

## Package Text\_IO

```
with Io_Exceptions;
package Text_IO is

  type FILE_TYPE is
    limited private;

  type FILE_MODE is
    (inFILE, outFILE);

  type COUNT is
    range 0.. INTEGER>Last;

  subtype POSITIVE_COUNT is
    COUNT range 1 .. COUNT>Last;

  Unbounded      -- line and page length
  ; constant COUNT := 0;

  subtype FIELD is
    INTEGER range 0.. INTEGER>Last;

  subtype NUMBER_BASE is
    INTEGER range 2 .. 16;

  type TYPE_SET is
    (LOWER_CASE, UPPER_CASE);

  -- file management

  procedure Create
    ( File      : in out FILE_TYPE;
      Mode      : in FILE_MODE := out_FILE;
      Name      : in STRING := "";
      Form      : in STRING := "" );

  procedure Open
    ( File      : in out FILE_TYPE;
      Mode      : in FILE_MODE;
      Name      : in STRING;
      Form      : in STRING := "" );

  Ingénierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-1-11/10/08-08:56AM v1.1.10
```

## Package Text\_IO

```
procedure Close (
  File : in out File_Type );
procedure Delete (
  File : in out File_Type );
procedure Reset (
  File : in out File_Type;
  Mode : in File_Mode );
procedure Reset (
  File : in out File_Type );

function Mode (
  File : in File_Type )
return File_Mode;
function Name (
  File : in File_Type )
return String;
function Form (
  File : in File_Type )
return String;

function Is_Open (
  File : in File_Type )
return Boolean;

-- control of default input and
-- output files

procedure Set_Input (
  File : in File_Type );
procedure Set_Output (
  File : in File_Type );

function Standard_Input return File_Type;
function Standard_Output return File_Type;
function Standard_Error return File_Type;

  Ingénierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-2-11/10/08-08:56AM v1.1.10
```

## Package Text\_IO

```
function Current_Input return File_Type;
function Current_Output return File_Type;

-- specification of line and
-- page lengths

procedure Set_Line_Length (
  File : in File_Type;
  To : in Count );
procedure Set_Line_Length (
  To : in Count );

procedure Set_Page_Length (
  File : in File_Type;
  To : in Count );
procedure Set_Page_Length (
  To : in Count );

function Line_Length (
  File : in File_Type )
return Count;
function Line_Length return Count;

function Page_Length (
  File : in File_Type )
return Count;
function Page_Length return Count;

-- column, line, and page control

procedure New_Line (
  File : in File_Type;
  Spacing : in Positive_Count := 1 );
procedure New_Line (
  Spacing : in Positive_Count := 1 );
```

Ingenierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-3-11/18/98-08:56AM v.1.1a

## Package Text\_IO

```
procedure Skip_Line (
  File : in File_Type;
  Spacing : in Positive_Count := 1 );
procedure Skip_Line (
  Spacing : in Positive_Count := 1 );

function End_Of_Line (
  File : in File_Type )
return Boolean;
function End_Of_Line return Boolean;

procedure New_Page (
  File : in File_Type );
procedure New_Page;

procedure Skip_Page (
  File : in File_Type );
procedure Skip_Page;

function End_Of_Page (
  File : in File_Type )
return Boolean;
function End_Of_Page return Boolean;

function End_Of_File (
  File : in File_Type )
return Boolean;
function End_Of_File return Boolean;

procedure Set_Col (
  File : in File_Type;
  To : in Positive_Count );
procedure Set_Col (
  To : in Positive_Count );
```

Ingenierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-4-11/18/98-08:56AM v.1.1a

## Package Text\_IO

```
procedure Set_Line (
  File : in File_Type;
  To : in Positive_Count);
procedure Set_Line (
  To : in Positive_Count);

function Col (
  File : in File_Type )
  return Positive_Count;
function Col return Positive_Count;

function Line (
  File : in File_Type )
  return Positive_Count;
function Line return Positive_Count;

function Page (
  File : in File_Type )
  return Positive_Count;
function Page return Positive_Count;

-- character input-output

procedure Get (
  File : in File_Type;
  Item : out Character );
procedure Get (
  Item : out Character );
procedure Put (
  File : in File_Type;
  Item : in Character );
procedure Put (
  Item : in Character );
```

Ingenierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-6-11/18/98-08:56AM v1.1.10

## Package Text\_IO

```
-- string input-output

procedure Get (
  File : in File_Type;
  Item : out String );
procedure Get (
  Item : out String );
procedure Put (
  File : in File_Type;
  Item : in String );
procedure Put (
  Item : in String );

procedure Get_Line (
  File : in File_Type;
  Item : out String;
  Last : out Natural );
procedure Get_Line (
  Item : out String;
  Last : out Natural );
procedure Put_Line (
  File : in File_Type;
  Item : in String );
procedure Put_Line (
  Item : in String );

-- generic package for input-output
-- of INTEGER types

generic
  type Num is range <>;
package Integer_IO is
  Default_Width
    : Field := Num*Width;

  Default_Base
    : Number_Base := 10;
```

Ingenierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-6-11/18/98-08:56AM v1.1.10

## Package Text\_IO

```
procedure Get (
  File : in File_Type;
  Item : out Num;
  Width : in Field := 0);
procedure Get (
  Item : out Num;
  Width : in Field := 0);
procedure Put (
  File : in File_Type;
  Item : in Num;
  Width : in Field := Default_Width;
  Base : in Number_Base := Default_Base );
procedure Put (
  Item : in Num;
  Width : in Field := Default_Width;
  Base : in Number_Base := Default_Base );

procedure Get (
  From : in String;
  Item : out Num;
  Last : out Positive );
procedure Put (
  To : out String;
  Item : in Num;
  Base : in Number_Base := Default_Base );

end Integer_IO;

-- generic package for input-output of real types
```

Ingenierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-7-11/10/08-08:56AM v.1.1.10

## Package Text\_IO

```
-- generic package for input-output
-- of INTEger types

generic
type Num is range <>;
package Integer_IO is
  Default_Width
    : Field := Num*Width;

  Default_Base
    : Number_Base := 10;

  procedure Get (
    File : in File_Type;
    Item : out Num;
    Width : in Field := 0);
  procedure Get (
    Item : out Num;
    Width : in Field := 0);

  procedure Put (
    File : in File_Type;
    Item : in Num;
    Width : in Field := Default_Width;
    Base : in Number_Base := Default_Base );
  procedure Put (
    Item : in Num;
    Width : in Field := Default_Width;
    Base : in Number_Base := Default_Base );

  procedure Get (
    From : in String;
    Item : out Num;
    Last : out Positive );
  procedure Put (
    To : out String;
    Item : in Num;
    Base : in Number_Base := Default_Base );

end Integer_IO;

Ingenierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-8-11/10/08-08:56AM v.1.1.10
```

## Package Text\_IO

```
-- generic package for input-output of real types
generic
type Num is digits <>;
package Float_IO is
  Default_Fore
    : Field := 2;

  Default_Aft
    : Field := Num'digits - 1;

  Default_Exp
    : Field := 3;
  procedure Get (
    File : in File_Type;
    Item : out Num;
    Width : in Field := 0 );
  procedure Get (
    Item : out Num;
    Width : in Field := 0 );
  procedure Put (
    File : in File_Type;
    Item : in Num;
    Fore : in Field := Default_Fore;
    Aft : in Field := Default_Aft;
    Exp : in Field := Default_Exp );
  procedure Put (
    Item : in Num;
    Fore : in Field := Default_Fore;
    Aft : in Field := Default_Aft;
    Exp : in Field := Default_Exp );
  procedure Get (
    From : in String;
    Item : out Num;
    Last : out Positive );
  procedure Put (
    To : out String;
    Item : in Num;
    Aft : in Field := Default_Aft;
    Exp : in Field := Default_Exp );
end Float_IO;
```

Ingenierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-9-11/18/98-08:56 AM v.1.10

