

**Rational Environment  
Reference Manual**

**Text Input/Output (TIO)**

Copyright © 1985, 1986, 1987 by Rational

Document Control Number: 8001A-26

Rev. 1.0, October 1985

Rev. 2.0, December 1985

Rev. 3.0, May 1986

Rev. 4.0, July 1987 (Delta)

This document subject to change without notice.

Note the Reader's Comments form on the last page of this book, which requests the user's evaluation to assist Rational in preparing future documentation.

Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

Rational and R1000 are registered trademarks and Rational Environment and Rational Subsystems are trademarks of Rational.

VT100 is a trademark of Digital Equipment Corporation.

Rational  
1501 Salado Drive  
Mountain View, California 94043

## Contents

<b>How to Use This Book</b> . . . . .	xi
<b>Key Concepts</b> . . . . .	1
Files . . . . .	1
Devices and Windows . . . . .	2
File Handles . . . . .	4
Filenames . . . . .	4
Access Control . . . . .	5
Concurrency . . . . .	5
Representations of Terminators . . . . .	5
Exceptions . . . . .	6
Error Reactions . . . . .	6
<b>package Io</b> . . . . .	7
function "=" . . . . .	9
function "<" . . . . .	10
function ">" . . . . .	11
procedure Append . . . . .	12
procedure Close . . . . .	14
function Col . . . . .	15
function Convert . . . . .	16
function Convert . . . . .	17
procedure Convert . . . . .	18
subtype Count . . . . .	19
procedure Create . . . . .	20
function Current_Error . . . . .	22
function Current_Input . . . . .	23
function Current_Output . . . . .	24
procedure Delete . . . . .	25

procedure Echo	26
procedure Echo	27
procedure Echo	28
procedure Echo	29
procedure Echo	31
procedure Echo_Line	32
function End_Of_File	33
function End_Of_Line	34
function End_Of_Page	35
subtype Field	36
subtype File_Mode	37
type File_Type	38
procedure Flush	39
function Form	40
procedure Get	41
procedure Get	42
procedure Get	43
procedure Get	45
procedure Get	47
procedure Get	49
function Get_Line	50
procedure Get_Line	51
constant In_File	53
function Is_Open	54
function Line	55
function Line_Length	56
constant Lower_Case	57
function Mode	58
function Name	59
procedure New_Line	60
procedure New_Page	61
subtype Number_Base	62
procedure Open	63
constant Out_File	65
function Page	66
function Page_Length	67
procedure Pop_Error	68
procedure Pop_Input	69

procedure Pop_Output . . . . .	70
subtype Positive_Count . . . . .	71
procedure Put . . . . .	72
procedure Put . . . . .	73
procedure Put . . . . .	74
procedure Put . . . . .	76
procedure Put . . . . .	78
procedure Put_Line . . . . .	79
procedure Reset . . . . .	80
procedure Reset_Error . . . . .	81
procedure Reset_Input . . . . .	82
procedure Reset_Output . . . . .	83
procedure Save . . . . .	84
procedure Set_Col . . . . .	85
procedure Set_Error . . . . .	87
procedure Set_Input . . . . .	89
procedure Set_Line . . . . .	91
procedure Set_Line_Length . . . . .	93
procedure Set_Output . . . . .	94
procedure Set_Page_Length . . . . .	96
procedure Skip_Line . . . . .	97
procedure Skip_Page . . . . .	98
function Standard_Error . . . . .	99
function Standard_Input . . . . .	100
function Standard_Output . . . . .	101
subtype Type_Set . . . . .	102
constant Unbounded . . . . .	103
constant Upper_Case . . . . .	104
generic procedure Wildcard_Iterator . . . . .	105
generic formal procedure Note_Error . . . . .	106
generic formal procedure Process . . . . .	107
procedure Wildcard_Iterator . . . . .	108
end Wildcard_Iterator	
generic package Enumeration_Io . . . . .	109
constant Default_Setting . . . . .	110
constant Default_Width . . . . .	111
generic formal type Enum . . . . .	112
procedure Get . . . . .	113

procedure Get . . . . .	114
procedure Put . . . . .	115
procedure Put . . . . .	117
end Enumeration_Io	
generic package Fixed_Io . . . . .	119
constant Default_Aft . . . . .	120
constant Default_Exp . . . . .	121
constant Default_Fore . . . . .	122
procedure Get . . . . .	123
procedure Get . . . . .	125
generic formal type Num . . . . .	126
procedure Put . . . . .	127
procedure Put . . . . .	129
end Fixed_Io	
generic package Float_Io . . . . .	131
constant Default_Aft . . . . .	132
constant Default_Exp . . . . .	133
constant Default_Fore . . . . .	134
procedure Get . . . . .	135
procedure Get . . . . .	137
generic formal type Num . . . . .	138
procedure Put . . . . .	139
procedure Put . . . . .	141
end Float_Io	
generic package Integer_Io . . . . .	143
constant Default_Base . . . . .	144
constant Default_Width . . . . .	145
procedure Get . . . . .	146
procedure Get . . . . .	148
generic formal type Num . . . . .	149
procedure Put . . . . .	150
procedure Put . . . . .	152
end Integer_Io	
<b>end Io</b>	

<b>package Io_Exceptions</b> . . . . .	155
exception Data_Error . . . . .	156
exception Device_Error . . . . .	157
exception End_Error . . . . .	158
exception Layout_Error . . . . .	159
exception Mode_Error . . . . .	160
exception Name_Error . . . . .	161
exception Status_Error . . . . .	162
exception Use_Error . . . . .	163

**end Io\_Exceptions**

<b>package Text_Io</b> . . . . .	165
procedure Close . . . . .	166
function Col . . . . .	167
type Count . . . . .	168
procedure Create . . . . .	169
function Current_Input . . . . .	171
function Current_Output . . . . .	172
procedure Delete . . . . .	173
function End_Of_File . . . . .	174
function End_Of_Line . . . . .	175
function End_Of_Page . . . . .	176
subtype Field . . . . .	177
type File_Mode . . . . .	178
type File_Type . . . . .	179
function Form . . . . .	180
procedure Get . . . . .	181
procedure Get . . . . .	182
procedure Get_Line . . . . .	183
function Is_Open . . . . .	185
function Line . . . . .	186
function Line_Length . . . . .	187
function Mode . . . . .	188
function Name . . . . .	189
procedure New_Line . . . . .	190
procedure New_Page . . . . .	191
subtype Number_Base . . . . .	192
procedure Open . . . . .	193

function Page . . . . .	195
function Page_Length . . . . .	196
subtype Positive_Count . . . . .	197
procedure Put . . . . .	198
procedure Put . . . . .	199
procedure Put_Line . . . . .	200
procedure Reset . . . . .	201
procedure Set_Col . . . . .	203
procedure Set_Input . . . . .	205
procedure Set_Line . . . . .	206
procedure Set_Line_Length . . . . .	208
procedure Set_Output . . . . .	209
procedure Set_Page_Length . . . . .	210
procedure Skip_Line . . . . .	211
procedure Skip_Page . . . . .	212
function Standard_Input . . . . .	213
function Standard_Output . . . . .	214
type Type_Set . . . . .	215
constant Unbounded . . . . .	216
generic package Enumeration_Io . . . . .	217
constant Default_Setting . . . . .	218
constant Default_Width . . . . .	219
generic formal type Enum . . . . .	220
procedure Get . . . . .	221
procedure Get . . . . .	222
procedure Put . . . . .	223
procedure Put . . . . .	225
end Enumeration_Io	
generic package Fixed_Io . . . . .	227
constant Default_Aft . . . . .	228
constant Default_Exp . . . . .	229
constant Default_Fore . . . . .	230
procedure Get . . . . .	231
procedure Get . . . . .	233
generic formal type Num . . . . .	234
procedure Put . . . . .	235
procedure Put . . . . .	237
end Fixed_Io	



generic package Float_Io . . . . .	239
constant Default_Aft . . . . .	240
constant Default_Exp . . . . .	241
constant Default_Fore . . . . .	242
procedure Get . . . . .	243
procedure Get . . . . .	245
generic formal type Num . . . . .	246
procedure Put . . . . .	247
procedure Put . . . . .	249
end Float_Io	
generic package Integer_Io . . . . .	251
constant Default_Base . . . . .	252
constant Default_Width . . . . .	253
procedure Get . . . . .	254
procedure Get . . . . .	256
generic formal type Num . . . . .	257
procedure Put . . . . .	258
procedure Put . . . . .	260
end Integer_Io	
<b>end Text_Io</b>	
<b>Index . . . . .</b>	<b>263</b>

RATIONAL

## How to Use This Book

The Text Input/Output (TIO) book of the *Rational Environment Reference Manual* contains reference information describing some of the I/O packages for manipulating text files provided by the Rational Environment™. This includes reference information on the Ada®-predefined packages Text\_Io and Io\_Exceptions, as well as information on Rational®-developed I/O packages. Note that packages for performing I/O on binary data and to devices or editor windows are documented in the Data and Device Input/Output (DIO) of the *Rational Environment Reference Manual*. The reference entries for package !Io.Io\_Exceptions are duplicated in both TIO and DIO, because these exceptions can be raised by any of the I/O packages.

### Organization of the Reference Manual

The *Rational Environment Reference Manual* (Reference Manual for brevity) includes the following volumes (see accompanying illustration):

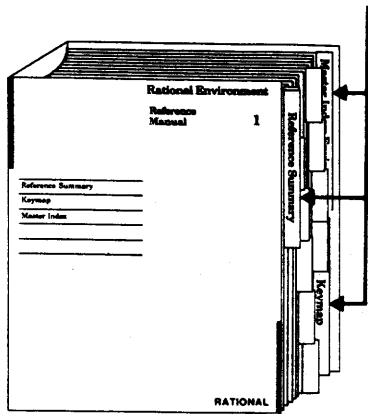
- 1 Reference Summary  
Keymap  
Master Index
- 2 Editing Images (EI)  
Editing Specific Types (EST)
- 3 Debugging (DEB)
- 4 Session and Job Management (SJM)
- 5 Library Management (LM)
- 6 Text Input/Output (TIO)
- 7 Data and Device Input/Output (DIO)
- 8 String Tools (ST)
- 9 Programming Tools (PT)
- 10 System Management Utilities (SMU)
- 11 Project Management (PM)

Each *volume* of the Reference Manual contains one or more *books* separated by large colored tabs. Each book contains information on particular features or areas of application in the Environment. The abbreviation for the name of each book (for example, EI for Editing Images) appears on the binder cover and spine, and this abbreviation is used in page numbers and cross-references. The books grouped into one volume are not necessarily logically related.

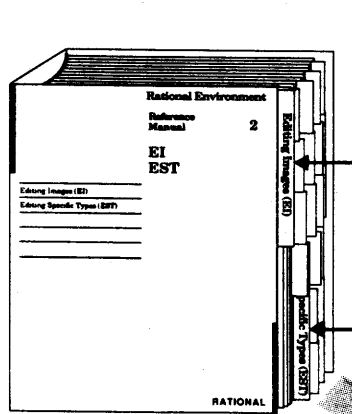
# Organization of the Rational Environment Reference Manual

11 volumes containing 14 books

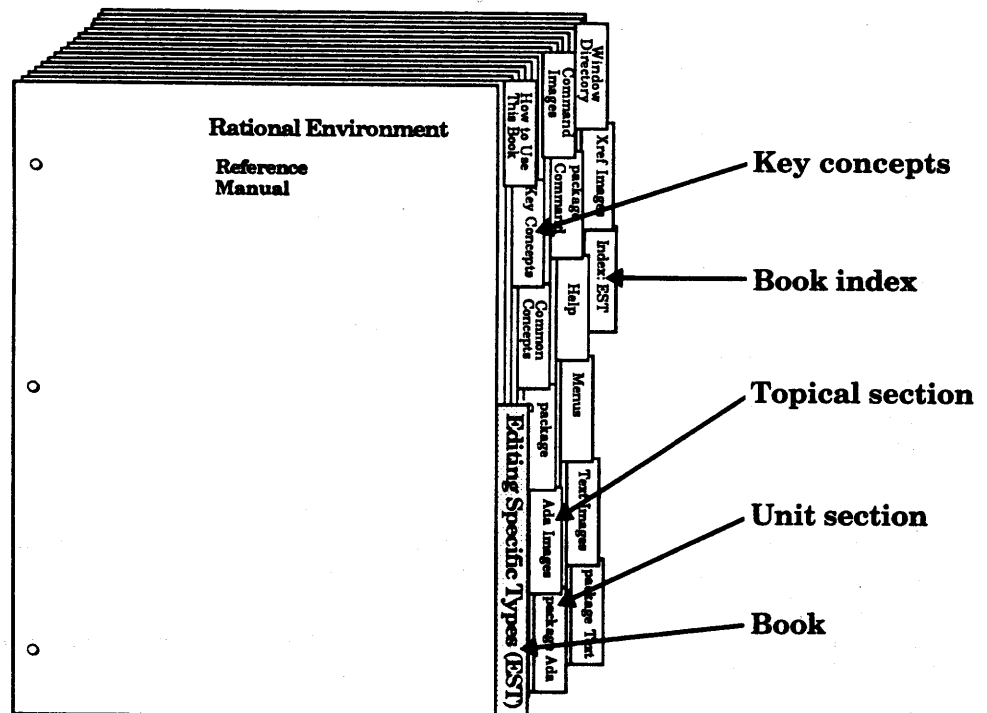
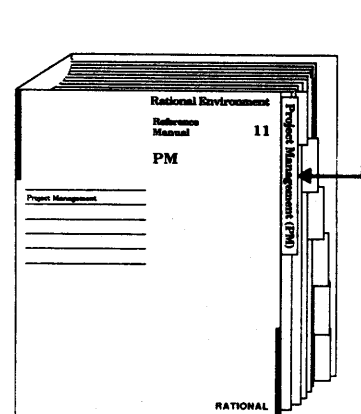
Volume 1: 3 books



Volume 2: 2 books



Volume 11: 1 book



A sample book

The Reference Manual provides reference information organized to efficiently answer specific questions about the Rational Environment. The *Rational Environment User's Guide* complements this manual, providing a user-oriented introduction to the facilities of the Environment. Products other than the Rational Environment (for example, Rational Networking—TCP/IP or Rational Target Build Utility) are documented in individual manuals, which are not part of the Reference Manual.

### Volume 1

Volume 1, intended to be used as a quick reference to the resources provided by the Environment, contains the following books:

- **Reference Summary:** The Reference Summary contains the full Ada specification for each unit in the standard Environment. The unit specifications are organized by their pathnames. The World ! section provides a list of the units in the library system of the Environment and the manual/book in which they are documented.
- **Keymap:** The Rational Environment Keymap presents the standard Environment key bindings, organized by topic and by command name. The topical section includes both a quick reference for commonly used commands and a more detailed reference for key bindings.
- **Master Index:** The Master Index combines all of the index information for each of the books in the Reference Manual.

### Volumes 2-11

Each book in Volumes 2-11 begins with a colored tab on which the name of the book appears. Each book typically contains the following sections:

- **Contents:** The table of contents provides a complete list of all the units in the book and their reference entries.
- **Key Concepts section:** Most of the books contain a section describing key concepts that pertain to all of the Environment facilities documented in that book. This section is located behind its own tab after the table of contents.
- **Unit sections:** Each of the commands, tools, and so on has a declaration within an Ada compilation unit (typically a package) in the Environment library system. For each unit, there is a section that contains reference entries for the declarations (for example, procedures, functions, and types) within that unit. Each section is preceded by a tab.

The sections for units are alphabetized by the simple names of the units. For example, the section for package !Tools.String\_Utilities is alphabetized under String\_Utilities.

For many units, introductory material and/or examples specific to the unit appear after the section tabs.

Within the section for a given unit, the reference entries describing the unit's declarations are organized alphabetically after the section introduction. Appearing at the top of each page in a reference entry are the simple name of the given declaration and the fully qualified pathname of the enclosing unit.

