EasyRexx Guide

Ketil Hunn

COLLABORATORS

	TITLE : EasyRexx Guide	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Ketil Hunn	July 22, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

EasyRexx Guide

1.1 Documentation for EasyRexx

easyrexx.library

1.2 Copyrights and Licence

DISCLAIMER

The files are provided "AS-IS" and with no warranties. Use at your own risk.

DISTRIBUTION

The files may be distributed as needed. That means that for products that use the easyrexx.library, only that file needs to be distributed. For development purposes, the library and its documentation should be all distributed together.

LICENSE

The license is the same for all software, regardless of what type of software the library is used in, be they commercial, freeware, shareware or whatever as long as you:

1) Note in the program and documentation that easyrexx.library is copyrighted © 1994,1995 Ketil Hunn.

2) You give me a copy (address below) of the software it is used in which
includes different versions of the software that use the library.
There should be no cost to me.

1.3 Description

DESCRIPTION
easyrexx.library is a small and very fast shared run-time library that
lets application developers add an AREXX port to your application with no
fuzz at all. A whole new world will open up, just by calling 4 functions
in this library! :D
When a message arrives at the AREXX port, easyrexx.library will parse the
line for you and stuff all arguments in an array. Interpreting EasyRexx
AREXX messages will now be as easy as reading arguments from a DOS prompt!
Most functions are tagbased which makes them very easy to use and fully
extendable for future features. The example source is pretty much self-
explanatory so you should take a look it and see how easy it is to use the
library.
All functions in the library are thoroughly described in the EasyRexx
autodoc.

1.4 Programming using easyrexx.library

This is how you use easyrexx.library to add AREXX to your application
(description given in C only):
1) #include <libraries/easyrexx.h>
2) Define unique IDs for each AREXX command your application will handle.
ex:
#define AREXX_QUIT 1
#define AREXX_OPEN 2
#define AREXX_SAVE 3
3) Create a table of commands that your application will handle. The
table contains:
{
LONG id // ID that will be returned (i.e. AREXX_QUIT).
UBYTE command, // Name of the command. Will be parsed quicker if
the name is given in upper-case (i.e. "QUIT").
template // Argument template. Standard DOS way of defining
how the arguments should be parsed. (i.e.

"FORCE/S". More about this later.
APTR userdata // You can put anything you want here. Will not be
touched by any functions in the library.
Normally you could put the address of a function
to be associated with a certain AREXX command.
Test2 shows how to use this method.
}
ex:
struct ARexxCommandTable table[]=
{
AREXX_QUIT, "QUIT", "FORCE/S", NULL,
AREXX_OPEN, "OPEN", "FILENAME/K,FORCE/S", NULL,
AREXX_SAVE, "SAVE", NULL, NULL,
TABLE_END,
};
4) Create your function to handle incoming AREXX events.
ex:
void myHandleAREXX(void)
{
if(GetARexxMsg(context))
{
switch(context->id)
{
case AREXX_QUIT:
quit=TRUE;
break;
case AREXX_OPEN:
OpenProject(ARGSTRING(context, 0), ARGBOOL(context, 1));
break;
case AREXX_SAVE:
SaveProject();
break;
}
ReplyARexxMsg(context, TAG_DONE);
}
}
5) Make your event-handler handle incoming AREXX signals.
BEFORE:
signal=Wait(1L<<myport->mp_SigBit);

