Sourcecode: Example5A.c

COLLABORATORS			
	TITLE : Sourcecode: Example5A.c		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sourcecode: Example5A.c

1.1 Example5A.c

```
/*********************************************************/
/*                                                       */
/* Amiga C Encyclopedia (ACE)           Amiga C Club (ACC) */
/* --------------------------           ------------------ */
/*                                                       */
/* Manual:  AmigaDOS                    Amiga C Club       */
/* Chapter: Handlers                    Tulevagen 22       */
/* File:    Example5A.c                 181 41  LIDINGO    */
/* Author:  Anders Bjerin               SWEDEN             */
/* Date:    93-03-16                                     */
/* Version: 1.4                                          */
/*                                                       */
/*   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 use the Pipe handler   */
/* to copy (pipe) data to another program. This is the first    */
/* part of the example, program A. To see how the Pipe handler  */
/* work you have to start both program A and B. Once both are   */
/* running can you enter some text in this (progra A's) console */
/* window. When you press enter the console will be closed and  */
/* the text you have entered is piped to the other program.     */
/* Program B will receive the text and prints it in it's own    */
/* console window.                                              */
/*                                                              */
/* If you want to transfer several lines of data you have to    */
/* remember that the Pipe handler uses a small buffer (around   */
/* 4000 bytes, 4kB), and you will only be able to collect data  */
/* when the buffer has been filled, or when the file is closed. */
/* In in this example I close the file once the data has been   */
/* sent so it will immediately be available for program B.      */
```

```
/* 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 <clib/exec_protos.h>      /* System functions...         */
#include <stdio.h>                 /* Std functions [printf()...] */
#include <stdlib.h>                /* Std functions [exit()...]   */
#include <string.h>                /* Std functions [strlen()...] */



/* The maximum number of characters that we can store in */
/* our small buffer (including a NULL sign at the end):  */
#define MAX_LENGTH 512



/* Set name and version number: */
UBYTE *version = "$VER: AmigaDOS/Handlers/Example5A 1.4";



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

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

/* Writes text to an already open file: */
/* (e.g. File, Console or Pipe handler) */
int print_text
(
  BPTR file,
  STRPTR text
);

/* Collects text from an already open file:*/
/* (e.g. File, Console or Pipe handler)    */
int collect_text
(
  BPTR file,
  STRPTR text
);



/* Main function: */

int main( int argc, char *argv[] )
{
  /* Store the number of characters used here: */
  int number;

  /* Create a buffers: */
  UBYTE user_input[ MAX_LENGTH ];

  /* A "BCPL" pointer to our Console window: */
```

```
BPTR my_console;

/* A "BCPL" pointer to our Pipe handler: */
BPTR my_pipe;



/* Open a Console window: (Note that we have not added a close */
/* gadget on the window, nor will it wait for the user to type */
/* Ctrl-\ before the window is closed.)                        */
my_console =
 Open( "CON:0/0/640/100/Program A", MODE_OLDFILE );

/* Have we opened the Console successfully? */
if( my_console == NULL )
{
  /* Inform the user: */
  printf( "Error! Could not open the Console device!\n" );

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



/* Open the Pipe handler: (We call the pipe "UserData", and   */
/* we open the pipe handler as a new handler. Program B opens */
/* the handler as an old handler. Note that one program must  */
/* open the handler as new and the other program open the     */
/* handler as old! The order does not matter.)                */
my_pipe =
 Open( "PIPE:UserData", MODE_NEWFILE );

/* Have we opened the Pipe handler successfully? */
if( my_pipe == NULL )
{
  /* Inform the user: */
  printf( "Error! Could not open the Pipe handler!\n" );

  /* Close the Console window: */
  Close( my_console );

  /* Exit with an error code: */
  exit( 21 );
}



/* Tell the user what to do: */
print_text( my_console,
  "1. Start Example7B so it will be ready to collect some text.\n" );
print_text( my_console,
  "2. Type some text in this window and press ENTER (RETURN).\n" );
print_text( my_console,
  "The text will be copied (piped) to the other progrm's window\n" );
print_text( my_console,
  "were it will be printed. This window closes automatically when\n" );
```

```
  print_text( my_console,
    "the text has been piped.\n\nText to send: " );

  /* Collect some text from the Console window: */
  number = collect_text( my_console, user_input );

  /* Send the data to the Pipe handler so the */
  /* other program can collect it:            */
  print_text( my_pipe, user_input );



  /* Close the Pipe handler: */
  Close( my_pipe );

  /* Close the Console window: */
  Close( my_console );

  /* Since we did not set the flag "WAIT" this console window */
  /* will automatically be closed when we close the file.     */



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



/* Writes text to an already opened file, and returns */
/* the number of characters actualy written.          */

int print_text
(
  BPTR file,
  STRPTR text
)
{
  /* Store the number of characters (bytes) actualy written here: */
  int characters_written;



  /* Write the text: */
  characters_written = Write( file, text, strlen( text ) );

  /* Returns the number of characters actually written: */
  return( TRUE );
}



/* Collects text from an already opened file, and returns */
/* the number of characters collected.                    */

int collect_text
(
```

```
  BPTR file,
  STRPTR text
)
{
  /* Store the number of characters (bytes) actualy read here: */
  int characters_read;



  /* Collect some text: */
  characters_read = Read( file, text, MAX_LENGTH - 1 );

  /* Put the NULL ('\0') sign at the end of the string: */
  text[ characters_read ] = NULL;

  /* Returns the number of characters collected: */
  return( characters_read );
}
```