ARexxBox

ARexx Interface Designer

Michael Balzer

Internet: balzer@heike.informatik.uni-dortmund.de
Zerberus: M.BALZER@AWORLD.ZER

Wildermuthstraße 18
W-5828 Ennepetal 14
GERMANY

June 20, 1992

Contents

1 Legal Stuff **1**
- 1.1 Copyright . . . 1
- 1.2 Distribution . . . 1

2 Introduction **1**
- 2.1 Why ARexxBox? . . . 1
- 2.2 System Requirements . . . 2

3 Using the Program **2**
- 3.1 Input . . . 2
- 3.2 MsgPort Basename . . . 3
- 3.3 CommandShell . . . 3
- 3.4 Merge in . . . 3
- 3.5 Print . . . 3
- 3.6 Generate Source . . . 4

4 The Generated Source **4**
- 4.1 Files . . . 4
  - 4.1.1 Basic Module . . . 4
  - 4.1.2 Headerfile . . . 4
  - 4.1.3 The Data Structures . . . 4
  - 4.1.4 Interface Module . . . 5
- 4.2 Interface Functions . . . 5
- 4.3 Results . . . 5
- 4.4 Errors . . . 6
- 4.5 The Main Program . . . 6

5 Other Stuff **8**
- 5.1 Thanks . . . 8
- 5.2 Bug Reports and Comments . . . 8

1 Legal Stuff

1.1 Copyright

ARexxBox is copyright ©1992 by Michael Balzer. All rights reserved. ARexxBox is public domain software, there are no fees, payments, licenses or the like necessary, neither for the ARexxBox itself nor for the generated code.

But — beeing a poor student :-) — I certainly appreciate any donations.

If you use ARexxBox or its generated code to build an ARexx interface for some program (commercial, shareware or public domain), you must note this in the manual or the about requester of the program.

Additionally I would like to hear about what kind of project you have done with ARexxBox and what rexx commands your application supports. Please email me.

1.2 Distribution

ARexxBox is public domain, you may copy it and distribute it as long as all files of the original distribution remain together and intact. I allow and encourage distribution on electronic services.

With the exception of distributing this software on Commodores Native Developer's Upgrade Kits and their electronic equivalents, the ARexxBox *must not* be sold commercially.

Some german PD authors think about setting up a set of rules for selling PD software. This set of rules will most probably apply to prices of single disks as well as prices of collections such as the AmigaLibDisk CD ROMs.

I herewith declare the ARexxBox to fall under all clauses of this not yet existant set of rules which apply to copyrighted PD software.

Most probably will the selling of PD discs with prices higher than 5 DM / 5 US$ or as part of a CD ROM for more than 30 DM / 30 US$ become illegal, so dealers should take care of this.

2 Introduction

You wouldn't read this manual if you were not already convinced of the importance and power of ARexx on the Amiga. So I only have to convince you of using ARexxBox to implement ARexx interfaces for your programs ;-).

2.1 Why ARexxBox?

ARexxBox (inspired by the GadToolsBox by Jan van den Baard) is a tool, which greatly simplifies the design and implementation of an ARexx interface for a program.

Some of the features are:

- Syntax and results of the arexx commands follows exactly the conventions suggested by the Style Guide, e.g. command arguments are parsed using the AmigaDOS ReadArgs function, and the keywords VAR and STEM are supported automatically.
- Each command may have an unlimited number of arguments and results.
- All ReadArgs template options are allowed for arguments and are supported *automatically*. For results, the options /N and /M are supported.
- Graphical user interface (designed using GadToolsBox :-).
- Creates C–Source: One module containing necessary basic functions and another containing the interface stub routines which you only have to add your code to.
- If desired, ARexxBox can generate source for CommandShells, shells in which the user may enter arexx commands directly and view their output.
- CommandShells can be used to execute external macros.
- A program may open as many arexx ports and CommandShells as the system provides message ports.
- I already entered the standard commands suggested by the Style Guide. Example code for some of them is also included.