- **Explanatory/topical sections:** Like the unit sections, explanatory/topical sections are preceded by tabs, and they are alphabetized with the unit sections. The topical sections, such as Help, located in Editing Specific Types (EST), discuss Environment facilities.
- **Index:** Preceded by a tab, the Index appears as the last section of each book. It contains entries for each unit or declaration, along with additional topical references. Each book index covers only the material documented in that particular book. The Master Index (in Volume 1) provides entries for the information documented in all the books within the Reference Manual.

Italic page numbers indicate the page on which the primary reference entry for a declaration appears; nonitalic page numbers indicate key concepts, defined terms, cross-references, and exceptions raised.

## Suggestions for Finding Information

The following suggestions may help you in finding various kinds of information in the documentation for Rational's products.

### Learning about Environment Facilities

If you are a novice user starting to use the Environment, consult the *Rational Environment User's Guide*.

If you are familiar with the Environment but are interested in learning about the Environment's library-management commands, for example, you might start by scanning the specifications for these units in the Reference Summary to get an idea of the kinds of things these tools can do. You should also look at the Key Concepts for the particular book, which describes important concepts and gives examples.

It may also be useful to glance through the introductions provided for some of the units in the book. These introductions, located immediately after the tabs for the units, often contain helpful examples.

### Finding Information on a Specific Item

If you know the name of the item and the book in which it is documented, consult either the table of contents or the index for that book. You can also turn through the pages of the book using the names and pathnames of the reference entries to locate the entry you want. Remember that the reference entries for a unit are organized alphabetically within the unit, and the units are organized alphabetically by simple name within the book.

If you know the simple name of the entry but do not know the book in which it is documented, look in the Master Index (in Volume 1) to find the book abbreviation and page number.

If you know the pathname of the entry but do not know the book in which it is documented, the World ! section of the Reference Summary (in Volume 1) provides a map of the units in the library system of the Environment and the books in which they are documented.

If you cannot find an item in the Master Index, the item either is not documented or is documented in the manuals for a product other than the Rational Environment (for example, Rational Networking—TCP/IP or Rational Target Build Utility). If you know the pathname, consult the World ! section of the Reference Summary to determine whether that item is documented and in which manual.

### Using the Index

The index of each book contains entries for each unit and its declarations, organized alphabetically by simple name. When using the index to find a specific item, consult the italic page number for the primary reference for that item. Nonitalic page numbers indicate key concepts, defined terms, cross-references, and exceptions raised.

### Viewing Specifications On-Line

If you know the pathname of a declaration and want to see its specification in a window of the Rational Environment, provide its pathname to the Common-Definition procedure—for example, Definition ("!Commands.Library");. If you know the simple name of the unit in which the declaration appears, in most cases you can use searchlist naming as a quick way of viewing the unit—for example, Definition ("\Library");.

### Using On-Line Help

Most of the information contained in the reference entries for each unit is available through the on-line help facilities of the Environment. Press the **Help on Help** key or consult the *Rational Environment User's Guide* or the *Rational Environment Reference Manual*, EST, Help, for more information on using this on-line help facility.

### Cross-Reference Conventions

The following conventions are used in cross-references to information:

- **Specific page/book:** For references to a specific place in a specific book, the book abbreviation is followed by the page number in the book (for example, LM-322). If the book abbreviation is omitted, the current book is implied (for example, the page numbers in the table of contents for a book do not include the book prefix).
- **Declaration in same unit:** References to the documentation for a declaration in the same unit are indicated by the simple name of the desired declaration. For example, within the reference entry for the Library.Copy procedure, a reference to the Library.Move procedure would be simply "procedure Move." Note that if there are nested packages in the unit, references to nested declarations use qualified pathnames.
- **Declaration in different unit, same book:** References to the documentation for a declaration in another unit are indicated by the qualified pathname of the desired declaration. For example, within the reference entry for the Library.Copy procedure, a reference to the Compilation.Delete procedure would be "procedure Compilation.Delete."

- **Declaration in different book:** References to the documentation for a declaration in another book are indicated by the addition of the abbreviation for that book. For example, within the reference entry for the Library.Copy procedure, a reference to the Editor.Region.Copy procedure in the Editing Images book would be "EI, procedure Editor.Region.Copy."

References to specific declarations in the library system of the Rational Environment (not the documentation for them) are typically indicated by fully qualified pathnames—for example, "procedure !Commands.Library.Copy." When the context is clear, however, a shorter name will be used. If the unit in which the declaration appears is undocumented, you may want to see its explanatory comments to understand what it does. To see these comments, either look at the unit's specification in the Reference Summary or view it on-line using the Rational Environment.

### **Feedback to Rational: Reader's Comments Form**

Rational wants to make its documentation as useful and error-free as possible. Please provide us with feedback. The last page of each book contains a Reader's Comments form that you can use to send us comments or to report errors. You can also submit problem reports and make suggestions electronically by using the SIMS problem-reporting system. If you use SIMS to submit documentation comments, please indicate the manual name, book name, and page number.



## Key Concepts

Text Input/Output (TIO) contains reference information describing the I/O packages for manipulating *text files*. Text files are files that contain ASCII characters, which are of the Character type intended for viewing, editing, and so on.

The packages documented in this book include the Ada-predefined packages Text\_Io and Io\_Exceptions as well as an additional, more powerful, Rational-provided package called package Io.

Package Io provides all of the facilities of package Text\_Io and other additional useful facilities, including a functional form of the Get\_Line procedure; preinstantiated Get and Put procedures for the Integer, Float, and Boolean types; and facilities for performing I/O on the Message window. It provides the mechanism of the Standard\_Error and Current\_Error functions, which is similar to the mechanism in the Standard\_Output and Current\_Output functions, but which defaults output to the Message window. The Io file type is private, rather than limited private, making some applications easier to develop. Package Io also provides file-type conversion operations to Text\_Io and Device\_Independent\_Io file types and a generic iterator for performing a generic operation on a set of files matching a wildcard.

Other packages available in the Environment for performing I/O are documented in Data and Device Input/Output (DIO).

### Files

The I/O packages manipulate information in objects stored in the library system of the Rational Environment. This includes files, Ada units, and devices such as windows and terminals. Since the Rational Environment offers a richer definition of the file than does the *Reference Manual for the Ada Programming Language*, in the description of all the I/O facilities in the Rational Environment, the term *files* may at times be used to denote any one of these entities.

Files that are objects of the file class in the library system of the Environment can be read from or written to. In the Rational Environment, a file is identified in a library display with an entry of the form:

## Key Concepts

```
name : file;
```

where *name* is the identifier of the simple name of the file. Files can exist only in libraries—that is, directories or worlds. Files can be created, opened, closed, deleted, and otherwise read from/written to by any of the Environment I/O packages. EST, package Text, provides facilities for text-specific editing of files, and a file append operation is available in LM, package File\_Utilities, and in package Io. It is common for a file to be created by the user with the facilities of package Text (EST) and then later read by the I/O packages discussed in this section.

Files thus provide the conventional notion of file storage. When a file is modified using the Rational Editor, changes to the file are not preserved until the file is committed. When a file is modified from a program, the updated value of the object is committed only when the file is closed. Thus, if a program does not explicitly close a file, the permanent contents of the file are unchanged by the execution of the program. This may be the intended result, but caution is warranted, especially in error situations in which exception handlers must determine whether to save the contents of a file by closing it. Not closing the file effectively abandons the changes made by the program.

Ada units are generally created through normal program development using the resources of the Rational Editor. Since Ada units can be read as streams of characters, the I/O facilities discussed in this book can be used to read this image. Ada units cannot be written directly. However, facilities exist in the Rational Environment for transforming a file into an Ada unit (see, for example, LM, procedure Compilation.Parse).

Files and Ada units are subject to the standard read/write synchronization protocols used throughout the Rational Environment. This synchronization permits multiple jobs to have simultaneous access to the same file for reading, but it allows only a single job to write to a file at any instant in time (with no readers allowed while a writer has the file open). Attempting to gain access to an object in a manner that violates this protocol results in the `Io_Exceptions.Use_Error` exception being raised. Attempting to open an Ada unit for writing also results in the `Use_Error` exception being raised.

## Devices and Windows

The Rational Environment supports I/O to or from several *devices*, including windows and terminals. In general, all of the I/O packages documented in this book can use any of these devices. I/O to tapes is permitted and is provided by package Tape (SMU). The exact effect of I/O with a particular device is, of course, unique to that device and is explained in the following paragraphs.

For each user session, one or more windows can be created to provide a medium for the files `Standard_Input` and `Standard_Output`, as defined in packages `Text_Io` and `Io`. Multiple windows are created when more than one job is simultaneously performing output. These windows are given names corresponding to the name of the job that is currently accessing them, or that accessed them most recently, and is of Text type. This window can be moved or expanded, and its contents can

be cut, copied, and otherwise manipulated with the Rational Editor commands, as explained in *Editing Images (EI)* and *Editing Specific Types (EST)*, package *Text*, in the *Rational Environment Reference Manual*. In general, this window automatically pops up when I/O is requested of the standard files. Output to `Standard_Output` appears in the window as characters (with control characters highlighted in a special font). Input requested from `Standard_Input` is denoted with the typical editor prompt, with the name *[input]*. The usual editing paradigm offered by the Rational Editor applies, so input can be typed ahead, edited, and even copied from other windows. Input is not sent to the waiting program until it is committed.

Session I/O windows optionally accumulate all I/O to standard files during one session, so the windows can be used to keep scripts of program interaction. Between jobs, I/O to these windows is separated by a job separator. The files `Standard_Input` and `Standard_Output` are automatically created at the start of each job and are automatically closed at the end of each job. If more than one job is initiated in a single session, and each job uses the resources of `Standard_Input` or `Standard_Output`, additional windows are created as necessary. If a job is executed and a session does not exist, `Standard_Input` and `Standard_Output` map to files with the names `Standard_Input` and `Standard_Output`, respectively. These files are created in the default context of the job that initiated the I/O.

Package *Io* introduces the notion of a `Standard_Error` file. This file maps to the Rational Editor Message window, so it typically is used to provide a common error-reporting mechanism among tools. If a job is executed and a session does not exist, `Standard_Error` maps to a file with the name `Standard_Error`, created in the default context of the job that initiated the I/O.

A programmatic interface for performing I/O to windows is provided with package `Window_Io` (*DIO*). In this case, the file abstraction is associated with an image. This image is displayed in a window on the terminal screen. Interfaces are provided to open images, put characters to any part of the image, get characters from the image, and close or delete the image when finished.

I/O also can be initiated directly to terminals using any of the I/O packages. Files can be opened using a name (of *String* type) in the form `!Machine.Devices.Terminal_n`, where *n* is an integer corresponding to a physical port on the processor. The exact names available on any particular machine can be found in `!Machine.Devices`. Once the file is opened, I/O can proceed as with any other file. Of course, the effect of the I/O depends on the nature of the physical device attached to the port. Physical devices other than the Rational Terminal can be attached to any port. If a program attempts to open a terminal that is already assigned to a job, the `Io_Exceptions.Use_Error` exception is raised.

Note that other lower-level operations for performing I/O on terminals that are logged in are available from package `!Io.Device_Independent_Io`, using the operations in package `!Io.Terminal_Specific`. They are not documented in the *Rational Environment Reference Manual*.

## Key Concepts

### File Handles

*File handles* are used for performing operations on files within Ada programs using the facilities provided by the I/O packages. When a file is opened or created, a file handle is returned. This handle is then used to refer to the file when calling the subprograms in the I/O packages.

The following is an example of a program that reads the lines from a text file named !Users.Blb.A\_Text\_File and displays them in the output window. The program first opens the file, which returns a file handle. This file handle is then used for reading the lines from the file and checking for an end of file condition.

```
with Io;
procedure Display_File is

    -- This program reads lines from a text file and displays them
    -- in the output window.

    File_Handle : Io.File_Type;
    -- This is the object that will contain the file handle.

begin

    Io.Open (File => File_Handle,
             Mode => Io.In_File,
             Name => "!users.blb.a_text_file");
    -- Opens the named file for reading and returns a file handle for
    -- performing I/O operations on that file within this program.

    while not Io.End_Of_File (File_Handle) loop
        declare
            Line : constant String := Io.Get_Line (File_Handle);
            -- Reads a line from the file.
        begin
            Io.Put_Line (Line);
            -- Writes the line to the output window (Standard_Output).
        end;
    end loop;

end Display_File;
```

### Filenames

Filenames supplied to the Create and Open procedures in the various I/O packages can be any legal Environment object name that uniquely identifies an object. Such names, for example, can contain wildcards and so on as long as the name can be resolved to a single object. Note that special names (for example, "<SELECTION>") can also be used to designate the name of a file. For more information on naming objects, see SJM, Key Concepts.

## Access Control

The Rational Environment provides access-control mechanisms that can be used to restrict the access that users and programs have to the objects in the library system. The operations provided in the I/O packages are subject to these access controls.

The access specified in Table 1-1 is required for performing I/O operations on files. If the required access does not exist, the `Io_Exceptions.Use_Error` exception will be raised by the attempted operation. See LM, Key Concepts, for more information on access control.

Table 1-1. Access Required for I/O Operations.

Operation	Access Required
All operations	Read access for all worlds enclosing the file
Creating a file	Create access to the world in which the file is to be created
Deleting a file	Read access to the file
Opening a file for reading (mode In)	Read access to the file
Opening a file for writing (mode Out)	Write access to the file

## Concurrency

The execution of any command or subprogram in the Rational Environment constitutes a *job*. Within a job, there may be several tasks that use I/O resources. If multiple tasks all share that same file handle, I/O may be arbitrarily interleaved and the results can be unpredictable. Thus, the I/O resources documented in this book may not offer or imply synchronization of the I/O activity. The Rational Environment does provide synchronization of I/O among different jobs, as discussed in "Devices and Windows," above.

## Representations of Terminators

Since packages `Text_Io` and `Io` observe the abstraction required by the *Reference Manual for the Ada Programming Language* of files containing line, page, and file terminators, it is sometimes useful to permit the user to simulate these terminators when creating or reading text files from programs. In the Environment, the line terminator is denoted by the character `Ascii.Lf`, the page terminator is denoted by the character `Ascii.Ff`, and the end-of-file terminator is implicit at the end of the file. A line terminator directly followed by a page terminator is compressed to the single character `Ascii.Ff`. Also, the line and page terminators preceding the file terminator are implicit and do not appear as characters in the file. For the sake of portability, programs should not depend on this representation, although it can be necessary to use this representation when importing text files from another system or exporting text files from the Rational Environment.

### Exceptions

Note that although most of the I/O packages contain renaming declarations for the exceptions defined in package `Io_Exceptions`, descriptions of these renaming declarations are omitted from the packages. Refer to the descriptions of the exceptions in the reference entries for package `Io_Exceptions`.

The Rational Environment provides additional information about exceptions raised by the I/O packages. This information, which describes why a given exception occurred, is typically displayed in parentheses after the exception name. See the reference entries for the exceptions in package `Io_Exceptions` for descriptions of this additional information.

### Error Reactions

When errors are discovered in a command, the command can respond by:

- Ignoring the error and trying to continue
- Issuing a warning message and trying to continue
- Raising an exception and abandoning the operation

For each job, the Environment maintains a default action for commands in package `Profile (SJM)` to take if an error occurs. There are commands for specifying and displaying the default error reaction for a job. Regardless of the default error reaction, any error reaction can be specified for any command.

The Environment has three default specifications for the profile it should use when responding to errors in a command. These are "`<PROFILE>`", "`<SESSION>`", and "`<DEFAULT>`", which refer, respectively, to the job response profile, the session response profile, and the default profile returned by the `Profile.Default_Profile` function.

## package Io

This package provides a superset of the capabilities required for Text\_Io in the *Reference Manual for the Ada Programming Language*, Chapter 14. In addition to the nested generic packages that deal with I/O for arbitrary discrete or real types, package Io provides virtually preinstantiated operations for the Standard.Integer, Standard.Float, and Standard.Boolean types.

The fundamental abstraction provided by package Io is the File\_Type type. Objects of this type denote file handles that can be mapped to external files. Each file is read or written sequentially, as a sequence of characters grouped into lines and a sequence of lines grouped into pages. Conversion operations are provided to convert Text\_Io file handles to Io file handles and vice versa (as well as Device-Independent-Io files).

