StripCodes

COLLABORATORS			
	TITLE : StripCodes		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 30, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

StripCodes

1.1 StripCodes

```
MODULE StripCodes;

(*
     $RCSfile: StripCodes.mod $
  Description: A utility to strip control codes from text files

   Created by: fjc (Frank Copeland)
    $Revision: 1.5 $
      $Author: fjc $
        $Date: 1994/08/08 16:28:40 $

  Copyright © 1994, Frank Copeland.
  This file is part of Oberon-A.
  See Oberon-A.doc for conditions of use and distribution.

  StripCodes is a program that strips most control characters from a
  file, leaving only the printable characters and tabs, line breaks and
  form feeds.  It is intended to be used to convert wordprocessor files
  into plain text files for import into other programs that do not
  understand the original format.


                Description
                Using StripCodes

                Imported Modules
                History
                Global Switches
```

1.2 History

```
  $Log: StripCodes.mod $
  Revision 1.5  1994/08/08  16:28:40  fjc
  Release 1.4
```

```
  Revision 1.4  1994/06/17  17:35:24  fjc
  - Updated for release

  Revision 1.3  1994/06/09  14:30:29  fjc
  - Incorporates changes to Amiga interface

  Revision 1.2  1994/06/05  22:27:09  fjc
  - Changed to use new Amiga interface

  Revision 1.1  1994/05/12  20:20:07  fjc
  - Prepared for release

  Revision 1.0  1994/01/16  13:59:55  fjc
  Start of version control


  16 Dec 93 [FJC] : Initial version
```

1.3 Global Switches

```
  $P- allow non-portable code
```

1.4 Imported Modules

```
IMPORT
  SYS := SYSTEM, Exec, Dos, DosUtil, Args, Errors, Files, IO := StdIO,
  Str := Strings;

(*
```

1.5 Description of StripCodes

StripCodes was created to solve a particular problem, but it was quickly obvious that it could be applied to a more general class of problems. The particular problem was to convert text files formatted for the text editor in the MS-DOS Ability program into plain ASCII files. The general problem is that of stripping formatting codes from word processor files and leaving the plain text.

StripCodes is a straight-forward filter program: it reads its input, transforms it and writes it out. The input is a file containing ASCII text and unspecified control codes. The output is another file containing the that part of the input consisting of printable characters, and the CR, LF, HT and FF control characters. Stripping the codes is performed by the Strip() procedure.

StripCodes needs to be able to process several files at once. This allows the user to convert all the files in a single directory with one invocation of the program. To do this it accepts an AmigaDOS file pattern as an argument. It then scans for all the files that match that

pattern and converts each one individually. The scanning is performed
by the Scan() procedure.

The output files are sent to a single directory, which is also specified
by the user as a command line argument. If this argument is omitted,
the output files are placed in the same directory as the input files.
The output file name is the same as the input file name, but with a
".DOC" extension in place of the original. The output file name is
constructed by the MakeOutputName() procedure.

The command line arguments are processed by the GetArgs() procedure and
are copied into the global variables pattern and dest.

The main body of the program simply consists of a call to GetArgs() to
get the pattern and destination directory, followed by a call to Scan().

1.6 Using StripCodes

StripCodes requires an Amiga running Kickstart V2.0 or higher. It may
only be run from the CLI at present.

From the CLI, the format for StripCodes is:

    StripCodes <Pattern> [<Destination>]

and the template is:

    PATTERN/A, DESTINATION

Pattern may be any valid AmigaDOS file pattern. Destination must be the
name of a directory that currently exists.

All the files matching Pattern will be processed by the program and the
results placed in Destination with the same name and an extension of
".DOC". If Destination is omitted the output files are placed in the
same directories as the input files.

