

extrdargs.doc

COLLABORATORS

	TITLE : extrdargs.doc		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		July 22, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	extrdargs.doc	1
1.1	extrdargs.doc	1
1.2	ExtReadArgs/--background--	1
1.3	ExtReadArgs/ExtFreeArgs()	2
1.4	ExtReadArgs/ExtReadArgs()	2

Chapter 1

extrdargs.doc

1.1 extrdargs.doc

```
--background--      ExtFreeArgs()  
ExtReadArgs()
```

1.2 ExtReadArgs/--background--

PURPOSE

This is a CLI/Workbench transparent argument interface. I don't liked the way of parsing ToolTypes and used only the ReadArgs() function. Thus all my tools can only be invoked from the CLI/Shell . Thats the reason for building this project !

FUNCTION

ExtReadArgs() copies all Workbench arguments in a single string and passes this string to the ReadArgs() function. All WBArg structure are expanded to their full filenames , enclosed in '"' and passed to the item specified via the erda_FileParameter field. Then all Tooltypes are strcat()'ed into one line, thus ReadArgs() can handle it. To handle each Tooltype correctly the argument is enclosed in '"' !

NOTE

There are some special feature according to the ReadArgs() function. If you have a template like "FROM/M/A,TO/A", you can select these files from workbench : Select the program, then the FROM files and finally select and double click th TO file. This is available,because ReadArgs() grab's the last string from a MultiArg FROM and uses it as the TO parameter, if no is explicitly given !

INSPIRATION

I get the main idea, how I implement the Workbench ReadArgs() interface from the author of ARoach Stefan Winterstein. Thanks for this idea of parsing ToolTypes !

1.3 ExtReadArgs/ExtFreeArgs()

NAME

ExtFreeArgs - free's all allocated resources from ExtReadArgs()

SYNOPSIS

```
ExtFreeArgs(extrdargs);  
  
void ExtFreeArgs(struct ExtrDArgs *);
```

FUNCTION

free's all allocated resources from a previously call to ExtReadArgs().

INPUTS

extrdargs (struct ExtrDArgs *) - same pointer, which was passed to ExtReadArgs()

RESULTS

none

SEE ALSO

ExtReadArgs()

1.4 ExtReadArgs/ExtReadArgs()

NAME

ExtReadArgs - CLI/Workbench transparent ReadArgs() function

SYNOPSIS

```
error = ExtReadArgs(ac,av,extrdargs);  
  
LONG ExtReadArgs(LONG ,STRPTR *,struct ExtrDArgs *);
```

FUNCTION

this function is a CLI/Workbench transparent interface to ReadArgs(). It uses the argcount and argvector like SASC from the main entry point, to get the initial startup parameter. If ac is zero, so the program is invoked from workbench and the av variable contains the WBStartup structure ! Before you can call this function, you must set up the library bases for dos.library and icon.library. Normally the SASC autoinitialization code does this for you !
If all went right you get a return value of zero. This means the passed arguments fits the template and are ready to use. Otherwise you get a IoErr() like return code. You can pass this return value directly to PrintFault() or something like that !

NOTE : You must call the ExtFreeArgs() function to clean up, even this function fails (see EXAMPLE) !!!

INPUTS

ac (LONG) - parameter normally get from main()
av (STRPTR *) - parameter normally get from main()

extrdargs (struct ExtrDArgs *) - structure , which hold any information used by ExtReadArgs()

structure fields to setup before calling ExtReadArgs() :

erda_Template - the really ReadArgs() template
 erda_Parameter - ReadArgs() LONG WORD array to hold the arguments
 erda_FileParameter - number of Argument in the template to use for the files passed via WBStartup->sm_ArgList or -1, that means you don't want any files
 erda_Window - window description string to open, if the program is started from the workbench or NULL for no window ! If in the ToolType Array exists a WINDOW description this is used instead of the parameter of the ExtrDArgs structure !
 erda_RDArgs - RDArgs structure to use for ReadArgs() call, thus you can use extended help !
 erda_Buffer - pointer to a buffer to use for the Workbench startup or NULL, that means ExtReadArgs() allocates a buffer for you
 erda_BufferSize - if you provided a buffer, here is the length of it. If not this is the length you would have ! This length is checked against a minimum of ERDA_MIN_BUFFER_SIZE !

RESULTS

zero for success, otherwise an IoErr() like error code.

If the function successes you can check the erda_Flags field for the FRDAF_WORKBENCH flag, if you want to know from where the program was started

EXAMPLE

```
/* In this example the dos.library and icon.library must be open
 * from autoinitialization code
 */
LONG main(LONG ac, STRPTR *av)
{
    struct ExtrDArgs eargs = {NULL};
    LONG para[2];
    LONG error;

    eargs.erda_Template      = "FILES/M/A,VERBOSE";
    eargs.erda_Parameter     = para;
    eargs.erda_FileParameter = 0;
    eargs.erda_Window        = "CON:///My WB-Window/CLOSE/WAIT";

    if((error = ExtReadArgs(ac, av, &eargs)) == 0)
    {
        /* do something */
    } else
    {
        PrintFault(error, "MyProgram");
        ExtFreeArgs(&eargs);
    }

    return((error == 0) ? RETURN_OK : RETURN_FAIL);
}
```

SEE ALSO

```
ExtFreeArgs(), dos.library/ReadArgs(),  
icon.library/GetDiskObjectNew()
```