At the beginning of program execution, the default input and output files are the *standard input file* and *standard output file*. These files are open, have the In\_File and Out\_File modes, respectively, and are associated with two implementation-defined external files. These files are implicitly closed at the end of each job. Package Io also introduces the notion of the *standard error file*, which follows the same semantics as the standard output file, except that it maps to the Message window by default.

From a logical point of view, a *text file* is a sequence of pages, a *page* is a sequence of lines, and a *line* is a sequence of characters. The characters of a line are numbered, starting from 1; the number of a character is called its *column number*. For a line terminator, a column number is also defined; it is one more than the number of characters in the line. The lines of a page, and the pages of a file, are similarly numbered. The current column number is the column number of the next character or line terminator to be transferred. The current line number is the number of the current line. The current page number is the number of the current page. These numbers are values of the Positive\_Count subtype of the Count subtype (by convention, the value 0 of the Count subtype is used to indicate special conditions).

For an output file, a maximum line length and a maximum page length can be specified. If a value to be output cannot fit on the current line, for a specified maximum line length, then a new line is automatically started before the value is output. Further, if this new line cannot fit on the current page, for a specified maximum page length, then a new page is automatically started before the value is output. Functions are provided to determine the maximum line length and the maximum

page length. When a file is opened with the Out\_File mode, both values are 0; by convention, this means that the line lengths and page lengths are unbounded. (Consequently, output consists of a single line if the subprograms for explicit control of line and page structure are not used.) The Unbounded constant is provided for this purpose.

Package Io provides a stack mechanism for managing current I/O files. This is the same mechanism that is accessible with the operations in package Log (SJM). It can be used, for example, in managing output or log files and the like when nested calls to output or logging procedures need to be coordinated. This mechanism allows all I/O to be performed on the current input, output, and error files. It provides push operations (the Set\_Input, Set\_Output, and Set\_Error procedures) that push files on the stack and make them the current input, output, and error files. Pop and reset operations are also provided. Pop removes a file from the stack. Reset is equivalent to performing a close operation and then a pop of the top element of the stack. Any files on the stack when a job terminates are automatically closed.

The package also provides a generic iterator, the Wildcard\_Iterator procedure, that can be used to perform a generic operation on a set of files matching a wildcard.



## function "="

---

```
function "=" (L, R : Count) return Boolean renames Text_Io."=";  
function "=" (L, R : File_Mode) return Boolean renames Text_Io."=";  
function "=" (L, R : Type_Set) return Boolean renames Text_Io."=";
```

---

### **Description**

Gives visibility to the Text\_Io equality operators.

---

```
function "<"  
package !Io.Io
```

## function "<"

---

```
function "<" (L, R : Count) return Boolean renames Text_Io."<";
```

---

### **Description**

Gives visibility to the Text\_Io comparison operator.

---

## function ">"

---

function ">" (L, R : Count) return Boolean renames Text\_Io.">";

---

### **Description**

Gives visibility to the Text\_Io comparison operator.

---

```
procedure Append
package !Io.Io
```

## procedure Append

---

```
procedure Append (File : in out File_Type;
                  Name : String;
                  Form : String := "");
procedure Append (File : in out File_Type;
                  Object : Directory.Version;
                  Form : String := "");
```

---

### Description

Opens the specified file for writing at the end of the file.

This procedure associates the specified file with an existing file having the specified name and form; if a file does not exist, it creates one. The current mode of the file is set to `Out_File`.

Output starting after an `Append` procedure will begin on a new line but on the same page as the previous end of the file.

The specified file is left open. After the file is opened, the page and line lengths are unbounded, and the current line and current page numbers are set to the current size of the file.

---

### Parameters

File : in out File\_Type;

Specifies the handle for the file.

Name : String;

Specifies the name of the external file to be appended.

Object : Directory.Version;

Specifies the object that is the external file to be appended.

Form : String := "";

Currently, the `Form` parameter, if specified, has no effect.

---

**Errors**

If the file handle is already open, the `Io_Exceptions.Status_Error` exception is raised.

If the string specified in the `Name` parameter does not allow the unique identification of a file, the `Io_Exceptions.Name_Error` exception is raised. In particular, this exception is raised if no file with the specified name exists.

The `Io_Exceptions.Use_Error` exception is raised when an attempt is made to perform an `Append` operation on objects on which the `Out_File` mode is not supported or if the file is locked by another job.

---

procedure Close  
package !Io.Io

## procedure Close

---

procedure Close (File : in out File\_Type);

---

### Description

Severs the association between the file handle and its associated file.

If the file has the current `Out_File` mode, it has the effect of calling the `New_Page` procedure. If the current page is already terminated, the procedure outputs a file terminator.

---

### Parameters

File : in out File\_Type;

Specifies the handle for the file.

---

### Errors

If the specified file is not open, the `Io_Exceptions.Status_Error` exception is raised.

---

### References

procedure `New_Page`

---

## function Col

---

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

---

### Description

Returns the current column number.

If a File parameter is omitted, the default file is the current output file.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Positive\_Count;  
Returns the current column number of the specified file.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the Positive\_Count value exceeds Count'Last, the Io\_Exceptions.Layout\_Error exception is raised.

---

function Convert  
package !Io.Io

## function Convert

---

```
function Convert (File: File_Type) return Text_Io.File_Type;  
function Convert (File: Text_Io.File_Type) return File_Type;
```

---

### **Description**

Converts a file handle of Io.File\_Type to Text\_Io.File\_Type and vice versa.  
When one type is converted to another, all file attributes are maintained.

---

### **Parameters**

File : File\_Type;  
Specifies the file handle to be converted.

return Text\_Io.File\_Type;  
Returns the file handle resulting from the conversion.

File : Text\_Io.File\_Type;  
Specifies the file handle to be converted.

return File\_Type;  
Returns the file handle resulting from the conversion.

---



## function Convert

---

function Convert (File: File\_Type) return Device\_Independent\_Io.File\_Type;

function Convert (File: Device\_Independent\_Io.File\_Type) return File\_Type;

---

### Description

Converts a file handle of Io.File\_Type to !Io.Device\_Independent\_Io.File\_Type and vice versa to allow access to device-specific options when the file is opened.

---

### Parameters

File : File\_Type;

Specifies the file handle to be converted.

return Device\_Independent\_Io.File\_Type;

Returns the file handle resulting from the conversion.

File : Device\_Independent\_Io.File\_Type;

Specifies the file handle to be converted.

return File\_Type;

Returns the file handle resulting from the conversion.

---

### Restrictions

Interchange of get/put and read/write operations between packages Io and !Io.Device\_Independent\_Io is undefined because of internal buffering in package Io. Use the Flush procedure to force the buffers to be written between these usages.

---

### References

procedure Flush

---

procedure Convert  
package !Io.Io

## procedure Convert

---

```
procedure Convert (From : Device_Independent_Io.File_Type;  
                  To   : in out Text_Io.File_Type);
```

---

### Description

Converts a file handle of !Io.Device\_Independent\_Io.File\_Type to Text\_Io.File\_Type.

This procedure can be used, for example, to save file handles in variables of Text\_Io.File\_Type type, which are limited private.

---

### Parameters

From : Device\_Independent\_Io.File\_Type;  
Specifies the file handle to be converted.

To : in out Text\_Io.File\_Type;  
Specifies the file handle that results from the conversion.

---

## subtype Count

---

subtype Count is Text\_lo.Count;

---

### **Description**

Specifies the range of possible values of the line and page count and the line and page length.

---

procedure Create  
package !Io.Io

## procedure Create

---

```
procedure Create (File : in out File_Type;  
                 Mode :           File_Mode := Out_File;  
                 Name :           String   := "";  
                 Form :           String   := "");
```

---

### Description

Establishes a new file with the specified name and associates this file with the specified file handle.

The specified file is left open.

---

### Parameters

File : in out File\_Type;

Specifies the handle for the file.

Mode : File\_Mode := Out\_File;

Specifies the access mode for which the file is to be used.

Name : String := "";

Specifies the name of the file to be created. A null string for the Name parameter specifies a file that is not accessible after the completion of the main program (a temporary file).

Form : String := "";

Currently, the Form parameter, if specified, has no effect.

---

### Restrictions

Files can be created only in directories or worlds.

---

## Errors

If the specified file handle is already open, the `Io_Exceptions.Status_Error` exception is raised.

The `Io_Exceptions.Name_Error` exception is raised under any of the following conditions:

- The filename does not conform to the syntax of a legal filename.
- An object of a nonfile class with the same name as the filename already exists in the context in which the creation is attempted.
- The context in which the creation is attempted cannot contain files. Files are allowed only in directories or worlds.

The `Io_Exceptions.Use_Error` exception will be raised under any of the following conditions:

- The file cannot be opened with the specified mode.
  - The executing job does not have create access.
  - Another job has locked the file.
-

function Current\_Error  
package !Io.Io

## function Current\_Error

---

```
function Current_Error return File_Type;  
function Current_Error return Text_Io.File_Type;
```

---

### **Description**

Returns the handle to the current default error file.

---

## function Current\_Input

---

function Current\_Input return File\_Type;

---

### **Description**

Returns the handle to the current default input file.

---

function Current\_Output  
package !Io.Io

## function Current\_Output

---

function Current\_Output return File\_Type;

---

### **Description**

Returns the handle to the current default output file.

---



## procedure Delete

---

procedure Delete (File : in out File\_Type);

---

### Description

Deletes the file associated with the specified file handle.

The file handle is closed, and the file ceases to exist.

---

### Parameters

File : in out File\_Type;  
Specifies the handle for the file.

---

### Errors

If the specified file handle is not open, the `Io_Exceptions.Status_Error` exception is raised.

The `Io_Exceptions.Use_Error` exception is raised under any of the following conditions:

- The Environment does not support deletion on the file.
  - The executing job does not have the access rights required to delete the file.
  - Another job has locked the file.
-

procedure Echo  
package !Io.Io

## procedure Echo

---

procedure Echo (Item : Character);

---

### **Description**

Writes a character to the current error file (by default, the Message window).

If the line length of the specified output file is bounded (that is, does not have the conventional value of 0) and the current column number exceeds it, this procedure has the effect of calling the `New_Line` procedure with a spacing of 1 for the current error file. Then the procedure outputs the specified character to the file.

---

### **Parameters**

Item : Character;

Specifies the value of the item to be written.

---

## procedure Echo

---

```
procedure Echo (Item : String := "");
```

---

### Description

Writes a string to the current error file (by default, the Message window).

This procedure determines the length of the specified string and attempts that number of echo operations for successive characters of the string. No operation is performed if the string is null.

---

### Parameters

Item : String := "";

Specifies the value of the item to be written.

---

```
procedure Echo
package !Io.Io
```

## procedure Echo

---

```
procedure Echo (Item : Integer;
               Width : Field      := 0;
               Base  : Number_Base := 10);
```

---

### Description

Writes an integer value to the current error file (by default, the Message window).

Values are output as decimal or based literals, without underline characters or exponents, and are preceded by a minus sign if negative. The format (which includes any leading spaces and a minus sign) can be specified by an optional field Width parameter. Values of widths of fields in output formats are of the nonnegative integer Field subtype. Values of bases are of the integer Number\_Base subtype.

This procedure outputs the value of the Item parameter as an integer literal, with no underlines, no exponent, and no leading zeros (but a single zero for the value 0), and with a preceding minus sign for a negative value.

If the resulting sequence of characters to be output has fewer characters than specified in the Width parameter, leading spaces are output to make up the difference.

The procedure uses the syntax for decimal literal if the Base parameter has the value 10; otherwise, it uses the syntax for based literal, with any letters in uppercase.

---

### Parameters

Item : Integer;

Specifies the value to be written.

Width : Field := 0;

Specifies the number of characters to be output.

Base : Number\_Base := 10;

Specifies the radix base with which to output the form of the number.

---

## procedure Echo

---

```
procedure Echo (Item : Float;  
               Fore : Field := 2;  
               Aft  : Field := 14;  
               Exp  : Field := 3);
```

---

### Description

Writes a floating-point value to the current error file (by default, the Message window).

Values are output as decimal literals without underline characters. The format of each value output consists of a Fore field, a decimal point, an Aft field, and (if a nonzero Exp parameter is supplied) the letter E and an Exp field. The two possible formats thus correspond to:

Fore . Aft

and:

Fore . Aft E Exp

with no spaces between these fields. The Fore field can include leading spaces and a minus sign for negative values. The Aft field includes only decimal digits (possibly with trailing zeros). The Exp field includes the sign (plus or minus) and the exponent (possibly with leading zeros).

The Echo procedure outputs the value of the Item parameter as a decimal literal with the format defined by the Fore, Aft, and Exp parameters. If the value is negative, a minus sign is included in the integer part. If Exp has the value 0, then the integer part to be output has as many digits as needed to represent the integer part of the value of the Item parameter, overriding Fore if necessary, or it consists of the digit 0 if the value of Item has no integer part.

If Exp has a value greater than 0, the integer part to be output has a single digit, which is nonzero except for the value 0.0 of the Item parameter.

In both cases, however, if the integer part to be output has fewer than Fore characters, including any minus sign, leading spaces are output to make up the difference. The number of digits of the fractional part is specified by Aft, or it is 1 if Aft equals 0. The value is rounded; a value of exactly one-half in the last place can be rounded either up or down.

procedure Echo  
package !Io.Io

If Exp has the value 0, there is no exponent part. If Exp has a value greater than 0, the exponent part to be output has as many digits as needed to represent the exponent part of the value of the Item parameter (for which a single-digit integer part is used), and it includes an initial sign (plus or minus). If the exponent part to be output has fewer than Exp characters, including the sign, leading zeros precede the digits to make up the difference. For the value 0.0 of the Item parameter, the exponent has the value 0.

---

### **Parameters**

Item : Float;

Specifies the value to be written.

Fore : Field := 2;

Specifies the number of characters to be output before the decimal point.

Aft : Field := 14;

Specifies the number of characters to be output after the decimal point.

Exp : Field := 3;

Specifies the number of exponent characters to be output.

---

## procedure Echo

---

```
procedure Echo (Item : Boolean;  
              Width : Field := 0);
```

---

### Description

Writes the Boolean value to the current error file (by default, the Message window).

Values are output using either uppercase or lowercase letters for identifiers.

The format (which includes any trailing spaces) can be specified by an optional field Width parameter.

The procedure outputs the value of the Item parameter as an enumeration literal. If the sequence of characters produced has fewer than the characters specified by the Width parameter, trailing spaces are output to make up the difference.

---

### Parameters

Item : Boolean;

Specifies the value to be written.

Width : Field := 0;

Specifies the number of characters to be written.

---

```
procedure Echo_Line  
package !Io.Io
```

## procedure Echo\_Line

---

```
procedure Echo_Line (Item : String := "");
```

---

### **Description**

Writes a string to the current error file (by default, the Message window) and advances the line.

This procedure calls the Put procedure for the specified string and then calls the New\_Line procedure with a spacing of 1 on the current error file.

---

### **Parameters**

```
Item : String := "";
```

Specifies the value of the string to be written.

---



## function End\_Of\_File

---

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

---

### Description

Returns true if a file terminator or the combination of a line, a page, and a file terminator is the next item to be read from the file; otherwise, the function returns false.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Boolean;

Returns true if a file terminator or the combination of a line, a page, and a file terminator is the next item to be read from the file; otherwise, the function returns false.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

```
function End_Of_Line
package !Io.Io
```

## function End\_Of\_Line

---

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

---

### Description

Returns true if a line terminator or a file terminator is the next item to be read from the file; otherwise, the function returns false.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

return Boolean;

Returns true if a line terminator or a file terminator is the next item to be read from the file; otherwise, the function returns false.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

## function End\_Of\_Page

---

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

---

### Description

Returns true if a file terminator or the combination of a line and a page terminator is the next item to be read from the file; otherwise, the function returns false.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Boolean;

Returns true if a file terminator or the combination of a line and a page terminator is the next item to be read from the file; otherwise, the function returns false.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

subtype Field  
package !Io.Io

## subtype Field

---

subtype Field is Integer range 0 .. Integer'Last;

---

### **Description**

Specifies the range of possible values for the number of character positions used in formatting strings that represent discrete or real values.

