Sourcecode: Example1.c

COLLABORATORS

	TITLE : Sourcecode: Example1.c	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sourcecode: Example1.c

1.1 Example1.c

```
/*********************************************************/
/*                                                       */
/* Amiga C Encyclopedia (ACE)           Amiga C Club (ACC) */
/* --------------------------           ------------------ */
/*                                                       */
/* Manual:  AmigaDOS                    Amiga C Club       */
/* Chapter: Parsing Command Line        Tulevagen 22       */
/* File:    Example1.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.  */
/* Since this is the first example it is relative simple.    */
/* The program expects one argument. If no argument is given */
/* will the parse function fail (uses the "/A" - "Always     */
/* required" option), and if more than one argument is give  */
/* it will also fail (the "/M" - "Multiple argument" option  */
/* is not set).                                              */



/* 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 simple command line template. This program expects */
/* one argument. If no argument is given will the parse function  */
/* fail since we have set the option "/A" ("Always required"),    */
/* and if more than one argument is given it will also fail since */
/* we have not set the "/M" ("Multiple argument") option.         */
#define MY_COMMAND_LINE_TEMPLATE "SoundFiles/A"

/* Here is a valid command line:                                  */
/*   Example1 Bird.snd                                            */
/*                                                                */
/* Here are some incorrect command lines:                         */
/*   Example1                    The file name is required!     */
/*   Example1 Bird.snd River.snd Only one argument may be used! */



/* Only one command template is used: */
#define NUMBER_COMMAND_TEMPLATES 1

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



/* Set name and version number: */
UBYTE *version = "$VER: AmigaDOS/ParsingCommandLine/Example1 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;

  /* 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" );

    /* See you later... */
    exit( 21 );
  }



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

  /* Print template 1, the file name, which is in this example the */
  /* only argument. Since the user must enter this argument, or    */
  /* else the ReadArgs() function would fail, we actually do not   */
  /* have to check that there is something in the array. However,  */
  /* it is easy to remove the "/A" option and forget to add a      */
  /* check later on, so we better always check that there is       */
  /* something in the array before we use it (you can't be too     */
  /* careful).                                                     */
  /*                                                               */
  /* (If the user would not have entered an argument and the "/A"  */
  /* option was not used, the LONG variable in the array would be  */
  /* NULL. If the user has entered an argument, which he must have */
  /* done in this example since it is required, the LONG variable  */
  /* in the arraw contains a pointer to a string with the argument */
  /* inside.)                                                      */
  if( arg_array[ SOUNDFILE_TEMPLATE ] )
    printf( "File name: %s\n", arg_array[ SOUNDFILE_TEMPLATE ] );



  /* 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.                                    */



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