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: Advanced Routines           Tulevagen 22       */
/* File:    Example4.c                  181 41  LIDINGO    */
/* Author:  Anders Bjerin               SWEDEN             */
/* Date:    93-03-17                                       */
/* 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 will examina all objects in a directory/device.   */
/* The files will be listed, and if finds a directory it will     */
/* with help of a recursive function examine all objects in       */
/* that directory also and so on... Good example on how to use    */
/* the Examine() and ExNext() functions in a recursive program.   */
/*                                                                */
/* This example can be used with all versions of the dos library. */



/* Please note that this example is  */
/* for experienced programmers only! */



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

/* Include memory definitions: */
```

```
#include <exec/memory.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 [strcpy()...] */



/* The maximum numbers of characters that can be */
/* stored in the complete file name with path:   */
#define MAX_LENGTH 120

/* The number of characters we will indent the line */
/* when we go inside another directory:             */
#define INDENT_LENGTH 4

/* The highest accceptable indent value: */
#define MAX_INDENT  80



/* Set name and version number: */
UBYTE *version = "$VER: AmigaDOS/Advanced Routines/Example4 1.1";



/* Declared our own functions: */

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

/* Our recursive directory lister: */
int ExamineDirectory( STRPTR dir_name, int indent );

/* Adds a directory name to a path: */
void AddToPath
(
  STRPTR complete,
  STRPTR name,
  STRPTR path
);



/* Main function: */

int main( int argc, char *argv[] )
{
  /* Store result code here: */
  int problems;



  /* The program name and one argument must have been entered: */
  if( argc < 2  || argc > 2)
  {
    /* Wrong number of arguments! */
    printf( "Error! Wrong number of arguments!\n" );
    printf( "You must enter a directory or volume name.\n" );
    printf( "Example4 Name/A\n" ); /* Simple template */

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



  /* Examine the directory: */
  problems = ExamineDirectory( argv[ 1 ], 0 );

  /* Any problems? */
  if( problems )
    printf( "There were problems, error code: %d\n", problems );



  /* The End! */
  exit( problems );
}



/* This function will:                                              */
/*   1. allocate some momory for a file info block structure.     */
/*   2. lock the directory.                                         */
/*   3. Examine the directory, and check that it is a directory. */
/*   4. List all objects in this directory and return when done. */
/*   5. If there are any directory inside this directory we call */
/*      ourself, and we have a nice recursive function...        */

int ExamineDirectory( STRPTR dir_name, int indent )
{
  /* This string will be used to store the complete name and path in: */
  UBYTE total_name[ MAX_LENGTH ];

  /* Our indent string: */
  UBYTE indent_string[ MAX_INDENT ];

  /* Simple loop variable: */
  int loop;

  /* String pointer: */
  STRPTR str_ptr;

  /* Store result codes here: */
  int result_code;

  /* A BCPL pointer to a file lock: */
  BPTR my_lock;

  /* Declare a pointer to a FileInfoStructure: (This structure  */
  /* must be long word aligned - start on an even word address. */
  /* To do this we must therefore allocate the structure with   */
  /* help of AllocMem(). If you have the new dos 2.xx or higher */
  /* you are recommended to use the AllocDosObject() function.  */
  /* This will be explained in the next version of the ACE.)    */
  struct FileInfoBlock *my_file_info_block;



  /* Set the result code to OK: */
  result_code = 0;

  /* Allocate enough memory for a FileInfoBlock structure: */
  /* (Any type of memory, fast or chip, and clear it.)     */
  my_file_info_block = (struct FileInfoBlock *)
    AllocMem( sizeof( struct FileInfoBlock ), MEMF_ANY | MEMF_CLEAR );

  /* Check if we have allocated the memory successfully: */
  if( my_file_info_block == NULL )
  {
    /* Inform the user about the problem: */
    printf( "Not enough memory!\n" );

    /* Return with an error code: */
    return( 21 );
  };



  /* We will now try to lock the directory: (We will */
  /* only read data so we can use a shared lock.)    */
  my_lock = Lock( dir_name, ACCESS_READ );

  /* Colud we lock the file? */
  if( my_lock == NULL )
  {
    /* Inform the user about the problem: */
    printf( "Could not lock the directory \"%s\"\n", dir_name );

    /* Deallocate the memory we have allocated: */
    FreeMem( my_file_info_block, sizeof( struct FileInfoBlock ) );

