saves and reorganizes a project.
Save & Reorg as	Saves and reorganizes with new filename.
Delete	Deletes project from disk.
Close	When your are done with a project.
Save & Close	Saves then closes project.
Check data integrity	If your project ever gets corrupted.
Swap records	Flush all records to disk.
Structure editor	Open structure editor.
Print structure	Overview of all tables and attributes.
Quit	Exit MUIbase.

Menu Preferences

Record memory	Size of record buffer.
Record delete req?	Safety requester when deleting records.
Ext editor for prog?	Use your favorite editor for programming.
Icon creation?	Create project icons.
Set icon tool name	Set tool name for project icons.
Set formats	Set real and date format.

Set external editor	Specify your external editor.
Set external viewer	Specify your external viewer.
Confirm save & reorg	Safety requester when reorganizing a project.
Confirm quit?	Safety requester when quitting MUIbase.
Project dep settings	Global versus project local preferences.
MUI	MUI's preferences.
Load preferences	Load preferences from disk.
Save preferences	Save preferences to disk.

#### Menu Table

New record	Adding a new record.
Duplicate record	Copying a record.
Delete record	If you don't need a record any more.
Delete all records	If you want to start from scratch.
Goto Record	Browsing records.
Change filter	Specify a filter expression.
Change order	How to specify an order.
Reorder all records	If records ever become unsorted.
Search for	How to search for a record.
Search forward	Go to the next matching record.
Search backward	Go to the previous matching record.
Import records	How to import records.
Export records	How to export records.

#### Menu Program

Edit	Where to enter a MUIbase program.
Compile	Compiling a program.
Include directory	Where to look for external include files.
Debug information	Compile with or without debug information.
Output file	Where program output goes.
Queries	Open the query editor.

#### Menu Help

Contents	This user manual.
About	About MUIbase.
About MUI	About Magic User Interface.

## 1.355 MUIbase/Acknowledgments

### Acknowledgments

\*\*\*\*\*

Thanks to:

- \* Mats Granstrom for his long term beta testing of MUIbase, his ideas to improve it, and for writing the tutorial. Mats sometimes writes really funny emails. It's always a pleasure to read them.
- \* Ralph Reuchlein (Ralphie) for huge bug report lists, tons of emails with ideas and suggestions, and his unlimited time for discussing them. Without his ideas (like e.g. the #-directives for programming) MUIbase would not be what it is today.

Furthermore, Ralphie did a great job in translating the MUIbase user manual to German. Thanks!

Ralphie also created and maintains the MUIbase home page <http://www.amigaworld.com/support/muibase>.

- \* Thomas Fricke for beta-testing and his images to improve the appearance of MUIbase. He is the one who painted the MUIbase and project icons, the MUIbase empty display image, the window open/closed images, and others.
- \* André Schenk and Klaus Gessner for beta-testing and their knowledge about relational databases and SQL.
- \* Allan Odgaard for his TextEditor class and his great support to improve it for MUIbase.
- \* Oliver Roberts for beta testing and some very good bug reports, and for FlGP-Ed (see <http://www.nanunanu.org/~oliver/>).
- \* Petri Nordlund for permission to use his registration files from Executive for MUIbase. If you don't know Executive yet, you should definitely check it out. It's available on Aminet.
- \* Henning Thilemann for ideas and beta testing.

## 1.356 MUIbase/Author

Author

\*\*\*\*\*

MUIbase is developed by:

