Controlled I/O: a Library for Scope-Based Files

Jeffrey R. Carter
PragmAda Software Engineering; email: jrcarter@acm.org; https://github.com/jrcarter

Abstract

This paper is a summary of the presentation at the 2024 Ada-Europe International Conference Ada Developers Workshop. A recent posting of a wishlist requested "Scope-based files". Controlled I/O is a response to that request.

Keywords: access types, memory management, Ada.

1 Introduction

A recent posting (by user pyj) [3] on the ada-lang.io forum requested "Scope-based files (controlled-type files), that close when they go out of scope". Since wrapping a File_Type in a limited-controlled type, and overriding Finalize to close the file if it is open, is trivial, I concluded that the poster would have spent the few minutes needed to implement it if that was all that was desired. Attempting to think what such a library would include in addition, I created **package** Controlled_IO [1].

2 Specification

Controlled_IO is defined by its specification:

```
with Ada.Directories;
with Interfaces;
package Controlled_IO is
  type File_Handle (<>) is tagged limited private;
  -- Opened/created on creation, closed on finalization
  -- Can be both read and written
  use type Ada.Directories.File_Kind;
  function Opened (Name : in String;
                   Form : in String := "")
  return File_Handle with
    Pre  => Ada.Directories.Exists (Name) and then
           Ada.Directories.Kind (Name) =
           Ada.Directories.Ordinary_File,
    Post => Opened'Result.Position = 1;
  -- Opens an existing file
  function Created (Name : in String;
                    Form : in String := "")
  return File_Handle with
    Post => Created'Result.Position = 1;
  -- Creates a new file named Name (deleting any
existing file named Name)
  function Opened_Or_Created (Name : in String;
                              Form : in String := "")
  return File_Handle is
    (if Ada.Directories.Exists (Name) then
       Opened (Name, Form)
     else
       Created (Name, Form) );
  function End_Of_File (File : in File_Handle)
  return Boolean;
  -- Returns File.Position > File.Size
  subtype Byte is Interfaces.Unsigned_8;
  type Byte_List is array (Positive range <>) of Byte;
  type Count_Value is mod 2 ** 64;
  subtype Position_Value is Count_Value range
    1 .. Count_Value'Last;
  function Size (File : in File_Handle)
  return Count_Value;
  -- Returns the current number of bytes in File
  procedure Set_Position (File : in out File_Handle;
                          Position : in Position_Value)
  with
    Pre => Position in 1 .. File.Size + 1;
  -- Sets the current position of File to Position
  function Position (File : in File_Handle)
  return Position_Value;
  -- Returns the current position in File
  function Next (File : in out File_Handle)
  return Byte with
    Pre => not File.End_Of_File;
  -- Returns the byte in File at the current position and
increments the current position
  procedure Read (File : in out File_Handle;
                  List :    out Byte_List)
  with
    Pre => not File.End_Of_File and then
         File.Size - File.Position + 1 >= List'Length;
  -- Calls File.Next for every Byte in List
  procedure Write
    (File : in out File_Handle; Value : in Byte);
  -- Writes Value to File at the current position and
increments the current position
  procedure Write
    (File : in out File_Handle; Value : in Byte_List);
  -- Calls File.Write for every Byte in Value
private -- Controlled_IO
...
end Controlled_IO;
```

A File_Handle is open when it exists, and closed when it ceases to exist. All files allow input, output, and seeking.

3 Children

There are two child packages of Controlled_IO: Text and UTF. Text implements text I/O similar to Ada.Text_IO. UTF implements the Universal Text File format [2] (not to be confused with Unicode Transformation Format).

4 Example/test programs

There are also three example programs.

Controlled_Test and Controlled_Text are user-unfriendly file-copy programs. Controlled_Test performs a binary copy; the output should always be identical to the input. Controlled_Text performs a line-by-line copy of text files; the output may have different line terminators than the input.

Controlled_UTF is a user-unfriendly program to convert a native text file to a Universal Text File.

5 Summary

Controlled I/O is a library for scope-based files that is somewhat more complete and complex than simply wrapping a File_Type in a limited-controlled type and overriding Finalize to close the file if it is open. Files are open when they exist, and closed when they cease to exist. All files allow input, output, and seeking.

References

[1] J. Carter, Controlled_IO implementation, [https://github.com/jrcarter/Controlled_IO].

[2] J. Carter, Universal Text File implementation, [https://github.com/jrcarter/Universal-Text-File].

[3] pyj, Ada wishlist, [https://forum.ada-lang.io/t/ada-library-wishlist/14/5].