---

## subtype File\_Mode

---

subtype File\_Mode is Text\_io.File\_Mode;

---

### **Description**

Specifies the mode of access for which a file is open.

In\_File mode denotes a file with read-only access; Out\_File mode denotes a file with write-only access.

---

type File\_Type  
package !Io.Io

## type File\_Type

---

type File\_Type is private;

---

### **Description**

Defines a file handle type for files to be processed by operations in this package.

---

## procedure Flush

---

```
procedure Flush (File : File_Type);
```

---

### **Description**

Causes any characters currently in internal buffers and not yet in the file to be forced out to the file.

The procedure can be used to force logging output from programs at key points out to files so that the intermediate contents of the log can be viewed by the Rational Editor.

---

### **Parameters**

File : File\_Type;  
Specifies the file to be forced.

---

### **References**

procedure Save

---

function Form  
package !Io.Io

## function Form

---

function Form (File : File\_Type) return String;

---

### **Description**

Currently returns the null string ("") in all cases.

When the Form parameter to the Create and Open procedures is supported in the future, this function will return the Form value provided in the call to the Create or the Open procedure.

---

### **Parameters**

File : File\_Type;

Specifies the handle for the file.

return String;

Currently returns the null string ("") in all cases.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

---

### **References**

procedure Create

procedure Open

---



## procedure Get

---

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

---

### Description

Reads a character from a file.

After skipping any line and page terminators, this procedure reads the next character from the specified input file and returns the value of this character in the Item parameter.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : out Character;  
Specifies the object that receives the value read.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not In\_File, the Io\_Exceptions.Mode\_Error exception is raised.

If an attempt is made to skip a file terminator, the Io\_Exceptions.End\_Error exception is raised.

---

## procedure Get

---

```
procedure Get (File : File_Type;  
              Item : out String);  
  
procedure Get (Item : out String);
```

---

### Description

Reads a string from a file.

This procedure determines the length of the specified string and attempts that number of get operations for successive characters of the string. No operation is performed if the string is null.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : out String;

Specifies the object that receives the value read.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

---

## procedure Get

---

```
procedure Get (File      : File_Type;  
              Item      : out String;  
              Last      : out Natural;  
              End_Of_Line : out Boolean;  
              End_Of_Page : out Boolean;  
              End_Of_File : out Boolean);
```

---

### Description

Reads part or all of a line from a file.

This procedure determines the length of the specified string and attempts that number of get operations for successive characters of the string up to the end of the current line, not including terminators (no operation is performed if the string is null). The index of the last character read is returned in the Last parameter. If no characters are read, the Last parameter contains a value one less than Item'First.

The procedure also returns an indication of whether or not various terminators are encountered.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : out String;

Specifies the object that receives the value read.

Last : out Natural;

Returns the index of the last character read. If no characters are read, the parameter returns a value one less than Item'First.

End\_Of\_Line : out Boolean;

Returns true if and only if the Item parameter contains the end-of-line terminator (possibly null).

End\_Of\_Page : out Boolean;

Returns true if and only if the value returned by End\_Of\_Line is true and this is the last line of the page.

procedure Get  
package !Io.Io

End\_Of\_File : out Boolean;

Returns true if and only if the value returned by End\_Of\_Page is true and this is the last page of the file.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not In\_File, the Io\_Exceptions.Mode\_Error exception is raised.

If an attempt is made to skip a file terminator, the Io\_Exceptions.End\_Error exception is raised.

---

## procedure Get

---

```
procedure Get (File : File_Type;  
              Item : out Integer;  
              Width : Field := 0);  
  
procedure Get (Item : out Integer;  
              Width : Field := 0);
```

---

### Description

Reads an integer value from a file.

If the value of the Width parameter is 0, the Get procedure skips any leading blanks, line terminators, or page terminators, reads a plus or a minus sign if present, and then reads according to the syntax of an integer literal (which may be a based literal). If a nonzero value of Width is supplied, then exactly Width characters or the characters (possibly none) up to a line terminator are input, whichever comes first; any skipped leading blanks are included in the count.

The procedure returns, in the Item parameter, the value of the Integer type that corresponds to the sequence input.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : out Integer;

Specifies the object that receives the value read.

Width : Field := 0;

Specifies the number of characters to be read.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the input sequence does not have the required syntax or if the value obtained is not of the `Integer` type, the `Io_Exceptions.Data_Error` exception is raised. When a sign is input, this rule applies to the succeeding numeric literal.

---

## procedure Get

---

```
procedure Get (File : File_Type;  
              Item : out Float;  
              Width : Field := 0);  
  
procedure Get (Item : out Float;  
              Width : Field := 0);
```

---

### Description

Reads a floating-point number from a file.

If the value of the Width parameter is 0, the Get procedure skips any leading blanks, line terminators, or page terminators, reads a plus or a minus sign if present, and then reads according to the syntax of a real literal (which may be a based literal). If a nonzero value of Width is supplied, then exactly Width characters or the characters (possibly none) up to a line terminator are input, whichever comes first; any skipped leading blanks are included in the count.

The procedure returns, in the Item parameter, the value of Float type that corresponds to the sequence input.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : out Float;

Specifies the object that receives the value read.

Width : Field := 0;

Specifies the number of characters to be read.

---

**Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the sequence input does not have the required syntax or if the value obtained is not of the `Float` type, the `Io_Exceptions.Data_Error` exception is raised. For this test, leading blanks are ignored. When a sign is input, this rule applies to the succeeding numeric literal.

---



## procedure Get

---

```
procedure Get (File : File_Type;  
              Item : out Boolean);  
  
procedure Get (Item : out Boolean);
```

---

### Description

Reads the Boolean value from a file.

After skipping any leading blanks, line terminators, or page terminators, the Get procedure reads a Boolean literal in either lowercase or uppercase.

The procedure returns, in the Item parameter, the value of the Boolean type that corresponds to the sequence input.

If a File parameter is omitted, the current default file is read.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : out Boolean;  
Specifies the object that receives the value read.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the sequence input does not have the required syntax or if the identifier or character literal does not correspond to a value of the Boolean type, the `Io_Exceptions.Data_Error` exception is raised.

---

```
function Get_Line
package !Io.Io
```

## function Get\_Line

---

```
function Get_Line (File : File_Type) return String;
function Get_Line return String;
```

---

### Description

Returns all remaining characters on the current line except for the line terminator.  
If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

return String;

Returns all remaining characters on the current line except for the line terminator.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not In\_File, the Io\_Exceptions.Mode\_Error exception is raised.

If an attempt is made to skip a file terminator, the Io\_Exceptions.End\_Error exception is raised.

---

## procedure Get\_Line

---

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

---

### Description

Reads a string on a single line from a file, not including the line terminator.

This procedure replaces successive characters of the specified string by successive characters read from the specified input file. Reading stops if the end of the line is encountered, in which case the Skip\_Line procedure is called (in effect) with a spacing of 1; reading also stops if the end of the string is encountered. Characters not replaced are left undefined.

If characters are read, the Last parameter contains the index value such that Item(Last) is the last character replaced (the index of the first character replaced is Item'First). If no characters are read, the Last parameter contains an index value that is one less than Item'First.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : out String;  
Specifies the object that receives the value read.

Last : out Natural;  
Specifies the index for the last character read into the string.

procedure Get\_Line  
package !Io.Io

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

---

### References

procedure Skip\_Line

---

## constant In\_File

---

In\_File : constant File\_Mode := Text\_io.In\_File;

---

### **Description**

Defines a named constant for the In\_File mode.

---

```
function Is_Open  
package !Io.Io
```

## function Is\_Open

---

```
function Is_Open (File : File_Type) return Boolean;
```

---

### **Description**

Returns true if the file handle is open (that is, if it is associated with a file); otherwise, the function returns false.

---

### **Parameters**

File : File\_Type;

Specifies the handle for the file.

return Boolean;

Returns true if the file handle is open (that is, if it is associated with a file); otherwise, the function returns false.

---

## function Line

---

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

---

### Description

Returns the current line number.

If a File parameter is omitted, the default file is the current output file.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Positive\_Count;  
Returns the current line number.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the `Positive_Count` value exceeds `Count'Last`, the `Io_Exceptions.Layout_Error` exception is raised.

---

```
function Line_Length
package !Io.Io
```

## function Line\_Length

---

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

---

### Description

Returns the maximum line length currently set for the specified output file; returns 0 if the line length is unbounded.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Count;

Returns the maximum line length currently set for the specified output file; returns 0 if the line length is unbounded.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---



## constant Lower\_Case

---

Lower\_Case : constant Type\_Set := Text\_Io.Lower\_Case;

---

### **Description**

Defines a value indicating that enumeration literals are to be displayed in lowercase.

---

function Mode  
package !Io.Io

## function Mode

---

```
function Mode (File : File_Type) return File_Mode;
```

---

### **Description**

Returns the current mode of the specified file.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

return File\_Mode;  
Returns the current mode of the specified file.

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

---

## function Name

---

function Name (File : File\_Type) return String;

---

### Description

Returns the name of the file currently associated with the specified file handle.

For temporary files, the function returns the unique name provided by the Rational Environment during the creation of the file.

---

### Parameters

File : File\_Type ;

Specifies the handle for the file.

return String;

Returns the name of the file currently associated with the specified file handle.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

---

```
procedure New_Line
package !Io.Io
```

## procedure New\_Line

---

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

---

### Description

Adds the specified number of line terminators to the current file, forcing subsequent output to the following line.

For a spacing of 1, the procedure outputs a line terminator, sets the current column number to 1, and then adds 1 to the current line number. If the current line number is already greater than or equal to the maximum page length for a bounded page length, a page terminator is output, the current page number is increased by 1, and the current line number is set to 1.

For a spacing greater than 1, the above actions are performed the number of times specified by the Spacing parameter.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Spacing : Positive\_Count := 1;  
Specifies the number of new lines to be added.

---

### Errors

If the specified file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

## procedure New\_Page

---

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

---

### Description

Outputs a page terminator.

The procedure outputs a line terminator if the current line is not terminated or if the current page is empty (that is, if the current column and line numbers are both equal to 1). It then outputs a page terminator, which terminates the current page, adds 1 to the current page number, and sets the current column and line numbers to 1.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

subtype Number\_Base  
package !Io.Io

## subtype Number\_Base

---

subtype Number\_Base is Integer range 2 .. 16;

---

### **Description**

Specifies the range of possible values for the radix of numeric values to be written or read.

---

## procedure Open

---

```
procedure Open (File : in out File_Type;  
               Mode : File_Mode := Out_File;  
               Name : String;  
               Form : String := "");  
  
procedure Open (File : in out File_Type;  
               Mode : File_Mode;  
               Object : Directory.Version;  
               Form : String := "");
```

---

### Description

Associates the file handle with an existing file having the specified name or version and sets the mode of the file to the specified mode.

After a file is opened with the Out\_File mode, the page length and line length are unbounded. After a file is opened with the In\_File or Out\_File mode, the current column, current line, and current page numbers are set to 1.

---

### Parameters

File : in out File\_Type;  
Specifies the handle for the file.

Mode : File\_Mode := Out\_File;  
Specifies the access mode for which the file is to be used.

Mode : File\_Mode;  
Specifies the access mode for which the file is to be used; by default, it will be a file that is open for writing.

Name : String;  
Specifies the name of the external file to be opened.

Object : Directory.Version;  
Specifies the object that is the external file to be opened.

Form : String := "";  
Currently, the Form parameter, if specified, has no effect.

---

**Errors**

If the file handle is already open, the `Io_Exceptions.Status_Error` exception is raised.

If the string specified in the `Name` parameter does not allow unique identification of a file, the `Io_Exceptions.Name_Error` exception is raised. In particular, this exception is raised if no file with the specified name exists.

The `Io_Exceptions.Use_Error` exception is raised if the file cannot be opened with the specified mode or if another job has locked the file.

---



## constant Out\_File

---

Out\_File : constant File\_Mode := Text\_io.Out\_File;

---

### **Description**

Defines a named constant for the Out\_File mode.

---

function Page  
package !Io.Io

## function Page

---

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

---

### **Description**

Returns the current page number.

If a File parameter is omitted, the default file is the current output file.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

return Positive\_Count;  
Returns the current page number.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the Positive\_Count value exceeds Count'Last, the Io\_Exceptions.Layout\_Error exception is raised.

---

## function Page\_Length

---

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

---

### Description

Returns the maximum page length currently set for the specified output file; returns 0 if the page length is unbounded.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Count;

Returns the maximum page length currently set for the specified output file; returns 0 if the page length is unbounded.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

procedure Pop\_Error  
package !Io.Io

## procedure Pop\_Error

---

procedure Pop\_Error;

---

### **Description**

Pops the current error file off the stack of error files.

The procedure sets the current error file to be a file value previously saved by a Set\_Error procedure. The current value of the error file is not closed and will not be closed automatically at the end of the job because it is no longer on the stack.

---

### **References**

procedure Reset\_Error

procedure Set\_Error

---

## procedure Pop\_Input

---

procedure Pop\_Input;

---

### **Description**

Pops the current input file off the stack of input files.

The procedure sets the current input file to be a file value previously saved by a Set\_Input procedure. The current value of the input file is not closed and will not be closed automatically at the end of the job because it is no longer on the stack.

---

### **References**

procedure Reset\_Input

procedure Set\_Input

---

procedure Pop\_Output  
package !Io.Io

## procedure Pop\_Output

---

procedure Pop\_Output;

---

### **Description**

Pops the current output file off the stack of output files.

The procedure sets the current output file to be a file value previously saved by a Set\_Output procedure. The current value of the output file is not closed and will not be closed automatically at the end of the job because it is no longer on the stack.

---

### **References**

procedure Reset\_Output

procedure Set\_Output

---

## subtype Positive\_Count

---

subtype Positive\_Count is Count range 1 .. Count'Last;

---

### **Description**

Specifies the range of possible values for the number of lines to be skipped or inserted.

---

### **References**

procedure New\_Line

procedure Skip\_Line

---

procedure Put  
package !Io.Io

## procedure Put

---

```
procedure Put (File : File_Type;  
              Item : Character);  
  
procedure Put (Item : Character);
```

---

### Description

Writes a character to a file.

If the line length of the specified output file is bounded (that is, does not have the conventional value 0) and the current column number exceeds it, the procedure has the effect of calling the `New_Line` procedure with a spacing of 1. Then the procedure outputs the specified character to the file.

If a `File` parameter is omitted, the current default file is understood to be specified.

---

### Parameters

`File` : `File_Type`;  
Specifies the handle for the file.

`Item` : `Character`;  
Specifies the value of the item to be written.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

### References

procedure `New_Line`

---



## procedure Put

---

```
procedure Put (File : File_Type;  
              Item : String);  
  
procedure Put (Item : String);
```

---

### Description

Writes a string to a file.

Determines the length of the specified string and attempts that number of put operations for successive characters of the string. No operation is performed if the string is null.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : String;  
Specifies the value of the string to be written.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

## procedure Put

---

```
procedure Put (File : File_Type;  
              Item : Integer;  
              Width : Field      := 0;  
              Base  : Number_Base := 10);  
  
procedure Put (Item : Integer;  
              Width : Field      := 0;  
              Base  : Number_Base := 10);
```

---

### Description

Writes an integer value to a file.

Values are output as decimal or based literals, without underline characters or exponents, and are preceded by a minus sign if negative. The format (which includes any leading spaces and a minus sign) can be specified by an optional field Width parameter. Values of widths of fields in output formats are of the nonnegative integer Field subtype. Values of bases are of the integer Number\_Base subtype.

This procedure outputs the value of the Item parameter as an integer literal, with no underlines, no exponent, and no leading zeros (but a single zero for the value 0), and with a preceding minus sign for a negative value.

If the resulting sequence of characters to be output has fewer characters than specified in the Width parameter, leading spaces are output to make up the difference.

This procedure uses the syntax for decimal literal if the Base parameter has the value 10; otherwise, it uses the syntax for based literal, with any letters in uppercase.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : Integer;  
Specifies the value to be written.