Steffen Gutmann  
Orleanstr. 47  
81667 Munich  
GERMANY

Email: <[gutmann@ieee.org](mailto:gutmann@ieee.org)>

---

## 1.357 MUIbase/Function index

Function index

\*\*\*\*\*

#define	#define
#elif	#elif
#else	#else
#endif	#endif
#if	#if
#ifdef	#ifdef
#ifndef	#ifndef
#include	#include
#undef	#undef
*	mul
+	add
-	sub
-	fdiv
1+	1+
1-	1-
<	Relational operators
<*	Relational operators
<=	Relational operators
<=*	Relational operators
<>	Relational operators
<>*	Relational operators
=	Relational operators
=*	Relational operators
>	Relational operators
>*	Relational operators
>=	Relational operators
>=*	Relational operators
ABS	ABS
AND	AND
APPEND	APPEND
ASC	ASC
ASKBUTTON	ASKBUTTON
ASKCHOICE	ASKCHOICE
ASKCHOICESTR	ASKCHOICESTR
ASKDIR	ASKDIR
ASKFILE	ASKFILE
ASKINT	ASKINT
ASKMULTI	ASKMULTI
ASKOPTIONS	ASKOPTIONS
ASKSTR	ASKSTR
ATTRNAME	ATTRNAME
CASE	CASE
CHANGES	CHANGES
CHR	CHR
CMP	CMP
CMP*	CMP*
CONCAT	CONCAT
CONCAT2	CONCAT2
COND	COND

CONS	CONS
CONSP	Type predicates
COPYREC	COPYREC
COPYSTR	COPYSTR
DATE	DATE
DATEP	Type predicates
DEFUN	DEFUN
DEFUN*	DEFUN*
DEFVAR	DEFVAR
DELETE	DELETE
DELETE*	DELETE*
DELETEALL	DELETEALL
DIRNAME	DIRNAME
DIV	DIV
DO	DO
DOLIST	DOLIST
DOTIMES	DOTIMES
EDIT	EDIT
EDIT*	EDIT*
ERROR	ERROR
EXIT	EXIT
FCLOSE	FCLOSE
FEOF	FEOF
FERROR	FERROR
FFLUSH	FFLUSH
FGETCHAR	FGETCHAR
FGETCHARS	FGETCHARS
FGETMEMO	FGETMEMO
FGETSTR	FGETSTR
FILENAME	FILENAME
FILLMEMO	FILLMEMO
FIRST	FIRST
FOPEN	FOPEN
FOR ALL	FOR ALL
FORMATMEMO	FORMATMEMO
FPRINTF	FPRINTF
FPUTCHAR	FPUTCHAR
FPUTMEMO	FPUTMEMO
FPUTSTR	FPUTSTR
FSEEK	FSEEK
FTELL	FTELL
FUNCALL	FUNCALL
GC	GC
GETDISABLED	GETDISABLED
GETFILTERACTIVE	GETFILTERACTIVE
GETFILTERSTR	GETFILTERSTR
GETISSORTED	GETISSORTED
GETLABELS	GETLABELS
GETMATCHFILTER	GETMATCHFILTER
GETORDERSTR	GETORDERSTR
GETWINDOWDISABLED	GETWINDOWDISABLED
GETWINDOWOPEN	GETWINDOWOPEN
HALT	HALT
IF	IF
INDENTMEMO	INDENTMEMO
INDEXBRK	INDEXBRK
INDEXBRK*	INDEXBRK*

---

---

INDEXSTR	INDEXSTR
INDEXSTR*	INDEXSTR*
INSMIDSTR	INSMIDSTR
INT	INT
INTP	Type predicates
LAST	LAST
LEFTSTR	LEFTSTR
LEN	LEN
LENGTH	LENGTH
LET	LET
LIKE	LIKE
LINE	LINE
LINES	LINES
LIST	LIST
LISTP	Type predicates
LISTTOMEMO	LISTTOMEMO
LOWER	LOWER
MAPFIRST	MAPFIRST
MAX	MAX
MAXLEN	MAXLEN
MEMO	MEMO
MEMOP	Type predicates
MEMOTOLIST	MEMOTOLIST
MESSAGE	MESSAGE
MIDSTR	MIDSTR
MIN	MIN
MOD	MOD
NEW	NEW
NEW*	NEW*
NEXT	NEXT
NOT	NOT
NOW	NOW
NTH	NTH
NULLP	Type predicates
onChange	onChange
onClose	onClose
onOpen	onOpen
OR	OR
PRINT	PRINT
PRINTF	PRINTF
PROG1	PROG1
PROGN	PROGN
PROJECTNAME	PROJECTNAME
RANDOM	RANDOM
REAL	REAL
REALP	Type predicates
RECNUM	RECNUM
RECORD	RECORD
RECORDS	RECORDS
RECP	Type predicates
REMCHARS	REMCHARS
REORDER	REORDER
REORDERALL	REORDERALL
REPLACESTR	REPLACESTR
REST	REST
RETURN	RETURN
REVERSE	REVERSE

