ARx_Intro.ag

COLLABORATORS

	TITLE : ARx_Intro.ag		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 17, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ARx_Intro.ag

1.1 ARexxGuide | Introduction

```
                AN AMIGAGUIDE® TO ARexx                 Second edition (v2.0)
by Robin Evans

Preface
```

1.2 ARexxGuide | Preface (1 of 5) | ACKNOWLEDGEMENTS

Acknowledgements
~~~~~~~~~~~~~~~~
A special thank-you to those who sent in registration fees for the first edition, especially Dean Adams who not only registered, but sent extensive and helpful errata reports after doing so, and who provided invaluable guidance for and testing of the utility programs for this edition.

I thank the many people who have written ARexx scripts over the years and released them publicly so that all of us could study and learn, to the folks on GEnie for asking challenging questions about ARexx and providing insightful answers, to Richard Stockton and Gramma's BBS (206-744-1254) -- not only one of the best sources of information on ARexx, but also one of the most incredible applications of the language -- and to those who provide fascinating discussions on Usenet's comp.lang.rexx of the many flavors of REXX. Thanks to Eddie Churchill of Inovatronics for pointing out that the guide already had over a dozen tutorial examples that weren't listed in the 'Tutorial' section. They are now listed in the expanded Techniques section.

The development of the first edition was aided greatly by Ed Patterson who had the patience to look through rough and early versions and provided invaluable suggestions. Thanks to Larry Shanahan for extensive link-testing and for raising several Multiview-compatibility issues.

Next: REFERENCES | Prev: Introduction | Contents: Introduction

1.3 ARexxGuide | Preface (1 of 5) | REFERENCES (1 of 1)

References
~~~~~~~~~~
The following works were used as references in preparing this guide.

ARexx User's Reference Manual, The REXX Language for the Amiga
by William S. Hawes
Wishful Thinking Development, 1987 (with disk-based updates)

```
The REXX Language, A Practical Approach to Programming
   by M.F. Cowlishaw
   TRL1:
      Prentice-Hall, First edition: 1985
      ISBN 0-13-780735-X  Prentice-Hall
   TRL2:
      Prentice-Hall, Second edition: 1990
      ISBN 0-13-780651-5

Using ARexx on the Amiga
   by Chris Zamara & Nick Sullivan
      Abacus, 1992
      ISBN 1-55755-114-6

The REXX Handbook
   edited by Gabriel Goldberg & Philip H. Smith III
      McGraw-Hill, 1992
      ISBN 0-07-023682-8

Extending ARexx (and other articles)
   by Marvin Weinstein
      AmigaWorld Tech Journal, 1991-1992
```

Next: AUTHOR | Prev: ACKNOWLEDGEMENTS | Contents: Introduction

1.4 ARexxGuide | Preface (1 of 5) | Author

```
                The author
~~~~~~~~~~
      No, the only perfect solution would have been the sixteen
      pockets, symmetrically disposed, each one with its stone. Then
      I would have needed neither to number nor to think, but
      merely, as I sucked a given stone, to move on the fifteen
      others, each to the next pocket, a delicate business
      admittedly, but within my power, and to call always on the
      same pocket when I felt like a suck.

                     --- Samuel Beckett, Molloy
```

Mr. (as in Robin Hood, in case you were wondering) Evans has, in his checkered past, sprayed asbestos onto the ceilings of mobile homes and designed databases for lawyers (two activities that seem somehow related). Between those periods, he edited a biweekly community newspaper in Seattle and typeset enough headlines and stories on a CompuGraphic EditWriter 7500 to realize that computers could be nifty things after all.

He was a serious enough philosophy major in college (a time when he thought that computers just might be evil) that he can still give a passable definition of "epistemology" or "ontology" (although he can’t always spell the words correctly.)

He moved up from a C-64 to his first Amiga (a 500) in 1988 and bought ARexx shortly thereafter. When he discovered what could be done with ARexx by itself and ARexx with TxEd, he was hooked.

Evans writes occasional articles on ARexx for ViewPort, GEnie’s monthly online Amiga magazine and teaches a class in REXX/ARexx at CALC, the online college available on GEnie. (Write to either of the addresses listed in the
Distribution
node for more information on the class.)

Next: Distribution | Prev: REFERENCES | Contents: Introduction

1.5 ARexxGuide | Preface (1 of 5) | Distribution restrictions

Distribution
~~~~~~~~~~~~~

```
Robin Evans                   14 Mar 1994

   from Internet:
      robin@halcyon.com
      r.evans6@genie.geis.com

   on GEnie:
      R.EVANS6

   by phone:
      (206) 682-7077
```

