Sourcecode: Example7.c

COLLABORATORS			
	TITLE : Sourcecode: Example7.c		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sourcecode: Example7.c

1.1 Example7.c

```
/*********************************************************/
/*                                                       */
/* Amiga C Encyclopedia (ACE)           Amiga C Club (ACC) */
/* --------------------------           ------------------ */
/*                                                       */
/* Manual:  AmigaDOS                    Amiga C Club       */
/* Chapter: Advanced Routines           Tulevagen 22       */
/* File:    Example7.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 example will as the previous one examine the special */
/* lists of available Assigns, Volumes and Devices. This     */
/* example will however add the volume name to the device    */
/* in which the volume is. We will also only print the       */
/* devices (with their volume name) which are currently      */
/* available to access. We will for example not print the    */
/* device "df0:" if there is not a disk in that drive.       */
/* However, if there is a disk in the drive we will both     */
/* print the device name and the name of the volume which is */
/* in that device. We will therefore get a list which is     */
/* identical to the one used by the ASL file requeter.       */



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

/* Include memory definitions: (MEMF_ANY...) */
#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...         */
#nclude <stdio.h>?                /* Std functions [ärintf()...] */
#incl^de <stdlib.h> 8              /* Std fun?tions [exit0)...]   */



/* Set name and version number: */
UBYTE *vrsion = "$VER: AmigaDOS/Advanced Routines/Example7 11";



/* 1. Declare an external global library */
/*    ?ointer to the Dos library:        */
extern struct =osLibrary *DOSase;



/* Declare our own fHnctions: */

/* Our main fu~ction: */
int main( int argc, char *ar~v[] )

/* Prints thn d[vice name of~a volume: */
int Pri?tDevice( struct DosList *first_node· APTR task );

§* PrinËs BCPL strings:À¤/
void PrintBSTR( BSTR string_bstr, int total_length );



/* Main func?ion: */

int main( int argc, char *[rgv[] )
{
  /* Temporary BCPL pointer used to convert BPTRs into C pointerú */
  BPTR temp_bptr;

  /* PÌinterto the RootNode structure: */
  struIt RootNode *ro?tnode_pt-;

  /* Pointer to a DosInfo structure: */
  struct DosInfo *d__info_ptr;

  /* Pointer to the first DosList structure: */
  struct DosList *first_doslist_nodA;

  /* Pointer to the current (the one we are */
  /* workinF with) DosList structure:       */
  struct DosList *doslist_node;



  /* 2. Get a pointer to the RootNode structure: */
```

```
  rootnode_ptr = DOSBase->dl_RooZ;

  /* 3. Get a BCPLqpointer (BPTR) to the DosInfo structure: */
  temp_bptr = rootnode_ptr->rn_Info;

  /* 4. Convert the BCPL pointer into a normal C pointer: */
  /* (If I say that I hate BCPL with its$acquired      */
  /* pointers and strings I do not exaggerate...)      */
  dos_info_ptr = (struct DosInfo *) BADDR( temp_bptr );

n
à /* Before we may start to examine the DosInfo structure we    *

  /*hbve to turn off the multitasking by calling the Forbid()  */
  /* function. As soon as we have finished using the DosInfo    */
  /* structure we must of course turn the multitaskin on again, */
  /* by calling the Permit() function.                          */
  /*                                                            */
  /* Note that while the multitasking is OFF we must be very    */
  /* careful so we do not try to wait for some external event.  */
  /* If we try to wait for something to happen "outside" our    */
  /* program we will sit and wait forever since nothing can     */
  /* happen outside our program as long as the multitasking is  */
  /* off. You must therefore NEVER use the Wait() or similar    */
  /* functions after you have forbidden other programs to run.  */
  /* As soon as we turn the multitasking on again, by using the */
  /* Permit() function, we may of course start to wait for      */
  /* external events.                                           */
  /*                                                            */
  /* A program that turns off the multitasking is interrupting  */
  /* other programs. You must therefore try to turn the         */
  /* multitaskin on again as soon as possible.                  */
  /*                                                            */
  /* With the new Release 2 you should actually use the special */
  /* LockDosList() and NextDosEntry() functions instead of      */
  /* using the Forbid() and Permit() functions. However, since  */
  /* this program should run on all Amigas we stick to the old  */
  /* methods. (See "Amiga DOS" chapter for more information on  */
  /* the new LockDosList() and NextDosEntry() functions.)       */

  /* 5. Turn the multitaskin OFF: */
  Forbid();



  /* 6. Scan the "DosList" nodes... */

  /* Get a BCPL pointer (BPTR) to the first "DosList" node: */