1.7 Variables

```
(*
  The theoretical maximum size of an AmigaDOS file specification is 255
  characters.
*)
CONST

  PathLen = 255;

TYPE

  Path = ARRAY PathLen + 1 OF CHAR;

VAR
```

```
  pattern,     (* The pattern to be searched for. *)
  dest         (* The destination directory. *)
    : Path;

  (*
    These variables are global so that they may be found by the Cleanup()
    procedure in the event of an abnormal exit
  *)

  input,       (* The current input file. *)
  output       (* The current output file. *)
    : Files.File;

(*
```

1.8 GetArgs()

```
PROCEDURE GetArgs ();

  (*-----------------------------------*)
  PROCEDURE Greetings ();

  BEGIN (* Greetings *)
    IO.WriteStr ("StripCodes\n");
    IO.WriteStr ("Written by Frank Copeland\n\n");
  END Greetings;

  (*-----------------------------------*)
  PROCEDURE Usage ();

  BEGIN (* Usage *)
    IO.WriteStr ("format  : StripCodes <pattern> <destination>\n");
    IO.WriteStr ("template: PATTERN/A, DESTINATION/A\n\n");
    IO.WriteStr ("<pattern>    : files to be converted\n");
    IO.WriteStr ("<destination>: destination directory\n\n");
  END Usage;

BEGIN (* GetArgs *)
  (* Make sure we have been run from the CLI. *)
  Errors.Assert (Args.IsCLI, "Sorry, no Workbench support :-(");

  (* Say hello. *)
  Greetings ();

  (* Check the number of arguments. *)
  IF (Args.argc # 2) & (Args.argc # 3) THEN
    Usage (); HALT (Dos.returnWarn)
  END;

  (* Just copy the pattern. *)
  COPY (Args.argv [1]^, pattern);

  IF Args.argc = 3 THEN
    (* The destination needs to be checked. *)
```

```
    COPY (Args.argv [2]^, dest);
    IF ~DosUtil.FileExists (dest) THEN
      IO.WriteStr ("Destination directory doesn't exist\n");
      HALT (Dos.returnError)
    END; (* IF *)
  ELSE
    dest := ""
  END; (* ELSE *)
END GetArgs;

(*
```

1.9 MakeOutputName()

```
PROCEDURE MakeOutputName
  (inputName : ARRAY OF CHAR; VAR outputName : ARRAY OF CHAR);

  VAR filePart : Exec.STRPTR; i : INTEGER;

BEGIN (* MakeOutputName *)
  IF dest [0] # 0X THEN
    filePart := Dos.base.FilePart (inputName);
    COPY (dest, outputName);
    IF ~Dos.base.AddPart (outputName, filePart^, PathLen) THEN
      IO.WriteStr ("Output file name too big\n");
      HALT (Dos.returnError)
    END;
  ELSE
    COPY (inputName, outputName)
  END; (* ELSE *)
  i := SHORT (Str.Length (outputName)) - 1;
  WHILE (i >= 0) & (outputName [i] # ".") DO DEC (i) END;
  IF i > (PathLen - 4) THEN
    IO.WriteStr ("Output file name too big\n");
    HALT (Dos.returnError)
  ELSE
    IF i >= 0 THEN outputName [i] := 0X END;
    Str.Append (outputName, ".DOC");
  END; (* ELSE *)
END MakeOutputName;

(*
```

