Sourcecode: Example5.c

COLLABORATORS			
	TITLE : Sourcecode: Example5.c		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sourcecode: Example5.c

1.1 Example5.c

```
/*********************************************************/
/*                                                       */
/* Amiga C Encyclopedia (ACE)           Amiga C Club (ACC) */
/* --------------------------           ------------------ */
/*                                                       */
/* Manual:  AmigaDOS                    Amiga C Club       */
/* Chapter: Parsing Command Line        Tulevagen 22       */
/* File:    Example5.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 your own strings  */
/* (command lines) which you then can parse with help of the      */
/* ReadArgs() function. We create a RDArgs structure as in the    */
/* last example, but this time we initialize the "RDA_Source"     */
/* field with our own command line. When we later call ReadArgs() */
/* it will notice that it already have a string to parse, and it  */
/* will therefore use that string and not read one from the       */
/* default input handler.                                         */



/* 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()...]   */
#include <string.h>                /* Std functions [srtlen()...] */



/* Here is our command line template: */
#define MY_COMMAND_LINE_TEMPLATE "SoundFile/A,V=Volume/K/N,F=Filter/S"

/* 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/Example5 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: */
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 ];

/* Here is our own command line we want to parse: */
/* Note the new line character ("\n") at the end  */
/* of the string. You must always include this    */
/* at the end of the strings you want to parse.   */
UBYTE *my_command_line = "Bird.snd Volume=35 Filter\n";



/* 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 a RDArgs structure: */
if( !my_rdargs )
  clean_up( "Could not get a RDArgs structure!", 21 );



/* Prepare the RDArgs structure so it uses our own command line: */

/* Give the RDArgs structure our own command line: (The command */
/* line will be fetched from the CSource structure if it is not */
/* empty. Normally the command line is fetched from the default */
/* input stream which was set up when the program started, but  */
/* you may want to parse some other string rather than the one  */
/* which was written when the user launched this program).      */
my_rdargs->RDA_Source.CS_Buffer = my_command_line;

/* Set the length of the command line: */
my_rdargs->RDA_Source.CS_Length = strlen( my_command_line );

/* Set the current character position so it starts to read */
/* the first character in the string (character 0):        */
my_rdargs->RDA_Source.CS_CurChr = 0;
```

```
  /* Parse the command line: (Note that we now use our */
  /* own RDArgs structure which we have prepared with  */
  /* our own customized command line.)                 */
  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 );
}
```