  temp_bptr = dos_info_ptr->di_DevInfo;

  /* Convert the BPTR into a C pointer: */
  first_doslist_node = (struct DosList *) BADDR( temp_bptr );

  /* Start with the first node: */
  doslist_node = first_doslist_node;

  /* Check all nodes: */
```

```
  while( doslist_node )
  {
    /* There exist three different types of objects we can find:    */
    /*   1. Devices (the "hardware parts") like the disk drives     */
    /*      "df0:", "df1:"..., possible harddisks "hd0:"...,        */
    /*      as well as all the special devices like "PRT:", "SER:", */
    /*      "CON:" etc...                                           */
    /*                                                              */
    /*   2. Volumes, which is the name of the different objects     */
    /*      (the name of the disks e.g. "ACE1:", name of the hard   */
    /*      disk partitions "HD0:", "HD1:" etc...) Each of these    */
    /*      volumes must be in one of the Devices! The "ACE" disk   */
    /*      is maybe in "df0:", and hard disk "HD0:" probably in    */
    /*      device "DH0:" and so on...                              */
    /*                                                              */
    /*   3. Assigns, simle assigns created with help of the Shell   */
    /*      command "Assign".                                       */
    /*                                                              */
    /* Since we only want to print the device names in which there  */
    /* is a volume, we only print the volume names and will then    */
    /* look up the device which the volume is in. Assigns we print  */
    /* directly.                                                    */

    /* Is it a volume? */
    if( doslist_node->dol_Type == DLT_VOLUME )
    {
      /* We have found a volume! Print the name of the */
      /* device in which the volume is:                */
      if( PrintDevice( first_doslist_node,
                       (APTR) doslist_node->dol_Task ) != RETURN_OK )
        printf( "Could not find the volume's device name!" );
      else
        PrintBSTR( doslist_node->dol_Name, 30 );

      printf ( "\n" );
    }

    /* Is it an Assign? */
    if( doslist_node->dol_Type == DLT_DIRECTORY )
    {
      /* It is an Assigs! Simply print the string "<ASN>" and */
      /* add the assign name:                                 */
      printf( "<ASN>   " );
      PrintBSTR( doslist_node->dol_Name, 30 );
      printf ( "\n" );
    }


    /* Get a BPTR to the next node and convert the BPTR */
    /* into a C pointer:                                */
    doslist_node = (struct DosList *) BADDR( doslist_node->dol_Next );
  }

  /* 7. Turn the multitaskin ON again: */
  Permit();
}
```

```
/* Prints the device name of a volume: (With help of the  */
/* address to the "task" which handles our volume we can  */
/* find the corresponding device. Both the device and the */
/* volume use the same "task" if the volume is in the     */
/* device.)                                               */
int PrintDevice
(
  struct DosList *first_node,
  APTR task
)
{
  /* Node pointer: */
  struct DosList *node;



  /* Start with the first node: (Rescan the whole list!) */
  node = first_node;

  /* Check all nodes: */
  while( node )
  {
    /* Check all devices: */
    if( node->dol_Type == DLT_DEVICE )
    {
      /* Has the device a pointer to the same task */
      /* as the volume which we are examining?     */
      if( task == (APTR) node->dol_Task )
      {
        /* Found it! Print the device name: */
        PrintBSTR( node->dol_Name, 8 );

        /* Return with the satisfaction of a job well done... */
        return( RETURN_OK );
      }
    }

    /* Convert the next node's BPTR into a C pointer: */
    node = (struct DosList *) BADDR( node->dol_Next );
  }

  /* Ooops, something is wrong here. Could not find */
  /* the volume's corresponding device name...      */
  return( RETURN_ERROR );
}



/* Handly little function which prints BCPL strings (BSTRs): */
/* (This one will add any necessary spaces to fill the whole */
/* "total_length" with text. This makes it look better since */
/* we can now use nice columns of text.)                     */

void PrintBSTR
(
```

```
  BSTR string_bstr,
  int total_length
)
{
  /* Temporary string pointer */
  UBYTE *string_ptr;

  /* The length of the BCPL string: */
  UBYTE length;

  /* Simple loop variable: */
  int loop;



  /* Conver the BSTR into a normal C pointer to a BCPL string: */
  string_ptr = (UBYTE *) BADDR( string_bstr );

  /* Get the length of the BCPL string: (A BCPL string does not */
  /* contain a NULL sign in the end, but uses instead the first */
  /* byte to tell how many characters the string contains. A    */
  /* BCPL string (BSTR) can therefore not contain more than 255 */
  /* characters.                                                */
  length = string_ptr[ 0 ];

  /* Print BCPL string: */
  for( loop=1; loop <= total_length; loop++ )
    if( loop <= length )
      putchar( string_ptr[ loop ] );
    else
      putchar( ' ' );
}
```