

MRQ

Matthias Bethke

COLLABORATORS

	<i>TITLE :</i> MRQ		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Matthias Bethke	January 19, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	MRQ	1
1.1	Index	1
1.2	introduction	1
1.3	features	2
1.4	requirements	2
1.5	installation	3
1.6	tooltypes	3
1.7	shellargs	6
1.8	configuration	6
1.9	examples	9
1.10	utilities	10
1.11	faq	10
1.12	history	11
1.13	bugs	13
1.14	legal	13
1.15	thanks	13
1.16	author	14
1.17	MagicUserInterface	14

Chapter 1

MRQ

1.1 Index

```
-----*****-----  
MRQ V1.12  
-----*****-----
```

```
Introduction  
Features  
Requirements  
Installation  
Tooltypes  
Shell Params  
Configfile  
Examples  
Utilities  
FAQ  
History  
Bugs / ToDo  
Legal babble  
Thanks  
Author
```

1.2 introduction

What it is and why I needed it

MRQ is a MUI-based system patch that tries to do everything¹ the well-known requester improver "ARQ" by Martin J. Laubach does - and a lot more.

ARQ has been around for years now and it always was among my personal "Top 5 Commodities", but I really wanted something a little more configurable. Most of its features are hardcoded, you can neither configure the ARexx interface nor the graphics nor the text scanner that chooses the graphics depending on the requester text.

See Features for how MRQ tries to change all this.

Well, there's a little drawback concerning asynchronous requesters. See the FAQ for more info!

1.3 features

Features

- Configurable like every MUI program (fonts, frames, group layout, ...)
- Complete keyboard control just like ARQ - the default button (i.e. the leftmost for intuition functions, configurable by the application in ReqTools) reacts on "Return", the rightmost on "Esc", and all buttons can be operated via the function keys (F1-F10 from left to right). The traditional lcommand-v/lcommand-b combinations still work of course.
- Requester texts can be scanned for arbitrary combinations of localized strings (identified by their catalog and string number, so a single config works independently of the used locale) and fixed strings. The text comparison can be both case-sensitive and -insensitive and can use AmigaDOS patternmatching as well as simple substring searches.
- Can decorate buttons with additional imagery (like checkmarks, red 'X'es etc.), also chosen according to the text on the button
- Manages an arbitrary number of images that are loaded via datatypes and remapped to the current palette. They can be loaded just-in-time and don't occupy any memory when no requester is open. Images can be of any depth, format and size (although it is of course not awfully smart to use huge pictures or slow formats like JPEG) If no predefined image matches the requester text, MRQ shows the calling program's icon if there is one. As a last resort, it looks for a file called "MRQ_DefaultImage" in the IMAGES directory (or in PROGDIR:MRQ-Images/ and S:MRQ-Images/ if the image directory is unspecified). Since V1.10, MRQ can not only load and remap truecolor images but also show them in hi-/truecolor if you use the appropriate screenmodes.
- Every image can be combined with an ARexx command, both command and port are configurable so you can send messages to any program when a particular requester pops up, even start programs or shellscrips through ARexx's system interface.
- also patches reqtools.library (only rtEZRequestA() at the moment)

1.4 requirements

MRQ requires at least

- a 68020 CPU
(it wouldn't have been a problem to compile MRQ for 68000, I just don't

think it makes sense on such systems though, it's just too slow. Time to upgrade, boyzngals!)

- AmigaOS 3.0 or higher
- MUI 3.x
- NewImage.mcc (still beta as of now, available at <http://www.linguistik.uni-erlangen.de/~msbethke/software.html>)
- NewImage.mcc requires guigfx.library and render.library (available at Aminet:dev/misc/guigfx.lha and Aminet:dev/misc/renderlib.lha respectively)
- optional: wbstart.library (Aminet:util/libs/WBStart.lha)

1.5 installation

Installation

Since V1.4 MRQ comes with an installer script. I tested it on a couple of different directories and it seems to work pretty well. Expect it to contain bugs anyway - it's constantly being developed and extended! In case you either don't have Installer V42.12 (minimum version required - you should get it from AmiNet anyway!) or the script doesn't work as expected, here's how to install MRQ by hand:

- Put MRQ somewhere on your HD, preferably the WBStartup drawer. Since V1.2 MRQ can be started from the Shell too, so the icon is no longer obligatory.
- copy MRQ.config to S: or to the same directory as MRQ.
- copy the "mrq-images" drawer anywhere on your HD (PROGDIR: and S: are searched automatically, if you put it anywhere else you have to tell MRQ through the IMAGES tooltype)

The installer script automatically sets the CONFIGFILE and IMAGES tooltypes in MRQ's icon. If you already have a previous version installed, the locations of config and image-dir are taken from the icon so you don't need to select the drawers on every update :-)

Since V2.0 of the installer script (MRQ V1.7) you may also use the installer to configure MRQ's tooltypes.

