Sourcecode: Example2.c

COLLABORATORS

	TITLE : Sourcecode: Example2.c		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sourcecode: Example2.c

1.1 Example2.c

```
/***********************************************************/
/*                                                         */
/* Amiga C Encyclopedia (ACE)           Amiga C Club (ACC) */
/* --------------------------           ------------------ */
/*                                                         */
/* Manual:  AmigaDOS                    Amiga C Club       */
/* Chapter: Parsing Command Line        Tulevagen 22       */
/* File:    Example2.c                  181 41  LIDINGO    */
/* Author:  Anders Bjerin               SWEDEN             */
/* Date:    93-03-06                                       */
/* Version: 1.0                                            */
/*                                                         */
/*   Copyright 1993, Anders Bjerin - Amiga C Club (ACC)    */
/*                                                         */
/* Registered members may use this program freely in their */
/*     own commercial/noncommercial programs/articles.     */
/*                                                         */
/***********************************************************/

/* This example demonstrates how to parse the command line    */
/* with several arguments. This example handles two types of  */
/* command templates. First it can collect one or more        */
/* words which will be used as file names. This demonstrates  */
/* the "/M" (Multiple argument) option. Secondly the example  */
/* accepts a special argument used as a switch. This          */
/* demonstrates the "/S" ("Switch") option. The special       */
/* argument is "Filter", but can also be abbreviated as "F".  */



/* Include the dos library definitions: */
#include <dos/dos.h>

/* Include information about the argument parsing routine: */
#include <dos/rdargs.h>

/* Now we include the necessary function prototype files:         */
#include <clib/dos_protos.h>       /* General dos functions...    */
```

```
#include <clib/exec_protos.h>      /* System functions...         */
#include <stdio.h>                 /* Std functions [printf()...] */
#include <stdlib.h>                /* Std functions [exit()...]   */



/* Here is our command line template. This program handles two     */
/* types of command templates:                                     */
/*                                                                 */
/* 1. "SoundFiles/A/M" The program accepts one or more arguments   */
/*                     which will be used as file names. The "/A"  */
/*                     option tells the ReadArgs() function that   */
/*                     at least one file name must be given. The   */
/*                     "/M" option tells the ReadArgs() function   */
/*                     that this template should accept several    */
/*                     arguments if necessary. (All arguments      */
/*                     which can not be placed anywhere else will  */
/*                     go here. Please note that only one "/M"     */
/*                     option can be used in the command line      */
/*                     template.)                                  */
/*                                                                 */
/* 2. "F=Filter/S"     The user has an option of adding the        */
/*                     argument "Filter". The "/S" option tells    */
/*                     the ReadArgs() function that this argument  */
/*                     should be treated as a switch. If the       */
/*                     argument is set the switch will be turned   */
/*                     "on", else it will be "off". The "F="       */
/*                     string means that the user also can use the */
/*                     abbreviation "F" in stead of writing the    */
/*                     whole argument "Filter".                    */
/*                                                                 */
/* (Note the comma [,] between the command templates and that      */
/* there are no spaces [ ].)                                       */
#define MY_COMMAND_LINE_TEMPLATE "SoundFiles/A/M,F=Filter/S"

/* Here are some valid command lines:                              */
/*   Example2 Bird.snd                                             */
/*   Example2 Bird.snd River.snd                                   */
/*   Example2 Bird.snd River.snd Sea.snd                           */
/*   Example2 Bird.snd Filter                                      */
/*   Example2 Bird.snd River.snd F                                 */
/*   Example2 Bird.snd Filter River.snd Sea.snd                    */
/*                                                                 */
/* Here are some incorrect command lines:                          */
/*   Example2                    One file name is required!        */
/*   Example2 Filter                       - " -                   */



/* Two command templates are used: */
#define NUMBER_COMMAND_TEMPLATES  2

/* The command template numbers: (Where the result of each */
/* command template can be found in the "arg_array".)      */
#define SOUNDFILES_TEMPLATE  0
#define FILTER_TEMPLATE      1
```

```
/* Set name and version number: */
UBYTE *version = "$VER: AmigaDOS/ParsingCommandLine/Example2 1.0";



/* Declare an external global library pointer to the Dos library: */
extern struct DosLibrary *DOSBase;



/* Declared our own function(s): */