2.2 System Requirements

There is only one real requirement for ARexxBox as well as the generated code: AmigaOS 2.04 at least. There are no hardware requirements, ARexxBox should run on any Amiga computer.

Okay, there is *one* requirement: If you have only non-interlaced NTSC resolution, you have to switch overscan on to see the ARexxBox window. Or set the system font to some 6 or 7 point sized...

You should get a copy of Nico François' reqtools.library if you don't already have one.

To compile the generated source, you need an ANSI compatible C compiler which is also able to import functions from `amiga.lib`. You need at least version 37.32 (12.11.91) of the `amiga.lib` to be able to link the code.

Version 37.32 of `amiga.lib` comes with the Native Developer's Upgrade V37.4 (11/91), or maybe already came with your compiler.

3 Using the Program

Using ARexxBox is quite normal amiga standard. If you ever used an amiga application, you can use ARexxBox .

3.1 Input

For entering some command definition, I suggest the following procedure: Press N, enter the name of the ARexx command and then press Return. ARexxBox now converts the name to upper case, changes illegal character codes to underscores ('_') and checks if

the name is unique. If there is already such a command in the list, you can correct the name.

Then enter the arguments. For each argument press E and enter the name. ARexxBox will check this name too.

Proceed the same way with results, except the shortcut for a new result field is W.

You may move arguments and results around after selecting them, using the corresponding 'up' and 'do' gadgets, or you could delete them using the corresponding 'Remove' gadget.

Enter template options in ReadArgs style (see AutoDocs for DOS), for example `ARG1/K/N` or `LIST/M`. Only options `/M` for list and `/N` for numeric type are allowed for result fields.

3.2 MsgPort Basename

The 'MsgPort Basename' is the default base name for any opened arexx message port of the application. The generated functions automatically append some incrementing number to this name until a port can be opened without conflict.

The application should provide (as shown in the demo code) an argument or tooltype named "PORTNAME", so the user can set the base name to something else. The new portname will be handled exactly the same way (see sample code).

Additionally, 'MsgPort Basename' is used to identify the ARexx scripts belonging to that application: It becomes the default file extension for the application's ARexx programs.

3.3 CommandShell

Selecting this gadget switches code generation on for the CommandShell functions. You normally should leave this option selected, as the CommandShell functions can also be useful for executing external macros.

3.4 Merge in

This function merges another ARexxBox binary file into the currently loaded one. Only functions not yet defined will be included. You can use this to create complex interface definitions from various small modules of commands importing only those functions needed.

3.5 Print

Is intended to be an aid in documentation of your interface. For each command, the name, syntax and results will be printed.

Suggestions for another (better) format are very welcome.

3.6 Generate Source

Starts the source generator. The selected filename will be cut to it's base name (removing any extension `.c, .h or _rxif.c`), and then will be checked for existance and coexistance with other files to keep you from destroying your work.

If a file `name_rxif.c` exists, ARexxBox asks if you are sure to overwrite this. This question is essential, as there is probably code of yours in that module, so you should first save this in another file. Don't answer this question too fast!

You have to merge the functions of the old and the new rxif modules after generating the new one.

ARexxBox generates three files: *name*`_rxif.c`, *name*`.c` and *name*`.h`.

4 The Generated Source

4.1 Files

4.1.1 Basic Module

The basic module will be named *name*`.c` and should not be edited. It contains all basic functions of the ARexx interface, functions for sending ARexx commands, for managing rexx hosts, the ARexx dispatcher and maybe the functions for the CommandShell.

You can find the exact descriptions of these functions in the included autodoc style formatted file `ARexxBox.doc`.

4.1.2 Headerfile

The header file will be named *name*`.h`. It contains all structs and declarations of basic functions and rexx interface functions as well as the definitions of the corresponding data structures for parameters. Have a look at such a file to learn more.