Next: COMPATIBILITY | Prev: Author | Contents: Introduction

1.6 ARexxGuide | Preface (1 of 5) | Compatibility with the REXX standard
~~~~~~~~~~~~~

ARexx variations from the REXX standard

REXX is a language used on many platforms. The current version of ARexx (1.15) distributed with the operating system was developed in the late 80’s but differs in some fundamental ways from the definition of the REXX language that was available at that time. It differs even more from the current standard.

The variations will be of little more that academic interest to those who use ARexx only, but will be of more concern to those who wish to develop scripts that can be used on the Amiga and on other systems, those who wish to use a script written for another system on the Amiga, or those who just like standards (but then, those folks aren’t likely to use Amiga).

Many of the differences between ARexx and the REXX standard are mentioned throughout this guide. Notations on compatibility issues are extensive, but not comprehensive.

What is the REXX standard?

The REXX language was developed by an IBM engineer named Mike Cowlishaw in the early 80’s on mainframes using the CMS operating environment. It gained quick acceptance in that environment and is still widely used there. IBM soon made the language part of its System Application Architecture (SAA), offering versions of the language to users of other IBM operating systems.

Mr. Cowlishaw described the purpose, syntax, and requirements of the language in a well-written book first published in 1985 -- The REXX Programming Language, A Practical Approach to Programming. That book (called TRL1 by the REXX cognoscenti) defined the first ‘standard’ for the language. A second edition of the book (called TRL2), was published in 1990. That edition has become the current standard for the language, but will soon be displaced by a formal definition from an ANSI standards committee.

The X3J18 REXX Standards Committee is working on a formal standard for the REXX language, using TRL2 as its base document. The Committee meets 3 or 4 times a year and holds ongoing discussions throughout the year by electronic mail.

The purpose of a standard

REXX is available in versions developed by IBM for virtually all IBM operating systems, including OS/2. It has also been developed by independent implementers for a wide range of other systems including MS-DOS, Windows, Unix, and -- of course -- the Amiga.

The reason for a standard definition of a programming language is to limit the differences that are encountered when using the language on different systems; to define a basic set of programming constructs that can be used on any of the systems. Each system would still offer extensions to the language to fit the current environment, but the basic syntax and fundamental constructs would be the same.

Next: INTRODUCTION | Prev: Distribution | Contents: Introduction

1.7 ARexxGuide | Introduction (1 of 5) | THE FIRST PROGRAM

It has become a mantra -- two simple words that have been repeated so often in so many programming languages that they seem to have become an incantation. It is the First Program, the Foundation.

Let us not delay. Here it is:

```
      /**/
      SAY 'Hello, world.'
```

That is an ARexx program, total and complete. The first line (a comment) is vital, but doesn't do anything. The second line, though -- that will cause ARexx to output to the current shell the words of the mantra.

In the jargon of ARexx, the second line is an instruction ; SAY is a keyword ; the quoted words form a string . The jargon is useful and the reader who has been pressing buttons has discovered that it is both used extensively and explained at length elsewhere in this guide, but the jargon used to define the language is not the heart of ARexx. The program is. That mantra -- or, rather, the means used to produce the mantra -- is the heart of the language.

Let us repeat it for good luck:

```
      /**/
      DO 5
         SAY 'Hello, world.'
      END
```

There we have a control structure that produces an iterative loop . What is important, though, is what it does: it repeats 5 times an instruction that is written out only once. ARexx offers a wealth of such control structures.

This guide seeks to explain the elements of the language, not because the jargon used to describe such things has any importance in its own right, but because through understanding how the language works the programmer will be better able to use the language to do just what she wants.

Next: WHY AREXX | Prev: Introduction | Contents: Introduction

1.8 ARexxGuide | Introduction (2 of 5) | WHY AREXX?

Why spend time learning a programming language like ARexx? For one thing, it's easy. REXX -- the language from which ARexx was born -- was designed to be easily accessible to non-programmers while still providing the power and structure expected by the seasoned programmer.

ARexx on the Amiga can make the machine more powerful and more personalized, not just through its own resources as a programming language, but also by controlling other programs with ARexx scripts.

Most commercial productivity programs for the Amiga use ARexx to add features to the program. The heavyweight image-processing tools make extensive use of ARexx as do the two major desktop publishing programs. Most word processors available use ARexx as their macro language.

ARexx does more than just replace the often-bizarre syntax of macro languages. It also allows different programs to communicate with one another. For instance, a bitmap picture in Professional Page can be sent to ADPro for manipulation and reinserted in Pro Page by simply choosing the name of an ARexx script in PPage. The PPage script then sends the necessary commands to ADPro to load the image, and could even be made to begin an automatic processing operation in ADPro. (Similar tasks could be performed with ImageMaster or ImageFX as well as with version 3.0 of Pagestream.)

ARexx itself doesn't know anything about anim files or iff files since the language was developed over a decade ago on an IBM mainframe that didn't have such wonderful things. It can, nonetheless, be used to control the creation of an animation in a program like AdPro that does know about iff and anims.

Many PD and shareware programs provide ARexx support as well. One of many fascinating programs available from your favorite network or BBS is SQLdb by Kyle Saunders. It is a shareware database engine using the industry-standard SQL language. It can be used without ARexx, but becomes part of a powerful database management system when used as a SQL server with ARexx.

Modems or other peripherals attached to the serial port can be controlled through ARexx with the help of functions provided by rexxserdev.library.

Another example is EdPlayer, a MED/MOD player. It also works fine without ARexx, but becomes more powerful when combined with ARexx. A list of chosen mods can be saved for later playback, but only through ARexx. An ARexx script available in some electronic file libraries will randomly choose mod files in a specified directory and play them using EdPlayer.

From the time ARexx was introduced, text editors have led the way as innovative ARexx implementations. With the help of ARexx and a terminal program that also communicates via the language, a text editor like TurboText, Edge, CED, or a handful of others can become a complete text engine, serving as a customized front-end for any communications service.

ARexx Applications list
~~~~~~~~~~~~~~~~~~~~~~~

A comprehensive list of programs that support ARexx commands, of add-on function libraries, and of other ARexx utilities is maintained by Daniel Barrett of the University of Massachusetts, Amherst. In February, 1994 the ARexx Applications List included over 370 products.

Look for the list on your favorite computer bulletin board. Those with direct access to Internet services can get a copy of the list by anonymous ftp from any Aminet site (such as wuarchive.wustl.edu) in the directory /pub/aminet/util/rexx. The name of the file (always the most recent version) is 'ARexxAppList.lzh'.