1.6 tooltypes

Tooltypes

CONFIGFILE

Tells MRQ where to find its configfile. If none is specified, MRQ looks for the file MRQ.config first in PROGDIR, then in s:

Example: CONFIGFILE=ENV:MRQ.config

DEBUG

Makes MRQ print quite a lot of information about what is happening, which will come in handy when there should be problems with certain programs.

ToolType usage:

- "DEBUG" alone: opens a console window where text is printed
- "DEBUG=T:MRQLog": send debug output to a file called MRQLog in T:

Shell usage:

- DEBUG alone doesn't work, it needs a value. For debug info to be printed on the shell where MRQ was started, use DEBUG=""
- Logfile usage is just like from WB

IMAGES

The drawer where you keep the image files for MRQ. This drawer is used if you specify a relative or no path in the configfile's "IMAGE" entries. Specifying no drawer has MRQ search first PROGDIR:, then S: for a directory called MRQ-Images.

Example: IMAGES=SYS:Tools/MRQ-Images

SAMEWIDTH

Tries to make all buttons in a requester the same width.

Default is to make them only as wide as the text they contain.

SAMEWIDTH-buttons will probably look more aesthetic to most people.

MOUSEREQ

Makes requesters open under the mousepointer.

Default is to open all requesters centered on their screen.

FRONTSCREEN

Tries to open requesters on the frontmost intuition screen.

This is a hack!!!

It is not OS-legal to open windows on alien non-public screens, therefore MUI defaults to opening its window on either the default PubScreen or one that was configured for the particular application. But a couple of tools have always opened their windows on screens they do not own, and for all current Amiga models/OS versions it works fine. Just omit this tooltype if it makes you uncomfortable.

QUALITY

The quality to use for remapping pictures. Specify one of "LOW", "MEDIUM", "HIGH" and "BEST". Defaults to "MEDIUM".

Note: on screens with lots of free pens "LOW" may actually give the best results as it avoids all dithering.

TRANSPARENCY

Controls whether color #0 of the requester image should be rendered transparent, so custom MUI background images can shine through.

SINGLEFRAME

Use a single frame for image and requester text instead of framing them separately.

SIZEABLE

Make requester windows resizable. Probably quite useless... :-)

CENTERTEXT

Center all texts in the requester window. Gives a better look if the gadgets are very wide and there's little text in the requester.

If there is only a single line in the requester, it will always appear

centered.

IMG_YES / IMG_NO / IMG_CANCEL

Names of images to put on MRQ's buttons in certain cases

Since V0.6b MRQ can optionally decorate buttons with images, much like in the requesters you might know from Windoze crates.

As of V1.6 two methods of deciding which image to show for which button are available:

- The simple one, without IBUTTONSBYTEXT:
if the IBUTTONSBYTEXT tooltype is unset, MRQ only uses the IMG_YES and IMG_NO images. IMG_YES is put on the leftmost button, IMG_NO on the rightmost. Simple as that.
- The advanced, with IBUTTONSBYTEXT:
with the IBUTTONSBYTEXT tooltype (see there for a description) you can set strings to scan for in the button texts. If a string is found, the corresponding imagebutton is used. So, similar to MRQ's choice of requester images, the button images actually depend on what is written on the button.

Of course you can use pictures of any format and size here, too. (tested with the checkmark and a 250x350 JPEG at the same time :-))

The IMAGES directory doesn't apply here, specify the full path and filename for all images!

IBUTTONSBYTEXT

To activate MRQ's feature to choose an image for a button based on what text is visible on the button, set this tooltype to a string of the following format:

- three fields for YES, NO and CANCEL respectively, separated by commas
- each field may consist of any number of strings, separated by pipe characters ('|').

Example: yes|ja|ok,no,cancel|back

This lets MRQ choose the image IMG_YES for buttons containing "yes", "ja" or "ok"; IMG_NO for ones containing "no" and IMG_CANCEL for buttons with "cancel" or "back" on them.

