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: Files                       Tulevagen 22       */
/* File:    Example1.c                  181 41  LIDINGO    */
/* Author:  Anders Bjerin               SWEDEN             */
/* Date:    93-03-11                                     */
/* Version: 1.1                                          */
/*                                                       */
/*   Copyright 1993, Anders Bjerin - Amiga C Club (ACC)  */
/*                                                       */
/* Registered members may use this program freely in their */
/*     own commercial/noncommercial programs/articles.   */
/*                                                       */
/*********************************************************/

/* This program collects ten integer values from the user, and  */
/* saves them in a file called "HighScore.dat" on the RAM disk. */



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

/* Now we include the necessary function prototype files:         */
#include <clib/dos_protos.h>       /* General dos functions...    */
#include <stdio.h>                 /* Std functions [printf()...] */
#include <stdlib.h>                /* Std functions [exit()...]   */



/* Set name and version number: */
UBYTE *version = "$VER: AmigaDOS/InputOutput/Example1 1.1";
```

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

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



/* Main function: */

int main( int argc, char *argv[] )
{
  /* A "BCPL" pointer to our file: */
  BPTR my_file;

  /* The numbers: (10 integers will be saved) */
  int my_highscore[ 10 ];

  /* Store here the number of bytes actually written: */
  long bytes_written;

  /* A simple loop variable: */
  int loop;



  /* Let the user enter ten integer values: */
  printf( "Please enter ten integer values:\n" );
  for( loop=0; loop < 10; loop++ )
  {
    printf( "Value [%d]: ", loop );
    scanf( "%d", &my_highscore[ loop ] );
  }



  /* Try to open file "RAM:HighScore.dat" as a new file: (If */
  /* the file does not exist, it will be created. If it, on  */
  /* the the other hand, exist, it will be overwritten.)     */
  my_file = Open( "RAM:HighScore.dat", MODE_NEWFILE );

  /* Have we opened the file successfully? */
  if( !my_file )
  {
    /* Inform the user: */
    printf( "Error! Could not open the file!\n" );

    /* Exit with an error code: */
    exit( 20 );
  }

  /* The file has now been opened: */
  printf( "File open!\n" );



  /* We have now opened a file and the file cursor is  */
```

```
  /* pointing to the first byte (character) in our new */
  /* file. We can now start to write:                  */
  bytes_written = Write( my_file, my_highscore, sizeof( my_highscore ) );

  /* Did we write all data? */
  if( bytes_written != sizeof( my_highscore ) )
  {
    /* No! The numbers actually written was less */
    /* than we wanted to write!                  */
    printf( "Error! Could not save all values!\n" );
  }
  else
  {
    /* Yes, all numbers have been written to the file! */
    printf( "All values were saved successfully!\n" );
  }

  /* Since we store 10 integer values the file should be 40 bytes  */
  /* long. 1 integer (32 bits) = 4 bytes, 10 integers (320 bits) = */
  /* 40 bytes.                                                     */



  /* Close the file. With V36 or higher the Close() function  */
  /* will return a boolean value, TRUE if the file was        */
  /* successfully closed, FALS if the file could not be       */
  /* closed. If the file could not be closed there is sadly   */
  /* very little we can do about it. We should never try to   */
  /* close a file after it has been closed, successfully or   */
  /* not! Even if the actual file could not be closed most of */
  /* the memory used by the filehandler will still have been  */
  /* deallocated.                                             */
  /*                                                          */
  /* In most cases you can simply ignore what the Close()     */
  /* function returns since you can not do much about it.     */
  /* However, if you have saved important data in the file    */
  /* you might want to open a new file and save it all again  */
  /* just to be on the safe side. Before you may do this you  */
  /* should of course ask for the user's permission.          */
  if( Close( my_file ) )
    printf( "File closed!\n" );
  else
    printf( "Error! File could not be closed!\n" );

  /* Remember that even if the file could not be */
  /* closed we must NOT try to close it again!   */



  /* The End! */
  exit( 0 );
}
```