## Package Text\_IO

```
generic
type Num is delta <>;
package Fixed_IO is
  Default_Fore
    : Field := Num'Fore;
  Default_Aft
    : Field := Num'Aft;
  Default_Exp
    : Field := 0;
  procedure Get (
    File : in File_Type;
    Item : out Num;
    Width : in Field := 0 );
  procedure Get (
    Item : out Num;
    Width : in Field := 0 );
  procedure Put (
    File : in File_Type;
    Item : in Num;
    Fore : in Field := Default_Fore;
    Aft : in Field := Default_Aft;
    Exp : in Field := Default_Exp );
  procedure Put (
    Item : in Num;
    Fore : in Field := Default_Fore;
    Aft : in Field := Default_Aft;
    Exp : in Field := Default_Exp );
  procedure Get (
    From : in String;
    Item : out Num;
    Last : out Positive );
  procedure Put (
    To : out String;
    Item : in Num;
    Aft : in Field := Default_Aft;
    Exp : in Field := Default_Exp );
end Fixed_IO;
```

Ingenierie du Logiciel avec Ada - © Louis Granger -Ada-lab10-10-11/18/98-08:56 AM v.1.10

## Package Text\_IO

```
generic
type Enum is
(<>);
package Enumeration_io is
  Default_Width
    : Field := 0;

  Default_Setting
    : Type_Set := Upper_Case;

  procedure Get (
    File : in File_Type;
    Item : out Enum );
  procedure Get (
    Item : out Enum );

  procedure Put (
    File : in File_Type;
    Item : in Enum := Default_Width;
    Width : in Field := Default_Width;
    Set : in Type_Set := Default_Setting );
  procedure Put (
    Item : in Enum := Default_Width;
    Width : in Field := Default_Width;
    Set : in Type_Set := Default_Setting );

  procedure Get (
    From : in String;
    Item : out Enum;
    Last : out Positive );
  procedure Put (
    To : out String;
    Item : in Enum;
    Set : in Type_Set := Default_Setting );
end Enumeration_io;
```

Ingenierie du Logiciel avec Ada - 0. Louis Granger - Ada-lab10-11-11/18/9808:56 AM s.r.t. 10

## Package Text\_IO

```
- exceptions

Status_Error : exception
renames Io_Exceptions.Status_Error;

Mode_Error : exception
renames Io_Exceptions.Mode_Error;

Name_Error : exception
renames Io_Exceptions.Name_Error;

Use_Error : exception
renames Io_Exceptions.Use_Error;

Device_Error : exception
renames Io_Exceptions.Device_Error;

End_Error : exception
renames Io_Exceptions.End_Error;

Data_Error : exception
renames Io_Exceptions.Data_Error;

Layout_Error : exception
renames Io_Exceptions.Layout_Error;

private - Text_IO
type File_Object;
type File_Type is access File_Object;
-- detail omitted
end Text_io;
```

Ingenierie du Logiciel avec Ada - 0. Louis Granger - Ada-lab10-12-11/18/9808:56 AM s.r.t. 10

## Package Text\_IO-- Exemple

```
-- Problem 1.1
-- by Rick Conn
with Text_IO;           -- context specification

procedure Count_Down is -- an Ada mainline is always a
  procedure
    Number_Of_Iterations : Integer := 1;
    -- local variable definition
  package Int_Io is
    new Text_IO.Integer_IO (Integer); -- local package for
    Integer I/O
  begin -- Count_Down

    Text_IO.Put("Enter number of iterations: ");
    Int_Io.Get(Number_Of_Iterations);
    Text_IO.New_Line;

    Text_IO.Put("The loop will run for ");
    Int_Io.Put(Number_Of_Iterations);
    Text_IO.Put_Line(" iterations");

    for Index in 1 .. Number_Of_Iterations loop
      Text_IO.Put("iteration: ");
      Int_Io.Put(Index, 4);
      Text_IO.New_Line;           -- 4 column field
    end loop;
  end Count_Down;
```

Ingénierie du Logiciel avec Ada - 3. Louis Granger - Ada-lab10-13-11/18/9808:56 AM 10.11.10

## Package Text\_IO-- Exemple

```
Output

Enter number of iterations: 12
The loop will run for      12 iterations
iteration:  1
iteration:  2
iteration:  3
iteration:  4
iteration:  5
iteration:  6
iteration:  7
iteration:  8
iteration:  9
iteration: 10
iteration: 11
iteration: 12
```

Ingénierie du Logiciel avec Ada - 3. Louis Granger - Ada-lab10-14-11/18/9808:56 AM 10.11.10