The define `REXX_EXTENSION` contains the standard file extension for ARexx programs belonging to your application. So a `SendRexxCommand(host, "test")` will try to start the ARexx script `test.<rexx_extension>`. The extension is the same as the name used for message ports, which in turn should be the common project id (see Style Guide) for your application.

The RexxHost structure contains a pointer to the message port, the ports name, a reply counter for unreplied messages and a pointer to a `RDArgs` structure for argument parsing. For each new host, the management function `SetupARexxHost()` will create a new instance of this structure.

The string variable `RexxPortBaseName` contains the base name for all message ports.

4.1.3 The Data Structures

Each ARexx command has a corresponding rxif rexx interface function with an attached data structure for the parameters and result codes. This structure consists of at least the two return code fields rc and rc2. If there are arguments to the function, the structure will

also have a substructure called 'arg' and if there are result fields, they will be contained in another substructure 'res'.

The substructures contain one variable for each entry in the corresponding list (args or results), the variable name is derived from the name in the list by converting the name to lower case (and removing the options).

Example: An argument `FILENAME/A/K` can be accessed from C through the variable `rxd_struct.arg.filename`.

The type of a variable will be `char *` for text strings, `long *` for numeric values (option /N). Boolean values (option /S or /T) will be simple `long` types.

Lists (option /M) will be represented by a pointer to an array of `char *` resp. `long *` (therefore `char **` resp. `long **`). The lists may have any length, the end of a list is marked by a NULL pointer in the array.

The types and meanings of the result fields are following exactly the same manner.

4.1.4 Interface Module

The interface module (named *name*`_rxif.c`) contains one function body for each defined ARexx command, in which you normally just can insert your special code and that's it.

4.2 Interface Functions

Each interface function will be called three times by the parser, first time for allocating memory for the parameter structure and inserting default values there, second time to actually do the work[1], and third time for freeing the allocated memory.

Thus it is possible to use your own local variables without hurting the reentrance ability (for multiple hosts). The local variables must be added to the parameter structure. To do this, you just define a new local structure with the original structure beeing a substructure of the new one, like this:

```
void rx_help( ..., struct rxd_help **rxd, ... )
{
    struct myrxd {
        struct rxd_help rxd;
        /* insert local variables here! */
        ...
    } *rd = (struct myrxd *) *rxd;

    ...
}
```

The code tells you, which part of an interface function will be called when.

4.3 Results

The results of an ARexx command have to be assigned to the fields of the — if existant — substructure `res` of the `rxd` structure.

[1] Rem.: This call will only occur if the parsing was okay

The names and data types of the fields correspond to the names and template options of their command definition.

So you alway return *pointers* to data, pointers to charaters or longs or arrays of such pointers, where a NULL entry marks the end.

The formatting of the results according to the user–desired form will be done by the ARexxBox routines. If none of the keywords VAR or STEM was given, the result will be in VAR format and will be placed into the common ARexx variable `RESULT`.

The STEM format consists of a list of assignments, which each have a variable name and the assigned value. A variable name is built from the supplied argument for STEM and the name of the result field followed by a dot and then either the word "count" or an index number.

The first list entry will always be "count", yielding the number of values in the list.

Example: The command `INHALT` has a result `INHLISTE/M`. An ARexx call of the form "`INHALT STEM TEST.`" will produce a result list of the form:

```
TEST.INHLISTE.COUNT = <n>
TEST.INHLISTE.0 = <first entry>
TEST.INHLISTE.1 = <second entry>
...
TEST.INHLISTE.n-1 = <last entry>
```

The VAR format is derived from the STEM format by concatenating all list entries to one string, separated by SPACEs.

Please note, that *you* have to take care of SPACEs in your results, if there are any, you should put quotes around the string before returning it to ARexx.

4.4 Errors

If errors occur during execution of a command, they have to be returned to the user somehow to give him a chance to react properly.