if(signal & 1L<<myport->mp_SigBit)
handlestuff();
AFTER:
signal=Wait(1L<<myport->mp_SigBit | ER_SIGNAL(context));
if(signal & ER_SIGNAL(context))
myHandleAREXX(context);
else if(signal & 1L<<myport->mp_SigBit)
handlestuff();
6) Allocate and free the structure needed by the easyrexx.library.
struct ARexxContext *context;
context=AllocARexxContext(ER_Portname, "MYAPP",
ER_CommandTable, table,
TAG_DONE);
/* input handler */
FreeARexxContext(context);
7) That’s it and that’s that :D
See the easyrexx.doc (autodoc) for further details about defined tags and
macros.
Your application will now be able to understand three AREXX commands:
OPEN, SAVE, QUIT.
When easyrexx.library receives an AREXX command that matches a command in
the command table, it will parse the arguments according to the template.
If the arguments do not match or are too few it will reply the message
with an error and your application will never hear the message.
If success, your application will receive the command with all arguments
stuffed in an array with the first argument in 0, second in 1 etc.
QUIT takes one argument FORCE so both ’QUIT’ and ’QUIT FORCE’ will be
accepted. You can find out if the FORCE argument was given using the
macro ARG(context, 0) which is defined in <libraries/easyrexx.h>.
OPEN takes two arguments: FILENAME and FORCE. The FILENAME argument needs
the keyword FILENAME to be present.
Accepted:
OPEN FILENAME "ram:foo" FORCE
OPEN FILENAME "ram:foo"
Rejected:
OPEN "ram:foo" FORCE
OPEN "ram:foo"
SAVE takes no arguments.
Other useful macros to get the arguments:

ARGSTRING(context, i) - returns the string in argument i.
ARGNUMBER(context, i) - returns the number in argument i.
ARG(context, i) - returns 0 if the argument is NULL and non-zero if it present.
ARGBOOL(context,i) - returns TRUE if the argument is not empty and FALSE if it is.

Defining templates

Options in the template are separated by commas. To get the results of EasyRexx message, you examine the array of longwords you passed to it (one entry per option in the template). Exactly what is put in a given entry depends on the type of option. The default is a string (a sequence of non-whitespace characters, or delimited by quotes, which will be stripped, in which case the entry will be a pointer.

Options can be followed by modifiers, which specify things such as the type of the option. Modifiers are specified by following the option with a / and a single character modifier. Multiple modifiers can be specified by using multiple /s. Valid modifiers are:

/S - Switch. This is considered a boolean variable, and will be set if the option name appears in the command-line. The entry is the boolean (0 for not set, non-zero for set).

/K - Keyword. This means that the option will not be filled unless the keyword appears. For example if the template is "Name/K", then unless "Name=<string>" or "Name <string>" appears in the command line, Name will not be filled.

/N - Number. This parameter is considered a decimal number. If an invalid number is specified, an error will be returned. The entry will be a pointer to the longword number (this is how you know if a number was specified).

/T - Toggle. This is similar to a switch, but when specified causes the boolean value to "toggle". Similar to /S.

/A - Required. This keyword must be given a value during command-line processing, or an error is returned.

/F - Rest of line. If this is specified, the entire rest of the line is taken as the parameter for the option, even if other option keywords appear in it.

/M - Multiple strings. This means the argument will take any number of strings, returning them as an array of strings. Any arguments not considered to be part of another option will be added to this option. Only one /M should be specified in a template. Example for

a template "Dir/M,All/S" the command-line "foo bar all qwe" will set
the boolean "all", and return an array consisting of "foo", "bar",
and "qwe". The entry in the array will be a pointer to an array of
string pointers, the last of which will be NULL.
There is an interaction between /M parameters and /A parameters. If there
are unfilled /A parameters after parsing, it will grab strings from the
end of a previous /M parameter list to fill the /As. This is used for
things like Copy ("From/A/M,To/A").

1.5 History

V1.77 29.Nov.94
Initial release
V1.78 30.Nov.94
NEW Small linkable library included that automates calling context->
userdata functions. Included another testprogram with source that
shows this function handling.
V1.105 10.Jan.95
FIX SendARexxCommand vararg prototype was not named properly.
FIX ReplyARexxMsg did not handle returncodes properly.
FIX Some mistakes in the documentation corrected.
NEW ReplyARexxMsg can now return strings to the calling AREXX-script.
NEW ReplyARexxMsg can now return values to the calling AREXX-script.
NEW More tag aliases.
NEW 'Small linkable library'-source included to show how to automaticly
call functions.

1.6 Author

Send your contribution, suggestions and bug-reports to this address:
Ketil Hunn
Apartment 107C
Fabrikkveien 4-8
N-6400 Molde
NORWAY
e-mail:
Ketil.Hunn@hiMolde.no
After May 1995:
Ketil Hunn

Nabbetorpveien 35B

N-1632 Gamle Fredrikstad

NORWAY

Have fun :)

Ketil Hunn

January, 1995