/* Our main function: */
int main( int argc, char *argv[] );



/* Main function: */

int main( int argc, char *argv[] )
{
  /* Simple loop variable: */
  int loop;

  /* Store the pointer to the array of string pointers here: */
  UBYTE **string_array;

  /* Pointer to a RDArgs structure which will automatically */
  /* be created for us when we use the RDArgs() function:   */
  struct RDArgs *my_rdargs;

  /* The ReadArgs() function needs an arrya of LONGs where */
  /* the result of the command parsing will be placed. One */
  /* LONG variable is needed for every command template.   */
  LONG arg_array[ NUMBER_COMMAND_TEMPLATES ];

  /* Note! This "arg_array" must be cleared (all values set to */
  /* zero) before we may use it with the ReadArgs() function.  */
  /* If we declare this structure outside the main function    */
  /* all values will automatically be cleared by C, but if we, */
  /* as in this example, declare the array inside a function   */
  /* we have to clear it manually. (If we do not clear it we   */
  /* can not examine the array and see if a field is set or    */
  /* not.)                                                     */



  /* The built in command parsing routine was first  */
  /* introduced in Release 2. V36 of the dos library */
  /* was however rather "buggy", and you should only */
  /* use V37 or higher:                              */
  if( DOSBase->dl_lib.lib_Version < 37 )
  {
    /* Too old dos library! */
    printf( "This program needs Dos Library V37 or higher!\n" );
    /* Exit with an error code: */
    exit( 20 );
  }



  /* We will now clear the "arg_array" (set all values to zero): */
  for( loop = 0; loop < NUMBER_COMMAND_TEMPLATES; loop++ )
    arg_array[ loop ] = 0;



  /* Parse the command line: (ReadArgs() will read the command   */
  /* line and with the help of the command line template set     */
  /* the corresponding values in the "arg_array" which is used   */
  /* to store the result of the command parsing. The function    */
  /* will return a pointer to a RDArgs structure which has       */
  /* automatically been created for us, since we did not create  */
  /* one ourself. This structure must be removed with help of    */
  /* the FreeArgs() function before your program may terminate.) */
  my_rdargs =
    ReadArgs( MY_COMMAND_LINE_TEMPLATE,
              arg_array,
              NULL
            );

  /* Have AmigaDOS successfully parsed our command line? */
  if( !my_rdargs )
  {
    /* The command line could not be parsed! The user probably */
    /* forgot to enter an argument which is required.          */
    printf( "Could not parse the command line!\n" );

    /* Life isn't fair... */
    exit( 21 );
  }

  /* The comand line has successfully been parsed! */
  /* We can now examine the "arg_array":           */



  /* Print template 1, the file name argument. Since the user may */
  /* enter several file names (the "/M" option is set) the value  */
  /* in the "arg_array" will not be a pointer to a string.        */
  /* Instead, the value in the "arg_array" will be a pointer to   */
  /* another array of strings where the file names are stored.    */
  /* Please note that this will only happen if you have set the   */
  /* "/M" option.                                                 */

  /* Are there any file names (there must be at least one  */
  /* in this example, the "/A" option is se, but we better */
  /* check it anyway...)                                   */
  if( arg_array[ SOUNDFILES_TEMPLATE ] )
  {
    /* Store the pointer to the array of stirng pointers: */
    /* (I agree that double pointers look horrible...)    */
    string_array = (UBYTE **) arg_array[ SOUNDFILES_TEMPLATE ];

    /* What we have to do now is to examine all strings with help of */
    /* a simple while loop. The last string in the array will be set */
    /* to NULL so we know were the list ends.                        */

    /* Start with the first string: */
    loop = 0;

    /* Print all file names: */
    while( string_array[ loop ] )
    {
      /* Print the file name: */
      printf( "File name: %s\n", string_array[ loop ] );

      /* Increase the counter: */
      loop++;
    }

    /* All file names have now been printed! */
  }



  /* Print template 2, the filter switch. Since this is a switch  */
  /* argument it can either be on or off. If the user has entered */
  /* the argument "Filter" or the abbreviation "F" the second     */
  /* field in the "arg_array" will contain a non-zero number,     */
  /* else (the user has not entered the argument "Filter" or "F") */
  /* the second field in the "arg_array" is set to zero.          */

  /* Was the argument "Filter" or "F" set? */
  if( arg_array[ FILTER_TEMPLATE ] )
    printf( "The sound filter was turned on!\n" );
  else
    printf( "No sound filter will be used!\n" );



  /* Before our program terminates we have to free the RDArgs */
  /* structure which was automatically allocated for us:      */
  FreeArgs( my_rdargs );

  /* Please note that the arguments that was collected by the */
  /* ReadArgs() function will also be removed when you call   */
  /* FreeArgs. Any pointers in the "result_templates" array   */
  /* which pointed to some data, for example strings, may     */
  /* therefore not be used any more after you have called     */
  /* FreeArgs(). The data (strings) will have been            */
  /* deallocated.                                             */



  /* "And they lived happily ever after..." */
  exit( 0 );
```

```
}
```