All strings are case-insensitive and need only occur somewhere on the button - so "note" matches "no", and "Say yes!" matches "yes".

NORTPATCH

Do not patch reqtools.library

AFTERPATCH

You will only need this tooltype if you are running some other patch to EasyRequestArgs() that conflicts with MRQ. Many people reported problems with things like AssignWedge that are not easily circumvented because

- 1) MRQ must have patched EasyRequestArgs() earlier than these and
- 2) MRQ takes pretty long to start up, so it's likely that other programs have already started when it applies its patches.

Using this tooltype you can have MRQ run any other program after it has installed its patches, thereby asserting the right order.

If the specified program (with spaces!) exists and has an icon, it will be started using Stefan Becker's wbstart.library. Otherwise MRQ splits the name at the first space, using the first part as the program name and the rest of the line as a parameter list.

Example: AFTERPATCH=C:AssignWedge SomeOption MoreParams

Example: AFTERPATCH=SYS:Tools/Multi CX

The first example starts a shell program, the second a WB process, given

"SYS:Tools/Multi CX" and its icon exists.

CAUTION: if you activate this option, MRQ can not be removed any more because this would certainly cause programs to jump into unloaded code!

AVOIDTASKS

Any process whose name matches the pattern given here will be redirected to the original library functions.

The patternmatching is case sensitive here!

Example: AVOIDTASKS=Thor|Hacky-#?|foo#?bar

means: don't use MRQ for Thor, anything starting in "Hacky-" or starting in "foo" and ending in "bar".

DEFAULTICON

MRQ can show the calling process' icon if the requester text didn't match any configured pattern and the program does have an icon. Set this tooltype to activate icon display. Without it, the default image is used.

1.7 shellargs

Shell Parameters

MRQ's ReadArgs() template when started from the Shell looks like this:

```
Configfile,
IMD=ImageDir/K,
BY=Button_Yes/K,
BN=Button_No/K,
BC=Button_Cancel/K,
IBBT=IButtonsByText/K,
QU=Quality/K,
AT=AvoidTasks/K,
DI=DefaultIcon/S,
MR=MouseReq/S,
SW=SameWidth/S,
FS=FrontScreen/S,
SF=SingleFrame/S,
TR=Transparency/S,
SI=Sizeable/S,
CT=CenterText/S,
NRTP=NoRTPatch/S,
AP=AfterPatch/K,
Debug/S
```

The corresponding tooltype for every parameter should be obvious, see there for further info!

Example Usage:

```
mrq IMD=Work:Graphics/mrq-images OB=s:mrq-images/MRQ_OK.brush
CB=s:mrq-images/MRQ_Cancel.brush RP=exact sw fs tr mr
```

1.8 configuration

Configuration

MRQ has a configfile that tells it how to behave. Here's all the keywords: (all but NEWCLASS may be abbreviated, the two-character abbreviation is given in ReadArgs()-syntax as <abbrev.>=<keyword>!)

NEWCLASS

This starts a new entry. For each image and ARexx-command you need to define one "event class" - just like the "delete", "printerstuff", "software failure" etc. classes know from ARQ, you can just have more of them. The following keywords each need an event class they belong to and thereby define this class' behavior.

IM=IMAGE

Specifies filename (and optional path) of an image to display when a requester of the current class is detected. The image can be of any size and any format you have a datatype for, but remember it will be loaded every time a requester pops up, unless you use the PRELOAD switch (see below), so don't use too big images/slow formats if you don't have a super-fast machine.

If you specify a full path here, MRQ will use this to look for the image file; otherwise the "IMAGES"-tooltype's value is prepended.

PL=PRELOAD

This is a modifier for IMAGE. It changes the default behavior of loading images every time a requester pops up to loading the image while the config is being parsed and keeping it in memory. This increases memory usage, speed, and the time to start up. Since V1.10 there is no need any more to load all pictures once on startup because their size isn't needed that early.

The default image is always preloaded.

TR=TRANSPARENT

A modifier for IMAGE as well, this tells MRQ to render the image with transparent background. Every pixel in color #0 is considered background. The global TRANSPARENCY option still works to enable transparency for all images.

RP=REXXPORT

The name of an ARexx port which MRQ should send a command to when opening a requester of the current class.

Default (i.e. if you only specify REXXCMD) is "PLAY".