---

RIGHTSTR	RIGHTSTR
RINDEXBRK	RINDEXBRK
RINDEXBRK*	RINDEXBRK*
RINDEXSTR	RINDEXSTR
RINDEXSTR*	RINDEXSTR*
ROUND	ROUND
SELECT	SELECT
SETCURSOR	SETCURSOR
SETDISABLED	SETDISABLED
SETFILTERACTIVE	SETFILTERACTIVE
SETFILTERSTR	SETFILTERSTR
SETISSORTED	SETISSORTED
SETLABELS	SETLABELS
SETMATCHFILTER	SETMATCHFILTER
SETMIDSTR	SETMIDSTR
SETORDERSTR	SETORDERSTR
SETQ	SETQ
SETQ*	SETQ*
SETWINDOWDISABLED	SETWINDOWDISABLED
SETWINDOWOPEN	SETWINDOWOPEN
SORTLIST	SORTLIST
SORTLISTGT	SORTLISTGT
SPRINTF	SPRINTF
STAT	STAT
stdout	stdout
STR	STR
STRP	Type predicates
SYSTEM	SYSTEM
TABLERNAME	TABLERNAME
TACKON	TACKON
TIME	TIME
TIMEP	Type predicates
TODAY	TODAY
TRIMSTR	TRIMSTR
TRUNC	TRUNC
UPPER	UPPER
VIEW	VIEW
VIEW*	VIEW*
WORD	WORD
WORDS	WORDS

## 1.358 MUIbase/Concept index

Concept index

\*\*\*\*\*

1:1 relationships	One to one relationships
1:n relationships	One to many relationships
ABConvert	ABConvert
Accessing record contents	Accessing record contents
Acknowledgments	Acknowledgments
Active object	Active object
Active table	Active object
Attribute functions	Attribute functions

---

Attribute management	Attribute management
Attribute object editor	Attribute object editor
Attribute objects	Attribute objects
Attribute trigger	Attribute trigger
Attribute types	Attribute types
Attribute types (table)	Table of attribute types
Attributes	Attributes
Author	Author
Balance objects	Balance objects
BetterString	Third party material
Bool attributes	Bool type
Boolean functions	Boolean functions
Browsing records	Browsing records
Buttons	Button
Change filter	Changing filters
Change order	Changing orders
Changing attributes	Changing attributes
Changing tables	Changing tables
Check data integrity	Check data integrity
Choice attributes	Choice type
Clear project	Clear project
Close project	Close project
Command syntax	Command syntax
Comparison function	Comparison function
Comparison functions	Comparison functions
Confirm quit	Confirm quit
Confirm save & reorg	Confirm save & reorg
Constants	Constants
Copying attributes	Copying attributes
Copying MUIbase	Copying
Corrupted database	Check data integrity
Creating attributes	Creating attributes
Creating tables	Creating tables
Custom classes	Third party material
Data retrieval	Data retrieval
Data types for programming	Data types for programming
Date attributes	Date type
Defining commands	Defining commands
Delete project	Delete project
Delete record	Deleting records
Delete trigger	Delete trigger
Deleting attributes	Deleting attributes
Deleting tables	Deleting tables
Disclaimer	Disclaimer
Display field	Display field
Display management	Display management
Distribution	Distribution
Duplicate record	Adding records
Empty display image	Empty display image
Empty order	Empty order
Entering bool values	Changing records
Entering choice values	Changing records
Entering date values	Changing records
Entering NIL value	Changing records
Entering reference values	Changing records
Entering time values	Changing records
Export records	Exporting records

