ARx_Intro.ag

COLLABORATORS

	TITLE : ARx_Intro.ag	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		June 16, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ARx_Intro.ag

1.1 ARexxGuide | Introduction |

```
                  AN AMIGAGUIDE® TO ARexx                  Edition: 1.0
by Robin Evans

Preface

              Acknowledgements

              References

              About the author
                About this guide

              Navigating hints
                Intro to ARexx

              Hello World!

              Why ARexx?

              Getting it started

              Writing programs

              Running a script
                                    Copyright © 1993, Robin Evans.

 This guide is  shareware . If you find it useful, please register.
```

1.2 ARexxGuide | ACKNOWLEDGEMENTS (1 of 1)

```
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, and to those who provide fascinating discussions on
```

Usenet's comp.lang.rexx of the many flavors of REXX.

The development of this guide was aided greatly by Ed Patterson who had the patience to look through rough and early versions and provided invaluable suggestions. Thanks to Ron Adam for suggesting the header/footer method used to identify nodes. Thanks to Larry Shanahan for extensive link-testing and for raising several Multiview-compatibility issues.

Next, Prev, & Contents: Introduction contents

1.3 ARexxGuide | REFERENCES (1 of 1)

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
Prentice-Hall, First edition: 1985

Using ARexx on the Amiga
by Chris Zamara & Nick Sullivan
Abacus, 1992

The REXX Handbook
edited by Gabriel Goldberg & Philip H. Smith III
McGraw-Hill, 1992

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

Next, Prev, & Contents: Introduction contents

1.4 ARexxGuide | Author (1 of 1 [fortunately])

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 explanation of the meaning of "epistomology" or "ontology."

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. He teaches a class in REXX/ARexx at CALC, the online college available on GEnie.

Next, Prev & Contents: INTRODUCTION

1.5 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 which 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

1.6 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 through its facilities which can be used to control other programs.

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 of the 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 that does know about iff and anims.

Many PD and shareware programs provide ARexx support as well. EdPlayer, for example, is a wonderful MED/MOD player. It 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 can become a complete text engine, serving as a customized front-end for any communications service.

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

1.7 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.

That 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, but 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

1.8 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 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

1.9 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 which has one task in life: to launch ARexx scripts from the Amiga shell or from an icon in much the way that the command EXECUTE launches DOS scripts.

One of the example scripts that was installed in the rexx: directory by this guide's installation script 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 in the rexx: 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 ARexx standard, it need not be included.

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 may 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 merely entering is full name (including the extension) 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, which is possible for some files, is to store them in the rexx: directory without an extension.

That won't work in all cases, however. The arx_cmpr file is one of those

cases. It is called by its full name from within this guide as an external function, so the extension is significant.

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

1.10 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. Even if the script(s) presented there are not useful, it can be helpful to see instructions, assignments, and functions used in context.

It is also useful to study ARexx scripts written by others. Many scripts are now available on networks and public-domain disks.

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

1.11 ARexxGuide | Introduction (of) | Navigation

The three most important keys in this guide 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 this guide 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 node 2 and from there to node 3.

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 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 NOTE ABOUT LINK-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.

A new window can be opened for any link by shift-selecting the link point. Links identified as "NOTE:" on the link button will always open a new window.

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