Sourcecode: Example3.c

COLLABORATORS

	TITLE : Sourcecode: Example3.c		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sourcecode: Example3.c

1.1 Example3.c

```
/***********************************************************/
/*                                                         */
/* Amiga C Encyclopedia (ACE)           Amiga C Club (ACC) */
/* --------------------------           ------------------ */
/*                                                         */
/* Manual:  AmigaDOS                    Amiga C Club       */
/* Chapter: Parsing Command Line        Tulevagen 22       */
/* File:    Example3.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 a command line with */
/* both a string and an optional value argument which require */
/* a keyword. (Demonstrates the "/N" and "/K" options.)       */



/* 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. "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.)                                     */

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

/* Here are some valid command lines.                                  */
/*   Example3 Bird.snd                                                 */
/*   Example3 Bird.snd Volume=64                                       */
/*   Example3 Bird.snd Volume 64                                       */
/*                                                                     */
/* Here are some incorrect command lines:                              */
/*   Example3                  The file name is required!              */
/*   Example3 Bird.snd 64      The keyword "Volume" or "V" must        */
/*                             precede the number 64.                  */
/*   Example3 Bird.snd V=5.25  Decimal values may not be used.         */



/* 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 SOUNDFILE_TEMPLATE  0
#define VOLUME_TEMPLATE     1



/* Set name and version number: */
UBYTE *version = "$VER: AmigaDOS/ParsingCommandLine/Example3 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;

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

  /* 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.)                                                     */



  /* We need dos library version 37 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: */
  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" );

    /* Better luck next time... */
    exit( 21 );
  }



  /* 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. The volume was an optional */
  /* argument, and we are not sure if the user has given us  */
  /* a volume number or not. We must therefore check the     */
  /* "arg_array" and see if the second field contains a      */
  /* pointer to the volume number, or NULL (no volume is     */
  /* set).                                                   */
  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" );



  /* Before our program terminates we have to free the data that */
  /* have been allocated when we successfully called ReadArgs(): */
  FreeArgs( my_rdargs );

  /* Please note that any pointers in the "arg_array" which   */
  /* pointed to some data, for example strings, may not be    */
  /* used any more after you have called FreeArgs(). The data */
  /* (strings etc...) have now been deallocated, and can not  */
  /* be accessed any more.                                    */



  /* "The show must go on..." */
  exit( 0 );
}
```