    /* Return with an error code: */
    return( 22 );
  }



  /* Prepare the indent string: */

  /* If we have not indented the line too much we indent it: */
  if( indent < MAX_INDENT )
  {
    /* Fill the indent string with spaces: */
    for( loop = 0; loop < MAX_INDENT; loop++ )
      indent_string[ loop ] = ' ';
    /* Set the stop (NULL) sign: (The higher the indent value is */
    /* the further in the string we set the NULL sign.)          */
    str_ptr = indent_string + indent;

    /* Set the NULL sign: */
    *str_ptr = NULL;
  }



  /* Try to examine the directory: */
  if( Examine( my_lock, my_file_info_block ) )
  {
    /* Check if it is really a directory: */
    if( my_file_info_block->fib_DirEntryType > 0 )
    {
      /* Yes, this is a directory! */

      /* Examine all objects in this directory: */

      /* As long as we find objects in this directory we continue: */
      while( ExNext( my_lock, my_file_info_block ) )
      {
        /* If we find a file we print out the name, and if we */
        /* find a directory we cal our self (recursive):      */
        if( my_file_info_block->fib_DirEntryType < 0 )
        {
          /* It is a file! */

          /* Print the file name: */
          printf( "%s%s\n",
            indent_string, my_file_info_block->fib_FileName );
        }
        else
        {
          /* It is a directory, and should therefore call our self! */

          /* Print the directory name: */
          printf( "%s%s (Directory)\n",
            indent_string, my_file_info_block->fib_FileName );

          /* However, first we must add the directory name to the */
          /* current path (the "fib_FileName" field only contains */
          /* the file name of the directory, not the path):       */

          AddToPath( total_name,                        /* Name & Path */
                     my_file_info_block->fib_FileName, /* Name        */
                     dir_name                           /* Path        */
                   );

          result_code =
            ExamineDirectory( total_name, indent + INDENT_LENGTH );
        }
      }
      /* We have now left the while loop. It was either because there  */
      /* were no more objects in the directory, or there was an error: */
      /* (If the error message isn't "ERRROR_NO_MORE_ENTRIES" it was   */
      /* an error.)                                                     */
      if( IoErr() != ERROR_NO_MORE_ENTRIES )
      {
        /* Inform the user: */
        printf( "Error while reading directory \"%s\"!\n", dir_name );

        /* Set an error code: */
        result_code = 23;
      }
    }
    else
    {
      /* (This can only happen the first time this function */
      /* is called, since we will only call ourself if we   */
      /* know it is a directory.)                           */

      /* The user tried to examine a file! */
      printf( "\"%s\" is a file!\n", dir_name );

      /* No directory specified! */
      printf( "You must enter directory name!\n" );

      /* Give the user a command line template: */
      printf( "RecursiveExamine DIRECTORY/A\n" );

      /* Set an error code: */
      result_code = 24;
    }
  }
  else
  {
    /* We could no examine the object: */
    printf( "Could not examine %s!\n", dir_name );

    /* Set an error code: */
    result_code = 25;
  }



  /* Unlock the file: */
  UnLock( my_lock );

  /* Deallocate the memory we have allocated: */
  FreeMem( my_file_info_block, sizeof( struct FileInfoBlock ) );

  return( result_code );
}



/* This function will copy the path to the complete string, and */
/* then add the directory name together with a "/" sign if      */
/* necessary.                                                   */
```

```
void AddToPath
(
  STRPTR complete,
  STRPTR name,
  STRPTR path
)
{
  /* A temporary string pointer: */
  STRPTR string_pointer;



  /* Put a stop character at the beginning of the complete string: */
  complete[ 0 ] = NULL;



  /* Move to the last character in the string: (Isn't C nice?) */
  string_pointer = (STRPTR) path + strlen( path ) - 1;

  /* Check what the right most character in the path string is: */
  if
  (
    *string_pointer == ':' ||
    *string_pointer == '/' ||
    *string_pointer == '\0'
  )
  {
    /* We can simply add the directory to the path string:   */
    /* (Just check that there is enough room before we add   */
    /* the directory.)                                       */
    if( strlen( path ) + strlen( name ) < MAX_LENGTH )
    {
      /* Copy the path to the complete string: */
      strcpy( complete, path );

      /* Add the directory name to the path: */
      strcat( complete, name );
    }
  }
  else
  {
    /* We have to add the '/' sign before we add the directory name: */
    /* (Just check that there is enough room before we add them!)    */
    if( strlen( path ) + 1 + strlen( name ) < MAX_LENGTH )
    {
      /* Copy the path to the complete string: */
      strcpy( complete, path );

      /* Add the "/" sign: */
      strcat( complete, "/" );

      /* Add the directory name to the path: */
      strcat( complete, name );
    }
  }
}
```