---

---

External classes	Third party material
External editor	External editor
External editor for programming	External editor for programming
External viewer	External viewer
File format	File format
File format for import and export	Import file format
Filename attributes	String type
Filter	Filter
Filter examples	Filter examples
Filter expression	Filter expression
Font-name attributes	String type
Formats	Formats
Forward-backward search	Forward-backward search
Function triggering	Function triggering
Functional parameters	Functional parameters
Group editor	Group editor
Groups	Groups
Gui functions	Gui functions
How to register	Registration
I-O functions	I-O functions
Icon creation	Icon creation
Icon tool name	Icon tool name
Icons	Third party material
Image attributes	String type
Image editor	Image editor
Images	Images
Import and Export	Import and Export
Import file example	Sample import file
Import records	Importing records
Info	Info
Initial record	Records (concept)
Input requesting functions	Input requesting functions
Installing MUIbase	Getting started
Integer attributes	Integer type
Internal errors in database	Check data integrity
Kinds of programs	Kinds of programs
Label editor	Label editor
Lisp syntax	Lisp syntax
List functions	List functions
Many to many relationships	Many to many relationships
Masks	Masks
Mathematical functions	Mathematical functions
Memo attributes	Memo type
Memo context menu	Changing records
Memo functions	Memo functions
Memory consumption	Memory consumption
Menus	Menus
MUI	Third party material
MUI preferences	MUI
n:m relationships	Many to many relationships
Name conventions for program symbols	Name conventions
New attribute	Creating attributes
New record	Adding records
New table	Creating tables
New trigger	New trigger
onChange	onChange
onClose	onClose

---

---

One to many relationships	One to many relationships
One to one relationships	One to one relationships
onOpen	onOpen
Open project	Open project
Order	Order
Panel editor	Panel editor
Panels	Panels
Popups in cycle chain	Popups in cycle chain
Pre-defined constants	Pre-defined constants
Pre-defined variables	Pre-defined variables
Preferences	Preferences
Preferences file	Load and save preferences
Preprocessing	Preprocessing
Print structure	Print structure
Printing queries	Printing queries
Program control functions	Program control functions
Program debug information	Program debug information
Program editor	Program editor
Program include directory	Program include directory
Program output file	Program output file
Programming	Programming MUIbase
Programming language	Programming language
Programming virtual attributes	Programming virtual attributes
Project dependent settings	Project dependent settings
Project functions	Project functions
Projects	Projects
Query editor	Query editor
Query examples	Query examples
Quitting MUIbase	Getting started
Real attributes	Real type
Record delete requester	Record delete requester
Record filter	Record filter
Record functions	Record functions
Record memory	Record memory
Record-editing	Record-editing
Records	Records (concept)
Reference attributes	Reference
Reference filter	Reference filter
Register group editor	Register group editor
Register groups	Register groups
Registration	Registration
Relational operators	Relational operators
Relationships	Relationships
Reorder all records	Reorder all records
Reorganization	Save project
Root window	Windows
Save project	Save project
Search for	Search for
Search pattern examples	Search pattern examples
Search requester	Search requester
Select-from-where queries	Select-from-where queries
Semantics of expressions	Semantics of expressions
Sorting attributes	Sorting attributes
Sorting tables	Sorting tables
Space editor	Space editor
Space objects	Space objects
Starting MUIbase	Getting started

---

---

String attributes	String type
String functions	String functions
Structure editor	Structure editor
Swap records	Swap records
System functions	System functions
Table functions	Table functions
Table management	Table management
Tables	Tables
Tabular display	Data retrieval
Text editor	Text editor
Text objects	Text objects
TextEditor	Third party material
Third party material	Third party material
Time attributes	Time type
Tutorial	Tutorial
Type conversion functions	Type conversion functions
Type predicates	Type predicates
Type specific settings	Type specific settings
Type specific settings for attribute objects	Attribute object editor
Type specifiers	Type specifiers
User interface	User interface
Virtual attributes	Virtual
Why lisp?	Why lisp?
Window editor	Window editor
Windows	Windows

---