Those with mail-only access to Internet may get a copy through ftp-mail. To find out how to do that, send email to 'ftpmail@decwrl.dec.com' containing only the word 'HELP' in the message body.
~~~~~~~~~~~~~~~~~~~~~~~

A recent copy of the list is usually available in the Usenet newsgroup comp.sys.amiga.applications.

Next: GETTING STARTED | Prev: Hello world | Contents: Introduction

1.9 ARexxGuide | Introduction (3 of 5) | GETTING STARTED

By themselves, ARexx scripts are merely collections of text that are meaningless to the Amiga. Some computer languages sift through the text written by the programmer (called 'source code') and translate it into a form understandable by the computer just once. The source code is 'compiled' once by a program that reads the text and outputs an executable program. (A compiler is available for ARexx, but is not covered in this guide.)

ARexx, on the other hand, translates a script's source code each time it is executed. The script is interpreted by another program, one that understands the meaning of ARexx tokens , expressions , and clauses and is able to translate them into the computer language understood by the Amiga.

The interpreter program is RexxMast . It is located in the sys:System drawer on standard Workbench disks (for versions 2.0+ of the OS). The interpreter must be running before ARexx scripts can be run.

STORING AREXX FILES: ARexx looks for files specified by the RX command in a directory called 'REXX:'. Each user should decide where on the system ARexx files will be stored and then add to the user-startup the following line:

```
   assign REXX: <script location>
```

In its manuals, Commodore recommends assigning REXX: to the S: directory, which is where standard DOS scripts are located. Unfortunately, that could become confusing to those who use ARexx frequently. The REXX: directory can easily be filled with dozens of files. Many application programs (this guide among them) now include large collections of scripts (sometimes with unique file extensions) to be added to the REXX: directory. That can quickly become overwhelming if they are mixed up with the files in S:.

It strikes this writer as more useful to devote a new directory to REXX: and leave S: for DOS scripts. The following might, therefore, be a useful assign statement:

```
   assign REXX: sys:rexx
```

Next: WRITING | Prev: Why ARexx | Contents: Introduction

1.10 ARexxGuide | Introduction (4 of 5) | WHERE TO WRITE PROGRAMS

ARexx does not supply an ‘environment’ in which scripts must be written. Any text editor will do. The script must begin with a comment , and must follow the basic rules that are the subject of this guide. That’s it.

You don’t need to learn how to work the menus and requesters in a new program, since ARexx itself has none of those. Just start your trusted editor and type. It is even possible to use a word processor to write scripts, but care must be taken in such an environment. First of all, the script will have to be saved as ASCII text since most word processors add formatting codes to documents -- codes that would be confusing to ARexx.

Another danger of word processors is their treatment of text lines. Text editors usually allow a line to be as long as necessary -- extending past the edge of the window if necessary. Word-processors, in contrast, wrap lines to fit within a defined column width. The problem in ARexx programming is that a line is significant. A long line of code can be divided into several lines in ARexx, but each line must end with a special code (the comma continuation character) to indicate that condition.

Once the script is written, it can be saved under nearly any name. The standard convention, though, is to save ARexx scripts that are meant to be executed from the shell with a filename extension of ‘.rexx’. ARexx will run such a script using the RX command even if its name is entered without the extension.

ARexx scripts meant to be called as macros from an application program are usually given unique extensions determined by the developer of the application.

Next: RUNNING PROGRAMS | Prev: Startup | Contents: Introduction

1.11 ARexxGuide | Introduction (5 of 5) | RUNNING YOUR PROGRAMS

The ARexx distribution -- both the commercial version and that included with the operating system -- includes a utility program called RX that has one task in life: to launch ARexx scripts from the Amiga shell or from an icon in much the way that the EXECUTE command launches DOS scripts.

One of the example scripts included in the ARexxGuide archive is called ‘ARx_Cmpr.rexx’ To run that script, the following command can be typed in the shell:

```
   RX ARx_Cmpr
```

If the script is located either in the rexx: directory or in the current directory, ARexx will find it there, even if rexx: is not included in the system’s command search path. Because the file extension ‘.rexx’ is the default extension for scripts launched from the shell, it need not be included.

For frequently-used ARexx scripts, an entry in the alias list of the User-Startup file can be helpful. If the script is saved as ‘rexx:DoAlottaStuff.rexx’, then the following will make it more easily accessible from the shell:

```
   alias DoStuff "rx DoAlottaStuff []"
```

Once the alias is entered in the startup file, the shorter name can be used to launch the script.

Another, somewhat more complex way to start an ARexx script, is to treat it like a DOS script file. The DOS PROTECT command can be used to set what is called the ‘script bit’ for a file. When that bit is set, AmigaDOS recognizes the file as something that can be executed. If the file begins with an ARexx comment line, then AmigaDOS will launch it as an ARexx script.

This is the command to set the ‘script bit’:

```
   PROTECT rexx:ARx_Cmpr.rexx +s
```

Once that bit is set, the rexx: directory must be added to the command search path with a line similar to the following (which should, ideally, be included in the file ‘s:user-startup’):

```
   PATH rexx: add
```

With those two tasks accomplished, the script ‘arx_cmpr.rexx’ can be started by entering is full name (including the extension, unfortunately) on the shell.

The need to include the extension along with the file name means that this method doesn’t save typing over using the RX command. A quick fix that works for some files is to store them in the rexx: directory with the script bit set but without an extension. That won’t work in all cases, however. The ARx_Cmpr file is one of those cases. It is called as an external function by its full name (with extension) from within this guide, so the extension is significant. (The default ARexx extension for scripts called from AmigaGuide® is ‘.guide’. Since that seems confusing, the extension is not used here.)

Next: WHAT TO WRITE | Prev: WRITING | Contents: Introduction

1.12 ARexxGuide | Introduction (6 of 5) | WHAT TO WRITE

It is beyond the scope of this guide to tell the user what to write. Once one has learned how to write scripts in ARexx, the ‘what?’ question will answer itself. Most people who have mastered ARexx find that there are too many ideas about what to write and not enough time to write.

The information presented in this guide is meant to help users master the intricacies of ARexx, but it may be overwhelming to a new user. One approach that might be helpful is to look first at the Tutorial section and the Techniques section. The scripts and sample code presented there show instructions, assignments, functions, and commands used in context.

It is also useful to study ARexx scripts written by others. The ARexxGuide help system supplied with the archive serves as an interface between a script and the information in ARexxGuide. Many scripts are available on networks and public-domain disks and macro scripts are included with many programs. Open one of the scripts in Ed 2.0 or another editor using the help system, and press the help-system key defined for the editor. If the word or character under the cursor has significance to ARexx, the ARexxGuide node explaining it will be presented. Additional information about a clause can be retrieved using the help system's optional clause information window .

Next: Introduction | Prev: RUNNING PROGRAMS | Contents: Introduction

1.13 ARexxGuide | Foreword | About this guide (1 of 2) | REQUESTER VERSION

The two versions: Requesters and standard
~~~~~~
Two versions of ARexxGuide are distributed. The same information is included in the two versions, but it is presented in different ways. The version with 'RQ' in the archive name uses requesters to replace many of the links in the standard version. Instead of jumping to the glossary node for definitions of terms like STDERR , the RQ version will present a small requester with the same information. The requester includes buttons that can be pressed to move to a new ARexxGuide node for more information.

The requesters in the RQ versions are implemented with reqtools.library by Nico François which is a copyrighted, but freely-distributable program available on most networks and BBSes that support the Amiga. The ARexx interface to ReqTools, called rexxreqtools.library, was created by Rafael D'Halleweyn and is included in the ReqTools distribution archives. It is copyrighted by Mr. D'Halleweyn.

Both libraries are required for those using the RQ version of ARexxGuide. Because the current versions of Multiview do not open an ARexx port, the RQ version of the guide will work properly only with version 34 of the AmigaGuide® utility. The requesters will be displayed in version 39 (the Multiview version), but the buttons will not work properly.

The requesters in ARexxGuide-RQ were tested with release 2.2a of ReqTools which is library version 38.1210 of reqtools.library and version 37.71 of rexxreqtools.library. That release is recommended. Look for an archive named 'ReqTools22a_User.lha' in your favorite shareware source. Those with access to the Internet can find the archive in the 'util/libs' directory of Aminet. A minimum version number of 37.50 is required for rexxreqtools.