Width : Field := 0;  
Specifies the number of characters to be output.

Base : Number\_Base := 10;

Specifies the radix base with which to output the form of the number.

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

procedure Put  
package !Io.Io

## procedure Put

---

```
procedure Put (File : File_Type;  
              Item : Float;  
              Fore : Field      := 2;  
              Aft  : Field      := 14;  
              Exp  : Field      := 3);
```

```
procedure Put (Item : Float;  
              Fore : Field := 2;  
              Aft  : Field := 14;  
              Exp  : Field := 3);
```

---

### Description

Writes a floating-point value to a file.

Values are output as decimal literals without underline characters. The format of each value output consists of a Fore field, a decimal point, an Aft field, and (if a nonzero Exp parameter is supplied) the letter E and an Exp field. The two possible formats thus correspond to:

Fore . Aft

and:

Fore . Aft E Exp

with no spaces between these fields. The Fore field can include leading spaces and a minus sign for negative values. The Aft field includes only decimal digits (possibly with trailing zeros). The Exp field includes the sign (plus or minus) and the exponent (possibly with leading zeros).

The Put procedure outputs the value of the Item parameter as a decimal literal with the format defined by the Fore, Aft, and Exp parameters. If the value is negative, a minus sign is included in the integer part. If Exp has the value 0, the integer part to be output has as many digits as needed to represent the integer part of the value of the Item parameter, overriding Fore if necessary, or it consists of the digit 0 if the value of Item has no integer part.

If Exp has a value greater than 0, the integer part to be output has a single digit, which is nonzero except for the value 0.0 of the Item parameter.

In both cases, however, if the integer part to be output has fewer than Fore characters, including any minus sign, leading spaces are output to make up the difference. The number of digits of the fractional part is specified by Aft, or it is 1 if Aft equals 0. The value is rounded; a value of exactly one-half in the last place can be rounded either up or down.

If Exp has the value 0, there is no exponent part. If Exp has a value greater than 0, the exponent part to be output has as many digits as needed to represent the exponent part of the value of the Item parameter (for which a single-digit integer part is used), and it includes an initial sign (plus or minus). If the exponent part to be output has fewer than Exp characters, including the sign, leading zeros precede the digits to make up the difference. For the value 0.0 of the Item parameter, the exponent has the value 0.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : Float;

Specifies the value to be written.

Fore : Field := 2;

Specifies the number of characters to be output before the decimal point.

Aft : Field := 14;

Specifies the number of characters to be output after the decimal point.

Exp : Field := 3;

Specifies the number of exponent characters to be output.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

```
procedure Put
package !Io.Io
```

## procedure Put

---

```
procedure Put (File : File_Type;
              Item : Boolean;
              Width : Field      := 0);

procedure Put (Item : Boolean;
              Width : Field      := 0);
```

---

### Description

Writes the Boolean value to a file.

Values are output using either uppercase or lowercase letters for identifiers.

The format (which includes any trailing spaces) can be specified by an optional field Width parameter.

The procedure outputs the value of the Item parameter as an enumeration literal. If the sequence of characters produced has fewer characters than specified by the Width parameter, trailing spaces are output to make up the difference.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : Boolean;

Specifies the value to be written.

Width : Field := 0;

Specifies the number of characters to be written.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

## procedure Put\_Line

---

```
procedure Put_Line (File : File_Type;  
                  Item : String);  
  
procedure Put_Line (Item : String);
```

---

### Description

Writes a string to a file and advances the line.

This procedure calls the Put procedure for the specified string and then calls the New\_Line procedure with a spacing of 1.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : String;  
Specifies the value of the string to be written.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

### References

procedure New\_Line

procedure Put

---

```
procedure Reset
package !Io.Io
```

## procedure Reset

---

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

---

### Description

Resets the specified file so that reading from or writing to its elements can be restarted from the beginning of the file.

If a Mode parameter is supplied, the mode of the specified file handle is set to the specified mode.

If the file has the current Out\_File mode, this procedure terminates the current page and outputs a file terminator.

---

### Parameters

File : in out File\_Type;  
Specifies the handle for the file.

Mode : File\_Mode;  
Specifies the mode for which the file is to be used when the reset is completed.

---

### Errors

If the file handle is not open, the Io\_Exceptions.Status\_Error exception is raised.

If an attempt is made to change the mode of a file that is either the current default input file or the current default output file, the Io\_Exceptions.Mode\_Error exception is raised.

The Io\_Exceptions.Use\_Error exception is raised under any of the following conditions:

- The Environment does not support resetting for the file.
  - The file cannot be reset to the specified mode.
  - Another job has locked the file.
-



## procedure Reset\_Error

---

procedure Reset\_Error;

---

### **Description**

Closes the current error file and pops it off the stack of error files.

The procedure sets the current error file to be a file value previously saved by a Set\_Error procedure. The current value of the error file is closed.

---

### **References**

procedure Pop\_Error

procedure Set\_Error

---

procedure Reset\_Input  
package !Io.Io

## procedure Reset\_Input

---

procedure Reset\_Input;

---

### **Description**

Closes the current input file and pops it off the stack of input files.

The procedure sets the current input file to be a file value previously saved by a Set\_Input procedure. The current value of the input file is closed.

---

### **References**

procedure Pop\_Input

procedure Set\_Input

---

## procedure Reset\_Output

---

procedure Reset\_Output;

---

### **Description**

Closes the current output file and pops it off the stack of output files.

The procedure sets the current output file to be a file value previously saved by a Set\_Output procedure. The current value of the output file is closed.

---

### **References**

procedure Pop\_Output

procedure Set\_Output

---

procedure Save  
package !Io.Io

## procedure Save

---

procedure Save (File : File\_Type);

---

### **Description**

Causes the current contents of the file to be permanently saved after any characters currently in internal buffers and not yet in the file are forced out to the file using the Flush procedure.

The procedure can be used to force logging output from programs at key points out to files so that the intermediate snapshots of the logs can be saved if the program terminates before the file is closed normally.

---

### **Parameters**

File : File\_Type;  
Specifies the file to be saved.

---

### **References**

procedure Flush

---

## procedure Set\_Col

---

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

```
procedure Set_Col (To : Positive_Count);
```

---

### **Description**

Sets the current column number of the specified file.

If the file mode is `Out_File`:

If the value specified by the `To` parameter is greater than the current column number, this procedure outputs spaces, adding 1 to the current column number after each space, until the current column number equals the specified value. If the value specified by `To` is equal to the current column number, there is no effect. If the value specified by `To` is less than the current column number, the procedure has the effect of calling the `New_Line` procedure (with a spacing of 1), and then it outputs  $(To - 1)$  spaces and sets the current column number to the specified value.

If the file mode is `In_File`:

The procedure reads (and discards) individual characters, line terminators, and page terminators until the next character to be read has a column number that equals the value specified by `To`; there is no effect if the current column number already equals this value. Each transfer of a character or terminator maintains the current column, line, and page numbers in the same way that a `Get` or a `Get_Line` procedure does.

If a `File` parameter is omitted, the default file is the current output file.

The column number, line number, and page number are allowed to exceed `Count'Last` (as a consequence of the input or output of sufficiently many characters, lines, or pages).

---

### **Parameters**

`File` : `File_Type`;

Specifies the handle for the file.

`To` : `Positive_Count`;

Specifies the value of the column.

procedure Set\_Col  
package !Io.Io

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the file mode is `Out_File`, the `Io_Exceptions.Layout_Error` exception is raised if the value specified by the `To` parameter exceeds `Line_Length` when the line length is bounded (that is, when it does not have the conventional value of 0).

If the file mode is `In_File`, the `Io_Exceptions.End_Error` exception is raised if an attempt is made to read a file terminator.

---

### References

procedure `Get`

procedure `Get_Line`

procedure `New_Line`

---

## procedure Set\_Error

---

```
procedure Set_Error (File : File_Type);  
procedure Set_Error (Name : String := ">>FILE NAME<<");
```

---

### Description

Sets the current default error file to the specified file handle or file and pushes the file onto the stack of error files.

If a filename is specified, the file is opened with `Out_File` mode before it is set to be the default output file. A file is created if one does not exist.

Any files on the stack when a job terminates are automatically closed.

---

### Parameters

`File` : `File_Type`;  
Specifies the handle for the file.

`Name` : `String` := ">>FILE NAME<<";  
Specifies the filename to open (and create if necessary) and set to be the current default output file.

procedure Set\_Error  
package !Io.Io

---

## Errors

If a file handle is specified and it is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

The `Io_Exceptions.Name_Error` exception is raised under the following conditions:

- The filename does not conform to the syntax of a name.
- An object of a nonfile class with the same name as the filename already exists in the context in which the creation is attempted.
- The context in which the creation is attempted cannot contain files. Files are allowed only in directories or worlds.

The `Io_Exceptions.Use_Error` exception is raised under the following conditions:

- The file cannot be opened with `Out_File` mode.
- A create is attempted and the executing job does not have create access.
- The file is locked by another job.

---

## References

procedure Pop\_Error

procedure Reset\_Error

---



## procedure Set\_Input

---

```
procedure Set_Input (File : File_Type);  
procedure Set_Input (Name : String := "<SELECTION>");
```

---

### **Description**

Sets the current default input file to the specified file handle or file and pushes the file onto the stack of input files.

If a filename is specified, the file is opened with In\_File mode before it is set to be the default input file. A file is created if one does not exist.

Any files on the stack when a job terminates are automatically closed.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

Name : String := "<SELECTION>";  
Specifies the file to open (and create if necessary) and set to be the current default input file. By default, the file designated by the current selection will be used.

procedure Set\_Input  
package !Io.Io

---

### **Errors**

If a file handle is specified and it is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

The `Io_Exceptions.Name_Error` exception is raised under the following conditions:

- The `Name` parameter does not allow unique identification of a file.
- An object of a nonfile class already exists with the same name.
- The context for creating the file is not a library.

The `Io_Exceptions.Use_Error` exception is raised under the following conditions:

- The file cannot be opened with the `In_File` mode.
  - A create is attempted and the executing job does not have create access.
  - Another job has locked the file.
- 

### **References**

procedure Pop\_Input

procedure Reset\_Input

---

## procedure Set\_Line

---

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

---

### Description

Sets the current line number of the file.

If the file mode is Out\_File:

If the value specified by the To parameter is greater than the current line number, the procedure has the effect of repeatedly calling the New\_Line procedure (with a spacing of 1) until the current line number equals the specified value. If the value specified by To is equal to the current line number, there is no effect. If the value specified by To is less than the current line number, the procedure has the effect of calling the New\_Page procedure followed by a call to the New\_Line procedure with a spacing equal to (To - 1).

If the file mode is In\_File:

The procedure has the effect of repeatedly calling the Skip\_Line procedure (with a spacing of 1) until the current line number equals the value specified by To; there is no effect if the current line number already equals this value. (Short pages will be skipped until a page is reached that has a line at the specified line position.)

If a File parameter is omitted, the default file is the current output file.

The column number, line number, and page number are allowed to exceed Count'Last (as a consequence of the input or output of sufficiently many characters, lines, or pages).

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

To : Positive\_Count;  
Specifies the value of the line.

procedure Set\_Line  
package !Io.Io

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the file mode is `Out_File`, the `Io_Exceptions.Layout_Error` exception is raised if the value specified by the `To` parameter exceeds `Page_Length` when the page length is bounded (that is, when it does not have the conventional value of 0).

If the file mode is `In_File`, the `Io_Exceptions.End_Error` exception is raised if an attempt is made to read a file terminator.

---

### References

procedure `New_Line`

procedure `New_Page`

procedure `Skip_Line`

---

## procedure Set\_Line\_Length

---

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

```
procedure Set_Line_Length (To : Count);
```

---

### Description

Sets the maximum line length of the specified output file to the number of characters specified by the `To` parameter.

If a `File` parameter is omitted, the current default file is understood to be specified.

---

### Parameters

`File` : `File_Type`;

Specifies the handle for the file.

`To` : `Count`;

Specifies the value of the line length.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the specified line length is inappropriate for the associated file, the `Io_Exceptions.Use_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

```
procedure Set_Output  
package !Io.Io
```

## procedure Set\_Output

---

```
procedure Set_Output (File : File_Type);  
procedure Set_Output (Name : String := ">>FILE NAME<<");
```

---

### **Description**

Sets the current default output file to the specified file handle or file and pushes the file onto the stack of output files.

If a filename is specified, the file is opened with `Out_File` mode before it is set to be the default output file. A file is created if one does not exist.

Any files on the stack when a job terminates are automatically closed.

---

### **Parameters**

File : File\_Type;

Specifies the handle for the file.

Name : String := ">>FILE NAME<<";

Specifies the filename to open (and create if necessary) and set to be the current default output file.

---

## Errors

If a file handle is specified and it is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

The `Io_Exceptions.Name_Error` exception is raised under the following conditions:

- The filename does not conform to the syntax of a name.
- An object of a nonfile class with the same name as the filename already exists in the context in which the creation is attempted.
- The context in which the creation is attempted cannot contain files. Files are allowed only in directories or worlds.

The `Io_Exceptions.Use_Error` exception is raised under the following conditions:

- The file cannot be opened with `Out_File` mode.
- The file must be created and the executing job does not have create access.
- The file is locked by another job.

---

## References

procedure `Pop_Output`

procedure `Reset_Output`

---

```
procedure Set_Page_Length
package !Io.Io
```

## procedure Set\_Page\_Length

---

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

---

### Description

Sets the maximum page length of the specified output file to the number of lines specified by the To parameter.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

To : Count;  
Specifies the value of the page length.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the specified page length is inappropriate for the associated file, the Io\_Exceptions.Use\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---



## procedure Skip\_Line

---

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

```
procedure Skip_Line (Spacing : Positive_Count := 1);
```

---

### Description

Ignores all remaining characters in the subsequent lines (specified by the Spacing parameter) and sets the current line to the following line.

For a spacing of 1, this procedure reads and discards all characters until a line terminator has been read, and then sets the current column number to 1. If the line terminator is not immediately followed by a page terminator, the current line number is increased by 1. Otherwise, if the line terminator is immediately followed by a page terminator, the page terminator is skipped, the current page number is increased by 1, and the current line number is set to 1.

For a spacing greater than 1, the above actions are performed the number of times specified by the Spacing parameter.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Spacing : Positive\_Count := 1;

Specifies the number of lines to be skipped.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If an attempt is made to read a file terminator, the Io\_Exceptions.End\_Error exception is raised.

If the mode of the file is not In\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

## procedure Skip\_Page

---

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

---

### **Description**

Skips past all input until a page terminator is read.

This procedure reads and discards all characters and line terminators until a page terminator has been read. It then adds 1 to the current page number and sets the current column and line numbers to 1.

If a File parameter is omitted, the current default file is understood to be specified.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If an attempt is made to read a file terminator, the Io\_Exceptions.End\_Error exception is raised.

If the mode of the file is not In\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

## function Standard\_Error

---

```
function Standard_Error return File_Type;  
function Standard_Error return Text_Io.File_Type;
```

---

### **Description**

Returns the handle to the standard error file (by default, the Message window).

---

```
function Standard_Input  
package !Io.Io
```

## function Standard\_Input

---

```
function Standard_Input return File_Type;
```

---

### **Description**

Returns the handle to the standard input file.

---

## function Standard\_Output

---

function Standard\_Output return File\_Type;

---

### **Description**

Returns the handle to the standard output file.

---

```
subtype Type_Set  
package !lo.lo
```

## subtype Type\_Set

---

```
subtype Type_Set is Text_lo.Type_Set;
```

---

### **Description**

Specifies the case in which enumeration literals are to be displayed.

---

## constant Unbounded

---

Unbounded : constant Count := Text\_!o.Unbounded;

---

### **Description**

Denotes an unbounded line and/or page length.

---

constant Upper\_Case  
package !Io.Io

## constant Upper\_Case

---

Upper\_Case : constant Type\_Set := Text\_Io.Upper\_Case;

---

### **Description**

Defines a value indicating that enumeration literals are to be displayed in uppercase.

