Sourcecode: Example4.c

COLLABORATORS

	TITLE : Sourcecode: Example4.c		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sourcecode: Example4.c

1.1 Example4.c

```
/***********************************************************/
/*                                                         */
/* Amiga C Encyclopedia (ACE)           Amiga C Club (ACC) */
/* -------------------------            ------------------ */
/*                                                         */
/* Manual:  AmigaDOS                    Amiga C Club       */
/* Chapter: Parsing Command Line        Tulevagen 22       */
/* File:    Example4.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 you can create a RDArgs structure */
/* yourself and prepare it before you parse the command line with  */
/* the help of the ReadArgs() function. Since we can prepare the   */
/* RDArgs structure we can include extra help (will be displayed   */
/* if the user types "?" to display the command line template and  */
/* then types "?" again.), decide if the user should be able to    */
/* see the command line template, etc...                           */



/* 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 three      */
/* types of command templates:                                        */
/*                                                                    */
/* 1. "SoundFile/A" The ReadArgs() expects one file name, else the    */
/*                  function will fail. Since there is no "/M"        */
/*                  option only one file name may be given.           */
/*                                                                    */
/* 2. V=Volume/K/N" The second type of argument is optional (no "/A"  */
/*                  option. It must be a number ("/N" - Number option */
/*                  is set) and preceded by the keyword "Volume" or   */
/*                  "V" ("/K" - Keyword required). If a keyword is    */
/*                  needed the user can either write the keyword a    */
/*                  space and then the number, or the user may write  */
/*                  the keyword an equal sign (=) and then the        */
/*                  number. Please note that the "V=Volume" only      */
/*                  means that the user can write "V" instead of the  */
/*                  longer keyword "Volume", and this equal sign has  */
/*                  nothing to do with the optional equal sign the    */
/*                  user may write after the keyword and before the   */
/*                  number. (No decimal numbers, e.g. "4.57", "1.2",  */
/*                  are accepted.)                                    */
/*                                                                    */
/* 3. "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".                                */

#define MY_COMMAND_LINE_TEMPLATE "SoundFile/A,V=Volume/K/N,F=Filter/S"

/* Here are some valid command lines:                                 */
/*   Example4 Bird.snd                                                */
/*   Example4 Bird.snd Volume=64                                      */
/*   Example4 Bird.snd Volume 64                                      */
/*   Example4 Bird.snd Filter                                         */
/*   Example4 Bird.snd Volume=64 F                                    */
/*                                                                    */
/* Here are some incorrect command lines:                             */
/*   Example4                  The file name is required!             */
/*   Example4 Bird.snd 64      The keyword "Volume" or "V" must       */
/*                             precede the number 64.                 */
/*   Example4 Bird.snd V=5.25  Decimal values may not be used.        */



/* Three command templates are used: */
#define NUMBER_COMMAND_TEMPLATES 3

/* The command template numbers: (Where the result of each */
/* command template can be found in the "arg_array".)      */
#define SOUNDFILE_TEMPLATE  0
#define VOLUME_TEMPLATE     1
#define FILTER_TEMPLATE     2



/* Set name and version number: */
UBYTE *version = "$VER: AmigaDOS/ParsingCommandLine/Example4 1.0";



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



/* Declare a pointer to a RDArgs structure which we will allocate */
/* ourself with help of the AllocDosObject() function:            */
struct RDArgs *my_rdargs;



/* Declared our own functions: */

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

/* Cleans up nicely after us: */
void clean_up( STRPTR text, int code );



/* Main function: */

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

  /* A pointer to the volume value: */
  LONG *volume_value;

  /* Store the pointer which is returned by ReadArgs() here: */
  /* (ReadArgs() returns a pointer to a RDArgs structure if  */
  /* it could successfully parse the command line. Since we  */
  /* have created the RDArgs structure ourself before we     */
  /* call ReadArgs() it will simply return a pointer to the  */
  /* structure which we already have a pointer to. However,  */
  /* we need a separate variable to store the returned value */
  /* in since we need to check if ReadArgs() actually could  */
  /* parse the command line or not. If not NULL is returned. */
  struct RDArgs *temp_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 ];



  /* We need dos library version 37 or higher: */
  if( DOSBase->dl_lib.lib_Version < 37 )
    clean_up( "This program needs Dos Library V37 or higher!", 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;



  /* Get a RDArgs structure from AmigaDOS: (We want a RDArgs */
  /* structure with no special tags.)                       */
  my_rdargs = (struct RDArgs *) AllocDosObject( DOS_RDARGS, NULL );

  /* Did we get the RDArgs structure? */
  if( !my_rdargs )
    clean_up( "Could not creae the RDArgs structure!", 21 );



  /* If we set the "RDAF_NOPROMPT" flag in the "RDA_Flags" filed of */
  /* the RDArgs structure the user will not be allowed to see the   */
  /* command line template by typing a single question mark (?).    */
  /* If you set this flag the question mark will be accepted as a   */
  /* complete argument if written. Normally you should not turn of  */
  /* this help function! In this example I have therefore put the   */
  /* line inside comment marks. (If you take them away the user     */
  /* will not be be able to see the command line template nor the   */
  /* extra help line defined below.)                                */
  /*                                                                */
  /* my_rdargs->RDA_Flags = RDAF_NOPROMPT;                          */

  /* Set an extra help line: (The user can see this help line by    */
  /* typing a question mark (?), so he/she will see the command     */
  /* template line, and then type a question mark again.)           */
  my_rdargs->RDA_ExtHelp =  (UBYTE *) "This wasn't much help...";



  /* Parse the command line: (Note that we now use our */
  /* own RDArgs structure which we have prepared.)     */
  temp_rdargs =
    ReadArgs( MY_COMMAND_LINE_TEMPLATE,
              arg_array,
              my_rdargs
            );

  /* Have AmigaDOS successfully parsed our command line? */
  if( !temp_rdargs )
    clean_up( "Could not parse the command line!", 22 );
  /* The comand line has successfully been parsed! */
  /* We can now examine the "arg_array":           */

  /* Print template 1, the file name: */
  if( arg_array[ SOUNDFILE_TEMPLATE ] )
    printf( "File name: %s\n", arg_array[ SOUNDFILE_TEMPLATE ] );



  /* Print templat 2, the volume: */
  if( arg_array[ VOLUME_TEMPLATE ] )
  {
    /* Get a pointer to the volume value: */
    volume_value = (LONG *) arg_array[ VOLUME_TEMPLATE ];

    /* Print the volume: */
    printf( "Volume: %ld\n", *volume_value );
  }
  else
    printf( "No volume was set\n" );



  /* Print template 2, the filter switch: */
  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 data that */
  /* have been allocated when we successfully called ReadArgs(): */
  FreeArgs( my_rdargs );

  /* The RDArgs structure we allocated will be */
  /* deallocated in the clean_up() function.   */

  /* Clean up and exit with a smile on your face! */
  clean_up( "The End", 0 );
}



/* Handy function which closes and deallocates everything */
/* that you have previously opened or allocated. You can  */
/* call this function at any time, and it will clean up   */
/* nicely after you and quit.                             */

void clean_up( STRPTR text, int code )
{
  /* Return the RDArgs structure to AmigaDOS: */
  if( my_rdargs )
    FreeDosObject( DOS_RDARGS, my_rdargs );
  /* Print the last message: */
  printf( "%s\n", text );

  /* Quit: */
  exit( code );
}
```