Hyperlinks that call a requester in the RQ version are identified with marks on either side of the link word. (Those using non-standard fonts with the Amiga may not see the marks as what they should be: small dots in the center of the line.)

If ARexxGuide is launched with the included ARexxGuide.rexx script, then the message port used for the requesters will be opened and closed automatically. If it isn't already available, the port will be established when the first requester link is chosen. It can be closed at any time by choosing the 'Close Requester Port' link on the main contents page of the RQ version or by issuing this instruction:

```
   address 'ARX_REQPORT' 'QUIT'
```
~~~~~~

Formatting codes for slanted, bold, and highlighted text that are interpreted only by Multiview are used in the standard version of ARexxGuide, but not in the RQ version.

Next: Navigation | Prev: Introduction | Contents: Introduction

1.14 ARexxGuide | Foreword | About this guide (2 of 2) | NAVIGATION

Navigating ARexxGuide
~~~~~~~~~~~~~~~~~~~~~
The three most important keys in ARexxGuide are the -Browse Forward- key, the -Contents- key, and the mouse button or return key that chooses link points.

-- Press -Help- for general information about AmigaGuide keys --

The browse keys let you move from one node to the next. The contents key will bring you back to the closest in a nested series of contents pages. Some of those pages feature an overall explanation of the subject; some of the contents pages consist of little more than links to other nodes in the document.

Pressing the -Contents- key repeatedly will bring you back to the main contents page of the document 'ARexxGuide.guide'.

Nodes are attached to one another in a way that makes some of them invisible when using only the -Browse- keys.

```
1. Node name 1
 | a. subnode
 | b. subnode
 |  | b1. sub-subnode
 |  +-b2. sub-subnode
 +-c. subnode
2. Node name 2
 | a. subnode
 | b. subnode
 | c. subnode
 | d. subnode
 +-e. subnode
3. Node name 3
```

Nodes listed in the same column are directly accessible with the -Browse- keys. Pressing -Browse Forward- from node 1 will move you to from there to node 3. This allows for a quick overview of the subject before the subnodes are investigated for in-depth information.

One must press a link button to get from node 1 to node 1a, but once in node 1a the -Browse Forward- key will move you to node 1b and through the other subnodes of that level.

The lines extending from node 1c back up to node 1 indicate that the contents page for all of the subnodes of 1 is node 1 itself. Pressing -Browse Forward- from node 1c. will also move you back to node 1.
~~~~~~~~~~~~~~~~~~~~~

A line at the bottom of each node lists names of the nodes that will be shown when pressing the -Browse- keys or the -Contents- key. The titlebar identifies the section in which the current node is located and the number of nodes in the current level.

A NOTE ABOUT NODE-SURFING: Using a hypertext document is more like reading a newspaper than it is like reading a book. Just as it's possible to get lost in a newspaper when one has flipped back the section covers to follow a story's "jump" to the inside pages, so too, it's easy to get lost when following the links in a hypertext document.

The best thing to do with a newspaper is to fold back the sections to their front pages, rearranging the paper in its original form. So too, it's often best to move back to the beginning when jumping through links in a hypertext document. The -Contents- key allows you to rearrange each section so that its front page is on the front.

Some versions of AmigaGuide include a 'Bookmark' feature in the 'Navigation' menu. Setting the bookmark will let you quickly jump back to your starting point after investigating a few of the links.

When using the 'AmigaGuide' utility (but not with Multiview), a new window can be opened for any link by shift-selecting the link point. (Press Shift and Enter to select the current hyperlink, or Shift and the left mouse button to select any hyperlink.

Any open window (including the main one) can be closed immediately by pressing the Esc key.

Next: INTRODUCTION | Prev: Requester version | Contents: Introduction