RC=REXXCMD

Command to dispatch via ARexx. For example, if you want to keep using UPD as configured for ARQ, use something like "ID error_task_held" here.

Only one IMAGE, REXXPORT and REXXCMD should be specified for each class!

Strings a requester should be scanned for can be specified with the following two keywords, each of which may appear multiple times for each class:

ST=STRING

STRING needs only one argument: a string :-) If this string occurs as a requester's body text, it tells ARQ to use image and arexx command of the current class.

LO=LOCALE

LOCALE takes any number of arguments, the first of which must be the name of a locale catalog (e.g. "sys/devs.catalog") and the rest numeric arguments representing string numbers from that catalog.

See examples if you have no idea what this means :-)

If you do, well then, how do you get the locale catalog number of a given string? That's what dumpcat is for, see there its doc for more info!

To modify the behavior of the text scanner, STRING and LOCALE can be combined with a couple of switches as follows. Note that not every switch makes sense with both STRING and LOCALE!

PA=PATTERN

Use the AmigaOS patternmatching routines to compare the given string and the requester text. For a complete description of patterns see your AmigaOS manuals; some examples are here.

Can be used with STRING only.

SU=SUBSTRING

Simpler and less CPU-intensive than PATTERN, SUBSTRING only searches for the specified string at any position inside the requester text. SUBSTRING and PATTERN are mutually exclusive of course! (if both are found on one line, PATTERN is used and a warning printed if the debug console is open)

Can be used with both STRING and LOCALE.

NC=CMPNOCASE

Forces case-insensitive comparison.

Can be used with both STRING and LOCALE.

FO=FORMATTED

Don't search the text as input to the requester functions but the already formatted text as output into the requester.

To understand the difference, it's necessary to know that EasyRequestArgs()\$^1\$ (the function MRQ replaces) can construct requester texts from a format string and an argument list. Usually programs feed a localized format string to EasyRequestArgs() (it can look like "This is the %s with %d arguments" for example) and have certain placeholders (the percent-some-character things) replaced with arguments like "body text" and '2'. In this case it's just fine to search the format string for some pattern or substring to determine the correct requester image. But then, a few programs (among them the Workbench!) pass only a very general format string to EasyRequestArgs(), like "%s\n%s\n%s". The actual text is filled in with localized argument strings - the above format string can result in

"You MUST replace volume

Blah

in any drive!"

Of course you can't tell what the requester will look like from the format string alone, you have to scan the text as it appears in the requester, and that's what FORMATTED does. As there are always parameters that change from one requester to another (the volumename in the above example) you'll almost certainly want to combine FORMATTED with SUBSTRING to scan f.e. for the string

"You MUST replace volume" in the above requester.
FORMATTED can be used with both STRING and LOCALE.

\$^1\$Since V1.7 MRQ has been able to patch other functions, namely in reqtools.library. What is said here about EasyRequestArgs() equally applies to the other functions!

1.9 examples

Examples

Here's a short sample configuration that should make a few things clearer...

```
NEWCLASS
LOCALE hello.catalog 2 1 3
LOCALE test.catalog 5
STRING "Hello, (world|Brazil|Erlangen)?" PATTERN
STRING "good morning" NC SU
IMAGE hello.ilbm
REXXPORT "MYSOUNDPLAYER"
REXXCMD "play my_sample"
```

Let's have a look at the individual lines now:

NEWCLASS starts a new event class as described in Configuration. Following that is a LOCALE parameter defining three strings (numbers 1, 2 and 3 - order doesn't matter) from "hello.catalog" - a (hypothetical) catalog for a localized "Helloworld" program that might contain strings like "Hallo, Welt!", "Schweinewelt!" and "Wo soll das alles enden?" (in its german version :)). Now if any of these strings is found in a requester's text, this counts as positive identification of the current event class.

The next line adds another string from test.catalog to the list of strings that identify this event (you see, the total number of strings that identify a class and where they come from doesn't matter at all!).

Next is an explicitly specified (non-localized) STRING using case-sensitive patternmatching. The given pattern matches "Hello, world", "Hello, Brazil" and "Hello, Erlangen", with an optional character (an exclamation mark or something) at the end.

The string "good morning" has both the "case-insensitive" and "substring" switches set, that means anything like "good morning", "gOOD MornIng", "GOOD MOrnING", ... matches anywhere inside the requester text, no matter how much additional text is before or after the string (like, "A very nice GOOD MORNING to you all!" will match as well).