---



## generic procedure Wildcard\_Iterator

This procedure calls a procedure once with an open file handle with mode `In_File` corresponding to each of the files matched by the wildcard or filename specified by the input parameter. It can be used to perform an operation repeatedly over a set of files.

The procedure's formal parameter list is:

```
generic
  with procedure Process (File : in out File_Type) is <>;
  with procedure Note_Error (Message : String) is Io.Put_Line;
procedure Wildcard_Iterator (Names : String);
```

The `Process` procedure is called for each file. The `Note_Error` procedure is used to report exceptions.

generic formal procedure Note\_Error  
package !Io.Io

## generic formal procedure Note\_Error

---

with procedure Note\_Error (Message : String) is Io.Put\_Line;

---

### **Description**

Reports errors and exceptions encountered in the processing of each file matched by the wildcard passed to the Wildcard\_Iterator procedure.

If no procedure is provided for this generic formal in an instantiation of the Wildcard\_Iterator, the Put\_Line procedure will be used.

---

### **Parameters**

Message : String;

Specifies the error message to be output.

---

## generic formal procedure Process

---

with procedure Process (File : in out File\_Type) is ◊;

---

### Description

Performs the processing on each file matched by the wildcard.

This procedure will be called once for each file matched by the wildcard. The file will be opened with In\_File mode before the Process procedure is called by the Wildcard\_Iterator procedure.

If no procedure is provided for this generic formal in an instantiation of the Wildcard\_Iterator, a procedure (visible at the point of instantiation) with the name Process and a matching parameter profile will be used. Lack of either an explicit parameter or such a visible procedure is a semantic error.

---

### Parameters

File : in out File\_Type;  
Specifies the file to be processed.

---

```
procedure Wildcard_Iterator
package !Io.Io
```

## procedure Wildcard\_Iterator

---

```
procedure Wildcard_Iterator (Names : String);
```

---

### **Description**

Resolves the wildcard or filename provided.

For each file matched, the procedure performs the following processing in sequence:

- Opens the file with In\_File mode
- Calls the Process procedure
- Closes the file

If errors are encountered, the Note\_Error procedure is called and processing continues.

---

### **Parameters**

Names : String;

Specifies the wildcard or filename denoting a set of files.

---

---

```
end Wildcard_Iterator;
```

---

## generic package Enumeration\_Io

This package provides facilities for enumeration I/O.

Values are output using either uppercase or lowercase letters for identifiers. This is specified by the Set parameter, which is of the enumeration Type\_Set subtype.

The format (which includes any trailing spaces) can be specified by an optional field Width parameter.

constant Default\_Setting  
package !Io.Io.Enumeration\_Io

## constant Default\_Setting

---

Default\_Setting : Type\_Set := Upper\_Case;

---

### **Description**

Denotes the default type set of values to be output.

---

## constant Default\_Width

---

Default\_Width : Field := 0;

---

### **Description**

Denotes the default number of characters to be output.

---

generic formal type Enum  
package !Io.Io.Enumeration\_Io

## generic formal type Enum

---

type Enum is (<>);

---

### **Description**

Denotes the enumeration type to be used in instantiating this package.

---



## procedure Get

---

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

```
procedure Get (Item : out Enum);
```

---

### Description

Reads an enumeration value from a file.

After skipping any leading blanks, line terminators, or page terminators, the procedure reads an enumeration literal of Enum type in either lowercase or uppercase. If Enum is of Character type, the literal value must include the apostrophes.

The procedure returns, in the Item parameter, the value of the Enum type that corresponds to the sequence input.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : out Enum;

Specifies the object that receives the value read.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not In\_File, the Io\_Exceptions.Mode\_Error exception is raised.

If an attempt is made to skip a file terminator, the Io\_Exceptions.End\_Error exception is raised.

If the sequence input does not have the required syntax, or if the identifier or character literal does not correspond to a value of the Enum type, the Io\_Exceptions.Data\_Error exception is raised.

---

```
procedure Get
package !Io.Io.Enumeration_Io
```

## procedure Get

---

```
procedure Get (From : String;
              Item : out Enum;
              Last : out Positive);
```

---

### Description

Reads an enumeration value from a string.

The procedure reads an enumeration value from the beginning of the specified string, following the same rule as the Get procedure that reads an enumeration value from a file but treating the end of the string as a file terminator.

The procedure returns, in the Item parameter, the value of the Enum type that corresponds to the sequence input. It returns, in the Last parameter, the index value such that From(Last) is the last character read.

---

### Parameters

From : String;

Specifies the string to be read.

Item : out Enum;

Specifies the object that receives the value read.

Last : out Positive;

Specifies the index of the last character read.

---

### Errors

If the sequence input does not have the required syntax, or if the identifier or character literal does not correspond to a value of the Enum type, the Io\_Exceptions.Data\_Error exception is raised.

---

## procedure Put

---

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

---

### Description

Writes an enumeration value to a file.

The procedure outputs the value of the Item parameter as an enumeration literal (either an identifier or a character literal). The optional Set parameter indicates whether lowercase or uppercase letters are used for identifiers; it has no effect for character literals. If the sequence of characters produced has fewer than the number of characters specified by the Width parameter, trailing spaces are output to make up the difference.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : Enum;  
Specifies the value to be written.

Width : Field := Default\_Width;  
Specifies the number of characters to be written.

Set : Type\_Set := Default\_Setting;  
Specifies the type set of the value to be written.

procedure Put  
package !Io.Io.Enumeration\_Io

---

**Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

## procedure Put

---

```
procedure Put (To : out String;  
              Item : Enum;  
              Set : Type_Set := Default_Setting);
```

---

### Description

Writes an enumeration value to a string.

The procedure outputs the value of the Item parameter to the specified string, following the same rule as for output to a file, using the length of the specified string as the value for the Width parameter.

---

### Parameters

To : out String;

Specifies the string to which the value is to be written.

Item : Enum;

Specifies the value to be written.

Set : Type\_Set := Default\_Setting;

Specifies the type set of the value to be written.

---

### Errors

If the length of the actual string is insufficient for the output of the item, the Io\_Exceptions.Layout\_Error exception is raised.

---

---

end Enumeration\_Io;

---

RATIONAL

## generic package Fixed\_Io

This package provides facilities for fixed-point I/O.

Values are output as decimal literals without underline characters. The format of each value output consists of a Fore field, a decimal point, an Aft field, and (if a nonzero Exp parameter is supplied) the letter E and an Exp field. The two possible formats thus correspond to:

Fore . Aft

and:

Fore . Aft E Exp

with no spaces between these fields. The Fore field can include leading spaces and a minus sign for negative values. The Aft field includes only decimal digits (possibly with trailing zeros). The Exp field includes the sign (plus or minus) and the exponent (possibly with leading zeros).

constant Default\_Aft  
package !Io.Io.Fixed\_Io

## constant Default\_Aft

---

Default\_Aft : Field := Num'Aft;

---

### **Description**

Denotes the default number of characters to be used after the decimal point in output procedures.

---



## constant Default\_Exp

---

Default\_Exp : Field := 0;

---

### **Description**

Denotes the default number of exponent characters to be used in output procedures.

---

constant Default\_Fore  
package !Io.Io.Fixed\_Io

## constant Default\_Fore

---

Default\_Fore : Field := Num'Fore;

---

### **Description**

Denotes the default number of characters to be used before the decimal point in output procedures.

---

## procedure Get

---

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

---

### Description

Reads a fixed-point value from a file.

If the value of the Width parameter is 0, the procedure skips any leading blanks, line terminators, or page terminators, reads a plus or a minus sign if present, and then reads according to the syntax of a real literal (which may be a based literal). If a nonzero value of the Width parameter is supplied, then exactly Width characters are input, or the characters (possibly none) up to a line terminator are input, whichever comes first; any skipped leading blanks are included in the count.

The procedure returns, in the Item parameter, the value of the Num type that corresponds to the sequence input.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : out Num;  
Specifies the object that receives the value read.

Width : Field := 0;  
Specifies the number of characters to be read.

procedure Get  
package !Io.Io.Fixed\_Io

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the sequence input does not have the required syntax, or if the value obtained is not of the `Num` type, the `Io_Exceptions.Data_Error` exception is raised. For this test, leading blanks are ignored. When a sign is input, this rule applies to the succeeding numeric literal.

---

## procedure Get

---

```
procedure Get (From : String;  
              Item : out Num;  
              Last : out Positive);
```

---

### Description

Reads a fixed-point value from a string.

This procedure reads a real value from the beginning of the specified string, following the same rule as the Get procedure that reads a real value from a file but treating the end of the string as a file terminator.

The procedure returns, in the Item parameter, the value of the Num type that corresponds to the sequence input. It returns, in the Last parameter, the index value such that From(Last) is the last character read.

---

### Parameters

From : String;  
Specifies the string to be read.

Item : out Num;  
Specifies the object that receives the value read.

Last : out Positive;  
Specifies the index of the last character read.

---

### Errors

If the sequence input does not have the required syntax, or if the value obtained is not of the Num type, the Io\_Exceptions.Data\_Error exception is raised. When a sign is input, this rule applies to the succeeding numeric literal.

---

generic formal type Num  
package !Io.Io.Fixed\_Io

## generic formal type Num

---

type Num is delta <>;

---

### **Description**

Denotes the fixed-point type for which the package is being instantiated.

---

## procedure Put

---

```
procedure Put (File : File_Type;  
              Item : Num;  
              Fore : Field      := Default_Fore;  
              Aft  : Field      := Default_Aft;  
              Exp  : Field      := Default_Exp);  
  
procedure Put (Item : Num;  
              Fore : Field := Default_Fore;  
              Aft  : Field := Default_Aft;  
              Exp  : Field := Default_Exp);
```

---

### Description

Writes a fixed-point value to a file.

The Put procedure outputs the value of the Item parameter as a decimal literal with the format defined by the Fore, Aft, and Exp parameters. If the value is negative, a minus sign is included in the integer part. If Exp has the value 0, the integer part to be output has as many digits as needed to represent the integer part of the value of the Item parameter, overriding Fore if necessary, or it consists of the digit 0 if the value of Item has no integer part.

If Exp has a value greater than 0, the integer part to be output has a single digit, which is nonzero except for the value 0.0 of the Item parameter.

In both cases, however, if the integer part to be output has fewer than Fore characters, including any minus sign, leading spaces are output to make up the difference. The number of digits of the fractional part is specified by Aft, or it is 1 if Aft equals 0. The value is rounded; a value of exactly one-half in the last place can be rounded either up or down.

If Exp has the value 0, there is no exponent part. If Exp has a value greater than 0, the exponent part to be output has as many digits as needed to represent the exponent part of the value of the Item parameter (for which a single-digit integer part is used) and includes an initial sign (plus or minus). If the exponent part to be output has fewer than Exp characters, including the sign, leading zeros precede the digits to make up the difference. For the value 0.0 of Item, the exponent has the value 0.

If a File parameter is omitted, the current default file is understood to be specified.

procedure Put  
package !Io.Io.Fixed\_Io

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : Num;

Specifies the value to be written.

Fore : Field := Default\_Fore;

Specifies the number of characters to be output before the decimal point.

Aft : Field := Default\_Aft;

Specifies the number of characters to be output after the decimal point.

Exp : Field := Default\_Exp;

Specifies the number of exponent characters to be output.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---



## procedure Put

---

```
procedure Put (To   : out String;  
              Item :   Num;  
              Aft  :   Field := Default_Aft;  
              Exp  :   Field := Default_Exp);
```

---

### Description

Writes a fixed-point value to a string.

This procedure outputs the value of the Item parameter to the specified string, following the same rule as for output to a file, using a value for the width of the Fore field such that the sequence of characters output exactly fills the string, including any leading spaces.

For an item with a positive value, if output to a string exactly fills the string without leading spaces, then output of the corresponding negative value raises the Io\_Exceptions.Layout\_Error exception.

---

### Parameters

To : out String;

Specifies the string to which the value is to be written.

Item : Num;

Specifies the value to be written.

Aft : Field := Default\_Aft;

Specifies the number of characters to be output after the decimal point.

Exp : Field := Default\_Exp;

Specifies the number of exponent characters to be output.

procedure Put  
package !Io.Io.Fixed\_Io

---

**Errors**

If the length of the actual string is insufficient for the output of the item, the `Io_Exceptions.Layout_Error` exception is raised.

---

---

end Fixed\_Io;

---

## generic package Float\_Io

This package provides facilities for floating-point I/O.

Values are output as decimal literals without underline characters. The format of each value output consists of a Fore field, a decimal point, an Aft field, and (if a nonzero Exp parameter is supplied) the letter E and an Exp field. The two possible formats thus correspond to:

Fore . Aft

and:

Fore . Aft E Exp

with no spaces between these fields. The Fore field can include leading spaces and a minus sign for negative values. The Aft field includes only decimal digits (possibly with trailing zeros). The Exp field includes the sign (plus or minus) and the exponent (possibly with leading zeros).

constant Default\_Aft  
package !Io.Io.Float\_Io

## constant Default\_Aft

---

Default\_Aft : Field := Num'Digits - 1;

---

### **Description**

Denotes the default width to be used after the decimal point in output procedures.

---

## constant Default\_Exp

---

Default\_Exp : Field := 3;

---

### **Description**

Denotes the default exponent width to be used in output procedures.

---

constant Default\_Fore  
package !Io.Io.Float\_Io

## constant Default\_Fore

---

Default\_Fore : Field := 2;

---

### **Description**

Denotes the default width to be used before the decimal point in output procedures.

---

## procedure Get

---

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

---

### Description

Reads a floating-point number from a file.

If the value of the `Width` parameter is 0, the procedure skips any leading blanks, line terminators, or page terminators, reads a plus or a minus sign if present, and then reads according to the syntax of a real literal (which may be a based literal). If a nonzero value of the `Width` parameter is supplied, then exactly `Width` characters are input, or the characters (possibly none) up to a line terminator are input, whichever comes first; any skipped leading blanks are included in the count.

The procedure returns, in the `Item` parameter, the value of the `Num` type that corresponds to the sequence input.

If a `File` parameter is omitted, the current default file is understood to be specified.

---

### Parameters

`File` : `File_Type`;  
Specifies the handle for the file.

`Item` : `out Num`;  
Specifies the object that receives the value read.

`Width` : `Field := 0`;  
Specifies the number of characters to be read.

procedure Get  
package !Io.Io.Float\_Io

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the sequence input does not have the required syntax, or if the value obtained is not of the `Num` type, the `Io_Exceptions.Data_Error` exception is raised. For this test, leading blanks are ignored. When a sign is input, this rule applies to the succeeding numeric literal.

---



## procedure Get

---

```
procedure Get (From : String;  
              Item : out Num;  
              Last : out Positive);
```

---

### Description

Reads a floating-point value from a string.

This procedure reads a real value from the beginning of the specified string, following the same rule as the Get procedure that reads a real value from a file but treating the end of the string as a file terminator.

The procedure returns, in the Item parameter, the value of the Num type that corresponds to the sequence input. It returns, in the Last parameter, the index value such that From(Last) is the last character read.

---

### Parameters

From : String;  
Specifies the string to be read.

Item : out Num;  
Specifies the object that receives the value read.

Last : out Positive;  
Specifies the index of the last character read.

---

### Errors

If the sequence input does not have the required syntax, or if the value obtained is not of the Num type, the Io\_Exceptions.Data\_Error exception is raised. When a sign is input, this rule applies to the succeeding numeric literal.

---

generic formal type Num  
package !Io.Io.Float\_Io

## generic formal type Num

---

type Num is digits <>;

---

### **Description**

Defines the floating-point type of the items that form the elements in the I/O stream.

---

## procedure Put

---

```
procedure Put (File : File_Type;  
              Item : Num;  
              Fore : Field      := Default_Fore;  
              Aft  : Field      := Default_Aft;  
              Exp  : Field      := Default_Exp);  
  
procedure Put (Item : Num;  
              Fore : Field := Default_Fore;  
              Aft  : Field := Default_Aft;  
              Exp  : Field := Default_Exp);
```

---

### Description

Writes a floating-point value to a file.