ARexx normally only supplies the variable RC for this purpose. RC may contain a number, normally from 0 to 20, indicating a severity level.

The ARexxBox offers an extended method for this purpose. If an error occurs, the rexx routines automatically generate a new variable called RC2. This variable can hold error codes as well as error strings (descriptions), which can be used by the calling program or can be output to the user.

For technical details see `ARexxBox.doc`.

4.5 The Main Program

The main program has'nt much to do at all to get the rexx host up and running.

It has to open `exec.library`, `dos.library` and `rexxsyslib.library`, the first two of which will normally be opened by your compiler's startup code, so you just have to open the rexx library.

Then you can create an ARexx host using the function `SetupARexxHost()`. You may optionally give it another base name for the message port. All there remains to do is including the new port into the normal event loop and calling the arexx dispatcher for messages from the host's port.

A CommandShell gets an input, an output (not necessarily different) and a prompt string and then runs until EOF. It is also possible to start a CommandShell asynchronously using `dos.library / CreateNewProc`).

The following is a (minimal) demonstration program:

```
#define HOSTSIG(host) (1L << host->port->mp_SigBit)

void main( int argc, char *argv[] )
{
    struct RexxHost *myhost;
    BPTR fh;

    /* Init */

    if( !(RexxSysBase = OpenLibrary( "rexxsyslib.library", 35 )) )
    {
        printf( "No RexxSysLib\n" );
        exit( 20 );
    }

    /* open Host */

    if( !(myhost = SetupARexxHost(NULL)) )
    {
        printf( "No Host\n" );
        exit( 20 );
    }

    /* Number 1: The CommandShell... */

    if( fh = Open( "CON:////CommandShell/AUTO", MODE_NEWFILE ) )
    {
        CommandShell( myhost, fh, fh, "test> " );
        Close( fh );
    }
    else
        printf( "No Console\n" );

    /* Number 2: 'Real' ARexx mode */

    printf( "Address me on Port %s!\n", myhost->portname );
    printf( "Cancel me with CTRL-C\n" );

    while( 1 )
    {
        long s = Wait( SIGBREAKF_CTRL_C | HOSTSIG(myhost) );

        if( s == SIGBREAKF_CTRL_C )
```

```
            break;
        else
            ARexxDispatch( myhost );
    }

    CloseDownARexxHost( myhost );
    CloseLibrary( RexxSysBase );
    exit( 0 );
}
```

5 Other Stuff

This manual can't give you a detailed introduction to ARexx. A look at the corresponding chapters in the "User Interface Style Guide" by Commodore and the "ARexx User's Reference Manual" by William S. Hawes is strongly recommended.

The ARexxBox has been developed on an Amiga 2000 equipped with A2630, 7 MB RAM, GVP Series II with Quantum P210S and Seagate ST296N and a MultiVision flickerfixer plus NEC Multisync 3D.

ARexxBox was compiled using Manx's Aztec C 5.2a, I somehow got used to the thing...

Pleese ekscuse mei bäd inglisch.

5.1 Thanks

I would like to thank the following persons:

- My active beta testers: Ralf Kaiser, Garry Glendown, Wolfgang Kueting and Stefan Zeiger.
- The Amiga–Crew of Commodore, for the Amiga and AmigaOS Release 2.
- Jan van den Baard, for his excellent GadToolsBox, which inspired me to program this tool and was a big aid for designing the user interface.
- Nico François, for his ReqTools.Library with it's really programmer– and user–friendly requesters.
- William S. Hawes for his marvelous Rexx port.

5.2 Bug Reports and Comments

Please send me your bug reports and comments through email if possible!

Please *always* tell me about your exact configuration when writing a bug report, and *always* provide a *minimal* sample program which reliably reproduces the bug! Describe the error as good as you can, this helps me in correcting it.

Comments are very welcome!

Michael Balzer