IMAGE should be easy to understand - it's simply the name of a picture file that should be displayed in the requester if any of the above strings is found. You need not specify a path as long as you keep all your pictures in the directory specified with the IMAGES tooltype, but you can if you want to. The last two lines aren't difficult either - REXXPORT and REXXCMD tell MRQ to send the command "play my_sample" to the port "MYSOUNDPLAYER" when a requester of this class opens.

1.10 utilities

There's currently only one utility that you'll need to configure MRQ:

dumpcat

Dumpcat is a tool to dump the contents of a locale catalog file to the shell (stdout). It takes the following parameters:

Catalog/A,Neg/S,Max/N

Catalog is simply the name of the catalog file of which you want to see the contents, "sys/devs.catalog" f.e.

Neg tells dumpcat to scan catalog numbers from zero downwards. Normal catalog files contain strings numbered from 0 or 1 upwards, sometimes with unused numbers for historical or other reasons. I have only come across a single catalog that contains strings with negative numbers (sys/dos.catalog) though, but if a catalog seems to be empty, trying "Neg" could be the solution.

Max is the maximum number of strings to scan, defaults to 65536

Theoretically, catalogs may contain strings under every index possible in a longword, but scanning all of them would mean 2^{32} calls to locale.library, so the range has to be somewhat smaller :)

Catalog strings are printed one per line, preceded with their number (the one you want for MRQ's config!) If it finds newline characters in a string, it replaces them with the C-notation for newlines, Backslash-n ('\n').

Dumpcat will hopefully be obsoleted by the coming Prefs Editor which will handle all this cryptic stuff internally so you only need to click on the string you want and have its number and catalog name stored in the configfile.

1.11 faq

Frequently Asked Questions

Q: A few requesters like "delete" and "copyright" have the correct images but most show only the default image. what's wrong?

A: Most likely there are some catalog files missing from your LOCALE:Catlogs/sys directory because you are running an english system and all texts are english by default so there don't need to be any catalog files. I haven't had the time to make an extra config for you yet - it requires getting all the messages that are now encoded als LOCALE config lines and writing them verbatim into the config. If anybody wants to do this - go ahead, I'll include the config in the next release!

Q: Why don't AssignWedge and similar functions in MCP etc. work any more?

A: These programs also patch EasyRequestArgs(). Some (e.g. the original AssignWedge by Olaf Barthel) will work fine if started after MRQ. That's because MRQ replaces the entire EasyRequestArgs-function with code of its own and never calls the old function back like most patches do, so previously installed patches won't get called any more.

Since V1.8 there's the AFTERPATCH tooltype to have programs started by MRQ after the patches were applied.

Q: Why can I resize but not snapshot an MRQ window?

A: I tried to make this possible but it didn't do no good at all. The problem is that I have to assign a MUI object-ID to windows which is the same for all requesters - and MUI remembers the size of each window that has an ID for the next time it is opened. So when a requester with lots of text appears, the window will naturally be large. But subsequent requesters won't shrink, they are always as big as the largest requester you had open before - they have the same ID after all! This sucks, so I didn't use it...

Q: Why is MRQ so slow?

A: Both MUI and the datatypes system are not quite the fastest components in AmigaOS. They're programmed for versatility, not speed. You can do something to speed them up though:

- use PRELOAD for frequently used images. They will be kept in memory and no slow datatypes components are involved with their display.
- don't use too many patterns and backgrounds in MUI (looks ugly anyway :))
- don't use slow formats like GIF (let alone JPEG!) for your images

Q: What about patches for some other functions like the intuition async requester functions?

A: I tried to patch the async requesters but found it is unwise to do so. This is because they have to return a window pointer, and the window I open is managed by MUI. It is to be expected that the caller uses the async function to mess with the window somehow, and MUI is pretty allergic to that.

Since V1.7 MRQ can also patch regtools.library, currently only rtEZRequestA(), but I'll have a look for other possibilities soon.