This procedure outputs the value of the `Item` parameter as a decimal literal with the format defined by the `Fore`, `Aft`, and `Exp` parameters. If the value is negative, a minus sign is included in the integer part. If `Exp` has the value 0, the integer part to be output has as many digits as needed to represent the integer part of the value of the `Item` parameter, overriding `Fore` if necessary, or it consists of the digit 0 if the value of `Item` has no integer part.

If `Exp` has a value greater than 0, the integer part to be output has a single digit, which is nonzero except for the value 0.0 of the `Item` parameter.

In both cases, however, if the integer part to be output has fewer than `Fore` characters, including any minus sign, leading spaces are output to make up the difference. The number of digits of the fractional part is specified by `Aft`, or it is 1 if `Aft` equals 0. The value is rounded; a value of exactly one-half in the last place can be rounded either up or down.

If `Exp` has the value 0, there is no exponent part. If `Exp` has a value greater than 0, the exponent part to be output has as many digits as needed to represent the exponent part of the value of the `Item` parameter (for which a single-digit integer part is used) and includes an initial sign (plus or minus). If the exponent part to be output has fewer than `Exp` characters, including the sign, leading zeros precede the digits to make up the difference. For the value 0.0 of `Item`, the exponent has the value 0.

If a `File` parameter is omitted, the current default file is understood to be specified.

procedure Put  
package !Io.Io.Float\_Io

---

### **Parameters**

File : File\_Type;

Specifies the handle for the file.

Item : Num;

Specifies the value to be written.

Fore : Field := Default\_Fore;

Specifies the number of characters to be output before the decimal point.

Aft : Field := Default\_Aft;

Specifies the number of characters to be output after the decimal point.

Exp : Field := Default\_Exp;

Specifies the number of exponent characters to be output.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

## procedure Put

---

```
procedure Put (To   : out String;  
              Item :   Num;  
              Aft  :   Field := Default_Aft;  
              Exp  :   Field := Default_Exp);
```

---

### Description

Writes a floating-point value to a string.

This procedure outputs the value of the Item parameter to the specified string, following the same rule as for output to a file, using a value for the width of the Fore field such that the sequence of characters output exactly fills the string, including any leading spaces.

For an item with a positive value, if output to a string exactly fills the string without leading spaces, then output of the corresponding negative value raises the `Io_Exceptions.Layout_Error` exception.

---

### Parameters

To : out String;

Specifies the string to which the value is to be written.

Item : Num;

Specifies the value to be written.

Aft : Field := Default\_Aft;

Specifies the number of characters to be written after the decimal point.

Exp : Field := Default\_Exp;

Specifies the number of exponent characters to be written.

---

### Errors

If the length of the actual string is insufficient for the output of the item, the `Io_Exceptions.Layout_Error` exception is raised.

```
procedure Put
package !Io.Io.Float_Io
```

---

**Example**

```
package Real_Io is new Float_Io(Real); use Real_Io;
-- default format used at instantiation, Default_Exp = 3

X : Real := -123.4567; -- digits 8      (see 3.5.7)

Put(X); -- default format                "-1.2345670E+02"
Put(X, Fore => 5, Aft => 3, Exp => 2); -- "bbb-1.235E+2"
Put(X, 5, 3, 0); -- "b-123.457"
```

---

---

```
end Float_Io;
```

---

## generic package Integer\_Io

This package provides facilities for integer I/O.

Values are output as decimal or based literals, without underline characters or exponents, and preceded by a minus sign if negative. The format (which includes any leading spaces and a minus sign) can be specified by an optional field Width parameter. Values of widths of fields in output formats are of the nonnegative integer Field subtype. Values of bases are of the integer Number\_Base subtype.

constant Default\_Base  
package !Io.Io.Integer\_Io

## constant Default\_Base

---

Default\_Base : Number\_Base := 10;

---

### **Description**

Denotes the default radix base to be used in output procedures.

---



## constant Default\_Width

---

Default\_Width : Field := Num'Width;

---

### **Description**

Denotes the default width to be used in output procedures.

---

```
procedure Get
package !Io.Io.Integer_Io
```

## procedure Get

---

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

procedure Get (Item : out Num;
              Width : Field := 0);
```

---

### Description

Reads an integer value from a file.

If the value of the **Width** parameter is 0, the procedure skips any leading blanks, line terminators, or page terminators, reads a plus or a minus sign if present, and then reads according to the syntax of an integer literal (which may be a based literal). If a nonzero value of the **Width** parameter is supplied, then exactly **Width** characters are input, or the characters (possibly none) up to a line terminator are input, whichever comes first; any skipped leading blanks are included in the count.

The procedure returns, in the **Item** parameter, the value of the **Num** type that corresponds to the sequence input.

If a **File** parameter is omitted, the current default file is understood to be specified.

---

### Parameters

**File** : File\_Type;

Specifies the handle for the file.

**Item** : out Num;

Specifies the object that receives the value read.

**Width** : Field := 0;

Specifies the number of characters to be read.

---

**Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the sequence input does not have the required syntax, or if the value obtained is not of the `Num` type, the `Io_Exceptions.Data_Error` exception is raised. When a sign is input, this rule applies to the succeeding numeric literal.

---

```
procedure Get
package !Io.Io.Integer_Io
```

## procedure Get

---

```
procedure Get (From : String;
              Item : out Num;
              Last : out Positive);
```

---

### Description

Reads an integer value from a string.

This procedure reads an integer value from the beginning of the specified string, following the same rules as the Get procedure that reads an integer value from a file but treating the end of the string as a file terminator.

The procedure returns, in the Item parameter, the value of the Num type that corresponds to the sequence input. It returns, in the Last parameter, the index value such that From(Last) is the last character read.

---

### Parameters

From : String;

Specifies the string to be read.

Item : out Num;

Specifies the object that receives the value read.

Last : out Positive;

Specifies the index of the last character read.

---

### Errors

If the sequence input does not have the required syntax, or if the value obtained is not of the Num type, the Io\_Exceptions.Data\_Error exception is raised. When a sign is input, this rule applies to the succeeding numeric literal.

---

## generic formal type Num

---

type Num is range <>;

---

### **Description**

Defines the integer type of the items that form the elements in the I/O stream.

---

```
procedure Put
package !Io.Io.Integer_Io
```

## procedure Put

---

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

```
procedure Put (Item : Num;
               Width : Field      := Default_Width;
               Base  : Number_Base := Default_Base);
```

---

### Description

Writes an integer value to a file.

This procedure outputs the value of the `Item` parameter as an integer literal, with no underlines, no exponent, and no leading zeros (but a single zero for the value 0), and with a preceding minus sign for a negative value.

If the resulting sequence of characters to be output has fewer characters than specified by the `Width` parameter, leading spaces are output to make up the difference.

The procedure uses the syntax for decimal literal if the `Base` parameter has the value 10 (either explicitly or through `Default_Base`); otherwise, it uses the syntax for based literal, with any letters in uppercase.

If a `File` parameter is omitted, the current default file is understood to be specified.

---

### Parameters

`File` : `File_Type`;

Specifies the handle for the file.

`Item` : `Num`;

Specifies the value to be written.

`Width` : `Field` := `Default_Width`;

Specifies the number of characters to be output.

`Base` : `Number_Base` := `Default_Base`;

Specifies the radix base with which to output the form of the number.

---

**Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

```
procedure Put
package !Io.Io.Integer_Io
```

## procedure Put

---

```
procedure Put (To : out String;
              Item : Num;
              Base : Number_Base := Default_Base);
```

---

### Description

Writes an integer value to a string.

This procedure outputs the value of the Item parameter to the specified string, following the same rule as for output to a file, using the length of the specified string as the value for the Width parameter.

---

### Parameters

To : out String;

Specifies the string to which the integer is to be written.

Item : Num;

Specifies the value of the integer to be written.

Base : Number\_Base := Default\_Base;

Specifies the radix base to be used in writing the number.

---

### Errors

If the length of the actual string is insufficient for the output of the item, the `Io_Exceptions.Layout_Error` exception is raised.



---

**Example**

```
package Int_Io is new Integer_Io(Small_Int); use Int_Io;
-- default format used at instantiation, Default_Width = 4,
--                                     Default_Base = 10
    Put(126);                               -- "b126"
    Put(-126, 7);                           -- "bbb-126"
    Put(126, Width => 13, Base => 2);       -- "bbb2#1111110#"

```

---

---

end Integer\_Io;

---

package !lo.lo

---

end lo;

---

## package Io\_Exceptions

The exceptions in package `Io_Exceptions` can be raised by I/O operations. The general conditions under which these exceptions can be raised are described in this section. Specific circumstances under which they can be raised are provided for each operation exported by an I/O package. If more than one error condition exists, the corresponding exception that appears earliest in the package is the one that is raised.

Every other I/O package renames one or more of the exceptions exported from this package. Rather than repeat the following descriptions in each of these packages, documentation of the renaming declarations is omitted in the subsequent sections.

The Rational Environment provides additional information about exceptions raised by the I/O packages that describes why a given exception occurred. This information, typically displayed in parentheses after the exception name, is documented in the reference entry for each exception.

exception Data\_Error  
package !Io.Io\_Exceptions

## exception Data\_Error

---

Data\_Error : exception;

---

### Description

Defines an exception raised by the Read procedure if the element read cannot be interpreted as a value of the required type.

This exception is also raised by a Get or Read procedure if an input sequence fails to satisfy the required syntax or if the value input does not belong to the range of the required type or subtype.

This exception is also raised by the Read and Write procedures of package Polymorphic\_Sequential\_Io (DIO) if these operations are attempted on files containing unsafe types (that is, containing access or task types as any of their components).

The additional information supplied by the Environment when this exception is raised has the following meaning:

- Input\_Syntax\_Error: The input value has incorrect syntax.
  - Input\_Value\_Error: The input value is out of range.
  - Output\_Type\_Error: The output value is an unsafe type.
  - Output\_Value\_Error: An attempt has been made to write a value out of range.
-

## exception Device\_Error

---

Device\_Error : exception;

---

### Description

Defines an exception raised if an I/O operation cannot be completed because of a malfunction of the underlying system.

The additional information supplied by the Environment when this exception is raised has the following meaning:

- Device\_Data\_Error: A hardware error such as a parity error has occurred.
  - Illegal\_Reference\_Error: An illegal reference has been attempted.
  - Illegal\_Heap\_Access\_Error: An Illegal\_Heap\_Access exception was raised when the operation was attempted.
  - Page\_Nonexistent\_Error: A nonexistent page was referenced.
  - Write\_To\_Read\_Only\_Page\_Error: A write to a read-only page was attempted.
-

exception End\_Error  
package !Io.Io\_Exceptions

## exception End\_Error

---

End\_Error : exception;

---

### **Description**

Defines an exception raised by an attempt to skip (read past) the end of a file.

---

## exception Layout\_Error

---

Layout\_Error : exception;

---

### Description

Defines an exception raised in TIO, packages Text\_Io and Io, by a call to operations that violate the limits of Count and by an attempt to put too many characters to a string; also raised in package Window\_Io (DIO) by an attempt to position the cursor outside the image boundary.

The additional information supplied by the Environment when this exception is raised has the following meaning:

- Column\_Error: A column exceeds the line or page length.
  - Illegal\_Position\_Error: A position parameter is illegal.
  - Item\_Length\_Error: An item length is too big or small.
-

exception Mode\_Error  
package !Io.Io\_Exceptions

## exception Mode\_Error

---

Mode\_Error : exception;

---

### **Description**

Defines an exception raised by specifying a file whose mode conflicts with the desired operation.

For example, this exception is raised by a call to Set\_Input or Get when a file of the Out\_File mode is provided.

The additional information supplied by the Environment when this exception is raised is:

- Illegal\_Operation\_On\_Infile
  - Illegal\_Operation\_On\_Outfile
-



## exception Name\_Error

---

Name\_Error : exception;

---

### Description

Defines an exception raised by a call to the Create or Open procedure if the string given for the Name parameter does not allow the identification of a legal unique file.

The Name\_Error exception is raised by the Create procedure under any of the following conditions:

- The filename does not conform to the syntax of a name.
- An object of the nonfile class with the same name as the filename already exists in the context in which the creation is attempted.
- The context in which the creation is attempted cannot contain files. Files are allowed only in directories or worlds.

The additional information supplied by the Environment when this exception is raised has the following meaning:

- **Ambiguous\_Name\_Error:** A name does not identify a unique object.
  - **Illformed\_Name\_Error:** A name does not conform to the syntax for a legal Environment filename.
  - **Nonexistent\_Directory\_Error:** A library in the name does not exist.
  - **Nonexistent\_Object\_Error:** The specified object does not exist.
  - **Nonexistent\_Version\_Error:** The specified version of the object does not exist.
-

```
exception Status_Error  
package !Io.Io_Exceptions
```

## exception Status\_Error

---

```
Status_Error : exception;
```

---

### **Description**

Defines an exception raised by an attempt to operate upon a file handle that is not open and by an attempt to open a file handle that is already open.

The additional information supplied by the Environment when this exception is raised has the following meaning:

- **Already\_Open\_Error:** The file handle is already open.
  - **Not\_Open\_Error:** The file handle is not open.
-

## exception Use\_Error

---

Use\_Error : exception;

---

### Description

Defines an exception raised if an operation is attempted that is not possible for reasons that depend on the file and the executing job's access rights.

This exception is raised by an attempt to create when there are objects of nonfile classes with similar names, by an attempt to open or reset with a mode that is not supported for the file, and by a call to the Open parameter for a terminal object if the terminal is already assigned to a job.

This exception is raised by the Delete procedure, among other circumstances, when the corresponding file is an object that cannot be deleted.

This exception is raised by the Create and Open procedures in packages Direct\_Io and Sequential\_Io (DIO) if they are attempted with instantiations on unsafe types (that is, types containing access or task types as any of their components).

The additional information supplied by the Environment when this exception is raised has the following meaning:

- Access\_Error: There are insufficient access rights to perform the operation.
  - Capacity\_Error: The output file is full.
  - Check\_Out\_Error: The object is not checked out using the configuration management and version control system.
  - Class\_Error: There is an existing object of a different class.
  - Frozen\_Error: An attempt is made to change a frozen object.
  - Line\_Page\_Length\_Error: An improper value for line or page length is encountered.
  - Lock\_Error: Another job has locked the object.
  - Reset\_Error: The file cannot be reset or have its mode changed.
  - Unsupported\_Error: The operation is not supported.
- 

end Io\_Exceptions;

---

RATIONAL

## package Text\_Io

This package provides the capabilities for Text\_Io as required by the *Reference Manual for the Ada Programming Language*, Chapter 14.

The fundamental abstraction provided by package Text\_Io is the File\_Type type. Objects of this type denote file handles that can be mapped to files. Each file is read or written sequentially, as a sequence of characters grouped into lines and a sequence of lines grouped into pages.

At the beginning of program execution, the default input and output files are the *standard input file* and *standard output file*. These files are open, have the In\_File and Out\_File modes, respectively, and are associated with two implementation-defined files. These files are implicitly closed at the end of each job.

From a logical point of view, a text file is a sequence of pages, a page is a sequence of lines, and a line is a sequence of characters. The characters of a line are numbered, starting from 1; the number of a character is called its column number. For a line terminator, a column number is also defined; it is one more than the number of characters in the line. The lines of a page and the pages of a file are similarly numbered. The current column number is the column number of the next character or line terminator to be transferred. The current line number is the number of the current line. The current page number is the number of the current page. These numbers are values of the Positive\_Count subtype of the Count type (by convention, the value 0 of the Count type is used to indicate special conditions).