1.10 Strip()

```
(*$D-*)
PROCEDURE Strip (inputName : ARRAY OF CHAR);

  CONST
    CR = 0DX; LF = 0AX; TAB = 09X; FF = 0CX;

  VAR outputName : Path; r, w : Files.Rider; ch : CHAR;
```

```
BEGIN (* Strip *)
  IO.WriteF1 ("inputName= %s\n", SYS.ADR (inputName));
  input := Files.Old (inputName);
  IF input # NIL THEN
    Files.Set (r, input, 0);
    MakeOutputName (inputName, outputName);
    IO.WriteF1 ("outputName= %s\n", SYS.ADR (outputName));
    output := Files.New (outputName);
    IF output # NIL THEN
      IO.WriteF2
        (" !! %s -> %s\n", SYS.ADR (inputName), SYS.ADR (outputName));
      Files.Set (w, output, 0);
      WHILE ~r.eof DO
        Files.Read (r, ch);
        IF
          ((ch >= " ") & (ch <= "~")) OR
          (ch = CR) OR (ch = LF) OR (ch = TAB) OR (ch = FF)
        THEN
          Files.Write (w, ch)
        END;
      END; (* WHILE *)
      Files.Register (output); SYS.DISPOSE (output)
    ELSE
      IO.WriteF1 (" !! Could not open %s\n", SYS.ADR (outputName))
    END; (* ELSE *)
    Files.Close (input); SYS.DISPOSE (input)
  ELSE
    IO.WriteF1 (" !! Could not open %s\n", SYS.ADR (inputName))
  END; (* ELSE *)
END Strip;

(*
```

1.11 Scan()

```
Scan uses the standard AmigaDOS functions for enumerating files to
locate all the files that match the pattern provided by the user.  These
functions use a Dos.AnchorPath record to hold data they need between
calls and to return results.  To be useful, the structure must be
extended to provide a place for the path of each file found to be
written by AmigaDOS.  The extended type is declared as MyAnchor.

A call to Dos.MatchFirst() initialises the structure and locates the
first matching file name.  The pattern to be matched is in the global
variable pattern.  Subsequent calls to Dos.MatchNext() locate further
matches.  Both functions return 0 if a match is found and this is used
to drive a loop, ensuring that all files matching the pattern are
discovered.

Each file discovered is passed to Strip() for processing.
*)

PROCEDURE Scan ();
```

```
  VAR myAnchor : Dos.AnchorPath; result : LONGINT;

BEGIN (* Scan *)
  (*
    Allocate an anchor structure and initialise with the length of the
    path variable.
  *)
  myAnchor.strlen := 256;

  (* Find the first file matching the pattern *)
  result := Dos.base.MatchFirst (pattern, myAnchor);
  WHILE result = 0 DO
    (* Strip the file and get the next name. *)
    Strip (myAnchor.buf);
    result := Dos.base.MatchNext (myAnchor)
  END; (* WHILE *)
END Scan;

(*
```

1.12 Cleanup()

```
This procedure is installed as the global cleanup procedure in the
main body of the program.  The previously installed procedure is placed
in the OldCleanup variable and by convention (and necessity) is called
before this procedure exits.

The rc parameter by convention holds the program's return code.  This is
zero for a normal exit.  It may hold some other value if the program
exit is the result of a HALT statement or a run-time error.  It is
ignored in this implementation and is simply passed on to the previously
installed procedure.

Cleanup's task is to ensure that the input and output files are properly
closed when the program exits, to prevent files being lost or corrupted
and avoiding unreleased file locks, which are a right royal pain.  This
is only a concern if the program exits abnormally; that is, through a
HALT statement or run-time error.  The Strip() procedure normally closes
these files and sets the file variables to NIL.  It cannot do this if
the program fails for some reason.  Making the file variables global and
making sure that they are set to NIL when they are closed allows Cleanup
to detect whether they are still open and take the appropriate action.

The output file is purged rather than closed because it is assumed that
the contents will be incomplete or corrupt if the program fails.
*)

PROCEDURE * Cleanup ();

BEGIN (* Cleanup *)
  IF input # NIL THEN Files.Close (input) END;
  IF output # NIL THEN Files.Purge (output) END;
END Cleanup;
```

```
(*
```

1.13 Main Body

```
The Cleanup() procedure is installed as the global cleanup procedure.
The variables it uses are initialised to NIL so that they are guaranteed
to be in defined state at all times.  GetArgs() then obtains the
program's arguments from the command line and Scan() does the work.
*)

BEGIN (* StripCodes *)
  input := NIL; output := NIL;
  SYS.SETCLEANUP (Cleanup);
  GetArgs ();
  Scan ()
END StripCodes.
(*
```