1.12 history

History

V0.1	07-Nov-97	-	Ancient history :)
V1.8	20-Sep-98		
V1.9	01-Dec-98	-	changed DEBUG usage: you can now specify a file to send debug output to
		-	removed the exit delay when a debug console was open - if you want to read the output later, send it to a file or use the TEE: handler!
		-	changed requester on exit to a standard MUI requester.
		-	Bugfix: tiny bug in the search-that-darn-config routine removed

- Bugfix: pr_WindowPtr misinterpreted, removed all checks for that.
 - Bugfix: AT LAST! MRQ works with non-CGfx datatypes! These nonstandard bitmaps required some tricks :-((be assured: nothing illegal!) This also means: no more weird-looking imagebuttons!
- V1.10 05-Mar-99
(Homepage only)
- Linebuffer for configfile reading increased to 512 characters
 - Now finds SetMan and doesn't warn on exit any more if it is installed.
 - fixed the function to check AVOIDTASKS, now works with strange programs like THOR where GetProgramName() returns success and an empty name
 - the <RETURN>-key-handling has changed somewhat:
 - MRQ now activates the default button so it can be rendered with a colored frame by MUI.
 - there is no fixed binding of <RETURN> to the default button any more, so you can change the active object with the tab key (this has been possible ever since but before there was no active object until the first <TAB>)
 - if the application requests that no button should react on <RETURN>, no button will be activated when the requester opens. You can still cycle through them with <TAB> and press <RETURN> just like before though.
 - major rework of the graphics system! Now uses NewImage.mcc for much cleaner image handling. See [http://www.linguistik.uni-erlangen.de/~msbethke/](http://www.linguistik.uni-erlangen.de/~msbethke/software.html) ↵ software.html for the latest versions!
 - as a result, MRQ now supports truecolor images, displayed in all their glory on hi-/truecolor screens!
- V1.11b 11-May-99
(Homepage only)
- added DITHER AND PRECISION tooltypes
 - bugfix: errors while building the MUI object tree were not handled correctly at all
 - bugfix: printed garbage instead of program name if started from the shell and MRQ already running
 - replaced DITHER and PRECISION with NewImage.mcc's new QUALITY parameter.
 - Added DEFAULTICON option: if a requester text doesn't match any class, MRQ can now show the calling program's icon (if any) instead of the default image.
NOTE: this doesn't seem to work well right now. I don't know yet whether the fault is MRQ's or NewImage.mcc's.
- V1.12 21-Jan-00
- Fixed a few bugs that could cause Enforcer hits if function parameters were incomplete (which is legal in some cases for ReqTools)
 - Again: fixed a problem with pr_WindowPtr
 - Adapted to latest version of NewImage.mcc
 - Showing icons *does* work now! :)
-

1.13 bugs

Stuff that should really be fixed

- there's a pretty good chance for a crash if you quit MRQ via CX Exchange while there are still requesters open. Wanna make this foolproof so it will have to use semaphores. RSN!

Things planned for future releases (roughly in order of priority)

- real prefs editor with IFF configfile
- animation.datatype support
Had a hell of a time trying to integrate real BOOPSI objects in MUI windows already. No success so far, I'm not sure if it's possible at all. Now with NewImage.mcc things seem to get easier again...
- ...anything else? Send suggestions!

1.14 legal

MRQ is freeware

You know what that means, don't you?

If you don't - well, I want to spare myself the usual kilobyte-long disclaimers:

I don't guarantee anything, so if MRQ causes you damage of any kind, it's entirely your own problem. You get what you pay for.

Of course I tried to make it as bugfree as possible, and if you tell me about any bugs that show up on your system I will probably consider fixing them. Don't rely on it though - installing MRQ on a mission-critical system is definitely a Very ↔
Bad

Idea(tm)! But you knew that already :-)

1.15 thanks

Thanks go to:

- Martin Laubach for ARQ
 - Stefan Stuntz for MUI
 - Timm S. Müller for guigfx.library and a lot of help with using it
 - Jonas 'Zaphod' Petersson for UPD
 - Tony Matthews for the LinuxBrushes archive
 - Stefan Becker for wbstart.library
 - the SAS/C blokes for great work
 - all bugreporters for constructive criticism
 - Unconscious Collective, XIS, Hallucinogen, Acid Junkies and the Green Nuns of the Revolution for incredible sounds
 - Cris for being just wonderful
-

1.16 author

Author

Send comments, suggestions, gifts, flames, files etc. to:

email: Matthias.Bethke@gmx.net

smail: Matthias Bethke
Haagstr. 5
91054 Erlangen
Germany

Homepage: <http://www.linguistik.uni-erlangen.de/~msbethke>

Software page: <http://www.linguistik.uni-erlangen.de/~msbethke/software.html>

1.17 MagicUserInterface

This application uses

MUI - MagicUserInterface

(c) Copyright 1993/94 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