```
type Count is range 0 .. 1_000_000_000
```

```
subtype Positive_Count is Count range 1 .. Count'Last;
```

For an output file, a maximum line length and a maximum page length can be specified. If a value to be output cannot fit on the current line, for a specified maximum line length, a new line is automatically started before the value is output. Further, if this new line cannot fit on the current page, for a specified maximum page length, a new page is automatically started before the value is output. Functions are provided to determine the maximum line and page lengths. When a file is opened with the Out\_File mode, both values are 0; by convention, this means that the line and page lengths are unbounded. (Consequently, output consists of a single line if the subprograms for explicit control of line and page structure are not used.) The Unbounded constant is provided for this purpose.

procedure Close  
package !Io.Text\_Io

## procedure Close

---

procedure Close (File : in out File\_Type);

---

### Description

Severs the association between the specified file and its associated file.

If the file has the current Out\_File mode, it has the effect of calling the New\_Page procedure, unless the current page is already terminated; then it outputs a file terminator.

---

### Parameters

File : in out File\_Type;  
Specifies the handle for the file.

---

### Restrictions

The standard input and output files cannot be closed.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

---

### References

procedure New\_Page

---

## function Col

---

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

---

### Description

Returns the current column number.

If a File parameter is omitted, the default file is the current output file.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Positive\_Count;  
Returns the current column number for the specified file.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the Positive\_Count value exceeds Count'Last, the Io\_Exceptions.Layout\_Error exception is raised.

---

type Count  
package !Io.Text\_Io

## type Count

---

type Count is range 0 .. 1\_000\_000\_000;

---

### **Description**

Specifies the range of possible values of the line and page counts and the line and page lengths.

---



## procedure Create

---

```
procedure Create (File : in out File_Type;  
                 Mode :           File_Mode := Out_File;  
                 Name :           String   := "";  
                 Form :           String   := "");
```

---

### Description

Establishes a new file with the specified name and associates this file with the specified file.

The specified file is left open.

---

### Parameters

File : in out File\_Type;  
Specifies the handle for the file.

Mode : File\_Mode := Out\_File;  
Specifies the access mode for which the file is to be used.

Name : String := "";  
Specifies the name of the file to be created. A null string for the Name parameter specifies a file that is not accessible after the completion of the main program (a temporary file).

Form : String := "";  
Currently, the Form parameter, if specified, has no effect.

---

### Restrictions

Files can be created only in directories or worlds.

## **Errors**

If the specified file handle is already open, the `Io_Exceptions.Status_Error` exception is raised.

The `Io_Exceptions.Name_Error` exception is raised under any of the following conditions:

- The filename does not conform to the syntax of a name.
- An object of a nonfile class with the same name as the filename already exists in the context in which the creation is attempted.
- The context in which the creation is attempted cannot contain files. Files are allowed only in directories or worlds.

The `Io_Exceptions.Use_Error` exception is raised under any of the following conditions:

- The file cannot be opened with the specified mode.
  - The job attempting the create does not have create access for the world in which the file is being created.
  - Another job has locked the file.
-

## function Current\_Input

---

function Current\_Input return File\_Type;

---

### **Description**

Returns the handle to the current default input file.

---

function Current\_Output  
package !Io.Text\_Io

## function Current\_Output

---

function Current\_Output return File\_Type;

---

### **Description**

Returns the handle to the current default output file.

---

## procedure Delete

---

```
procedure Delete (File : in out File_Type);
```

---

### Description

Deletes the file associated with the specified file.

The specified file is closed, and the file ceases to exist.

---

### Parameters

File : in out File\_Type;  
Specifies the handle for the file.

---

### Restrictions

The standard input and output files cannot be deleted.

---

### Errors

If the file handle is not open, the `Io_Exceptions.Status_Error` exception is raised.

The `Io_Exceptions.Use_Error` exception is raised under any of the following conditions:

- The Environment does not support deletion on the file.
  - The executing job does not have the access rights required to delete the file.
  - Another job has locked the file.
-

```
function End_Of_File
package !Io.Text_Io
```

## function End\_Of\_File

---

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

---

### Description

Returns true if a file terminator or the combination of a line, a page, and a file terminator is the next item to be read from the file; otherwise, the function returns false.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Boolean;

Returns true if a file terminator or the combination of a line, a page, and a file terminator is the next item to be read from the file; otherwise, the function returns false.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

## function End\_Of\_Line

---

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

---

### Description

Returns true if a line terminator or a file terminator is the next item to be read from the file; otherwise, the function returns false.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Boolean;

Returns true if a line terminator or a file terminator is the next item to be read from the file; otherwise, the function returns false.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

```
function End_Of_Page
package !Io.Text_Io
```

## function End\_Of\_Page

---

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

---

### Description

Returns true if a file terminator or the combination of a line terminator and a page terminator is the next item to be read from the file; otherwise, the function returns false.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Boolean;

Returns true if a file terminator or the combination of a line terminator and a page terminator is the next item to be read from the file; otherwise, the function returns false.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not In\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---



## subtype Field

---

subtype Field is Integer range 0 .. Integer'Last;

---

### **Description**

Specifies the range of possible values for the number of character positions used in formatting strings that represent discrete or real values.

---

type File\_Mode  
package !Io.Text\_Io

## type File\_Mode

---

type File\_Mode is (In\_File, Out\_File);

---

### **Description**

Specifies the mode of access for which a file is open.

In\_File denotes a file with read-only access; Out\_File denotes a file with write-only access.

---

## type File\_Type

---

type File\_Type is limited private;

---

### **Description**

Defines a file handle type for files to be processed by operations in this package.

---

```
function Form
package !Io.Text_Io
```

## function Form

---

```
function Form (File : File_Type) return String;
```

---

### Description

Returns the null string ("" ) in all cases.

When the Form parameter to the Create and Open procedures is supported in the future, this function will return the Form value provided in the call to the Create or the Open procedure.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

return String;

Returns the null string ("" ) in all cases.

---

### References

procedure Create

procedure Open

---

## procedure Get

---

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

---

### Description

Reads a character from a file.

After skipping any line and page terminators, this procedure reads the next character from the specified input file and returns the value of this character in the Item parameter.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : out Character;  
Specifies the object that receives the value read.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

---

```
procedure Get
package !Io.Text_Io
```

## procedure Get

---

```
procedure Get (File : File_Type;
              Item : out String);
procedure Get (Item : out String);
```

---

### **Description**

Reads a string from a file.

This procedure determines the length of the specified string and attempts that number of get operations for successive characters of the string. No operation is performed if the string is null.

If a File parameter is omitted, the current default file is understood to be specified.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

Item : out String;  
Specifies the object that receives the value read.

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

---

## procedure Get\_Line

---

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

---

### Description

Reads a string on a single line from a file, not including the line terminator.

This procedure replaces successive characters of the string by successive characters read from the specified input file. Reading stops if the end of the line is encountered, in which case the Skip\_Line procedure is called (in effect) with a spacing of 1; reading also stops if the end of the string is encountered. Characters not replaced are left undefined.

If characters are read, the Last parameter contains the index value such that Item(Last) is the last character replaced (the index of the first character replaced is Item'First). If no characters are read, the Last parameter contains an index value that is one less than Item'First.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : out String;

Specifies the object that receives the value read.

Last : out Natural;

Specifies the index for the last character read into the string.

procedure Get\_Line  
package !Io.Text\_Io

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

### **References**

procedure Skip\_Line

---



## function Is\_Open

---

```
function Is_Open (File : File_Type) return Boolean;
```

---

### **Description**

Returns true if the file handle is open (that is, if it is associated with a file); otherwise, the function returns false.

---

### **Parameters**

File : File\_Type;

Specifies the handle for the file.

return Boolean;

Returns true if the file handle is open (that is, if it is associated with a file); otherwise, the function returns false.

---

```
function Line
package !Io.Text_Io
```

## function Line

---

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

---

### **Description**

Returns the current line number.

If a File parameter is omitted, the default file is the current output file.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

return Positive\_Count;  
Returns the current line number for the specified file.

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the `Positive_Count` value exceeds `Count'Last`, the `Io_Exceptions.Layout_Error` exception is raised.

---

## function Line\_Length

---

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

---

### Description

Returns the maximum line length currently set for the specified output file; returns 0 if the line length is unbounded.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

return Count;

Returns the maximum line length currently set for the specified output file; returns 0 if the line length is unbounded.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

function Mode  
package !Io.Text\_Io

## function Mode

---

```
function Mode (File : File_Type) return File_Mode;
```

---

### **Description**

Returns the current mode of the specified file.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

return File\_Mode;  
Returns the current mode of the specified file.

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

---

## function Name

---

function Name (File : File\_Type) return String;

---

### **Description**

Returns the name of the file currently associated with the specified file handle.

For temporary files, this function returns the unique name provided by the Rational Environment during the creation of the file.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

return String;  
Returns the name of the file currently associated with the specified file handle.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

---

```
procedure New_Line
package !Io.Text_Io
```

## procedure New\_Line

---

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

---

### Description

Outputs a line terminator.

For a spacing of 1, this procedure outputs a line terminator, sets the current column number to 1, and adds 1 to the current line number. If the current line number is already greater than or equal to the maximum page length for a bounded page length, a page terminator is output, the current page number is increased by 1, and the current line number is set to 1.

For a spacing greater than 1, the above actions are performed the number of times specified by the Spacing parameter.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Spacing : Positive\_Count := 1;

Specifies the value denoting the number of new lines to be added.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

## procedure New\_Page

---

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

---

### Description

Outputs a page terminator.

This procedure outputs a line terminator if the current line is not terminated or if the current page is empty (that is, if the current column and line numbers are both equal to 1). It then outputs a page terminator, which terminates the current page, adds 1 to the current page number, and sets the current column and line numbers to 1.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

subtype Number\_Base  
package !Io.Text\_Io

## subtype Number\_Base

---

subtype Number\_Base is Integer range 2 .. 16;

---

### **Description**

Specifies the range of possible values for the radix of numeric values to be written or read.

---



## procedure Open

---

```
procedure Open (File : in out File_Type;  
               Mode : File_Mode := Out_File;  
               Name : String;  
               Form : String := "");
```

---

### Description

Associates the specified file with an existing file having the specified name and sets the mode of the file to the specified mode.

After a file is opened with the Out\_File mode, the page and line lengths are unbounded. After a file is opened with the In\_File or Out\_File mode, the current column, current line, and current page numbers are set to 1.

---

### Parameters

File : in out File\_Type;

Specifies the handle for the file.

Mode : File\_Mode := Out\_File;

Specifies the access mode for which the file is to be used.

Name : String;

Specifies the name of the file to be created.

Form : String := "";

Currently, the Form parameter, if specified, has no effect.

---

### Restrictions

The standard input and output files cannot be opened.

procedure Open  
package !Io.Text\_Io

---

### Errors

If the specified file handle is already open, the `Io_Exceptions.Status_Error` exception is raised.

If the string specified in the `Name` parameter does not allow the unique identification of a file, the `Io_Exceptions.Name_Error` exception is raised. In particular, this exception is raised if no file with the specified name exists.

The `Io_Exceptions.Use_Error` exception is raised if the file cannot be opened with the specified mode or if another job has locked the file.

---

## function Page

---

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

---

### **Description**

Returns the current page number.

If a File parameter is omitted, the default file is the current output file.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

return Positive\_Count;  
Returns the current page number.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the Positive\_Count value exceeds Count'Last, the Io\_Exceptions.Layout\_Error exception is raised.

---

```
function Page_Length
package !Io.Text_Io
```

## function Page\_Length

---

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

---

### **Description**

Returns the maximum page length currently set for the specified output file; returns 0 if the page length is unbounded.

If a File parameter is omitted, the current default file is understood to be specified.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

return Count;  
Returns the maximum page length currently set for the specified output file; returns 0 if the page length is unbounded.

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

## subtype Positive\_Count

---

subtype Positive\_Count is Count range 1 .. Count'Last;

---

### **Description**

Specifies the range of possible values for the current line and page number.

---

```
procedure Put
package !Io.Text_Io
```

## procedure Put

---

```
procedure Put (File : File_Type;
              Item : Character);
procedure Put (Item : Character);
```

---

### Description

Writes a character to a file.

If the line length of the specified output file is bounded (that is, does not have the conventional value 0) and the current column number exceeds it, this procedure has the effect of calling the `New_Line` procedure with a spacing of 1. It then outputs the specified character to the file.

If a `File` parameter is omitted, the current default file is understood to be specified.

---

### Parameters

`File` : `File_Type`;  
Specifies the handle for the file.

`Item` : `Character`;  
Specifies the value of the item to be written.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

### References

procedure `New_Line`

---

## procedure Put

---

```
procedure Put (File : File_Type;  
              Item : String);  
  
procedure Put (Item : String);
```

---

### Description

Writes a string to a file.

This procedure determines the length of the specified string and attempts that number of put operations for successive characters of the string. No operation is performed if the string is null.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : String;  
Specifies the value of the item to be written.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

```
procedure Put_Line
package !Io.Text_Io
```

## procedure Put\_Line

---

```
procedure Put_Line (File : File_Type;
                   Item : String);
```

```
procedure Put_Line (Item : String);
```

---

### Description

Writes a string to a file and advances the line.

This procedure calls the Put procedure for the specified string and then calls the New\_Line procedure with a spacing of 1.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : String;

Specifies the value of the string to be written.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

### References

procedure New\_Line

procedure Put

---



## procedure Reset

---

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

```
procedure Reset (File : in out File_Type);
```

---

### Description

Resets the specified file so that reading from or writing to its elements can be restarted from the beginning of the file.

If a Mode parameter is supplied, the current mode of the specified file is set to the specified mode.

If the file has the current Out\_File mode, this procedure has the effect of calling the New\_Page procedure, unless the current page is already terminated, and it then outputs a file terminator. If the new file mode is Out\_File, the page and line lengths are unbounded. For all modes, the current column, line, and page numbers are set to 1.

---

### Parameters

File : in out File\_Type;

Specifies the handle for the file.

Mode : File\_Mode;

Specifies the mode for which the file is to be used when the reset is completed.

---

### Restrictions

The standard input and output files cannot be reset.

## **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If an attempt is made to change the mode of a file that is either the current default input file or the current default output file, the `Io_Exceptions.Mode_Error` exception is raised.

The `Io_Exceptions.Use_Error` exception is raised under any of the following conditions:

- The Environment does not support resetting for the file.
  - The file cannot be opened with the specified mode.
  - Another job has locked the file.
- 

## **References**

procedure `New_Page`

---

## procedure Set\_Col

---

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

---

### Description

Sets the current column number of the specified file.

If the file mode is Out\_File:

If the value specified by the To parameter is greater than the current column number, this procedure outputs spaces, adding 1 to the current column number after each space, until the current column number equals the specified value. If the value specified by To is equal to the current column number, there is no effect. If the value specified by To is less than the current column number, the procedure has the effect of calling the New\_Line procedure (with a spacing of 1). It then outputs (To - 1) spaces and sets the current column number to the specified value.

If the file mode is In\_File:

This procedure reads (and discards) individual characters, line terminators, and page terminators until the next character to be read has a column number that equals the value specified by To; there is no effect if the current column number already equals this value. Each transfer of a character or terminator maintains the current column, line, and page numbers in the same way that a Get or a Get\_Line procedure does.

If a File parameter is omitted, the default file is the current output file.

The column, line, and page numbers are allowed to exceed Count'Last (as a consequence of the input or output of sufficiently many characters, lines, or pages).

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

To : Positive\_Count;  
Specifies the value of the column.

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the file mode is `Out_File`, the `Io_Exceptions.Layout_Error` exception is raised if the value specified by the `To` parameter exceeds `Line_Length` when the line length is bounded (that is, when it does not have the conventional value of 0).

If the file mode is `In_File`, the `Io_Exceptions.End_Error` exception is raised when an attempt is made to read a file terminator.

---

### **References**

procedure `Get`

procedure `Get_Line`

procedure `New_Line`

---

## procedure Set\_Input

---

```
procedure Set_Input (File : File_Type);
```

---

### Description

Sets the current default input file to the specified file handle.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

procedure Set\_Line  
package !Io.Text\_Io

## procedure Set\_Line

---

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

```
procedure Set_Line (To : Positive_Count);
```

---

### Description

Sets the current line number of the file.

If the file mode is `Out_File`:

If the value specified by the `To` parameter is greater than the current line number, this procedure has the effect of repeatedly calling the `New_Line` procedure (with a spacing of 1) until the current line number equals the specified value. If the value specified by `To` is equal to the current line number, there is no effect. If the value specified by `To` is less than the current line number, it has the effect of calling the `New_Page` procedure followed by a call to the `New_Line` procedure with a spacing equal to  $(To - 1)$ .

If the mode is `In_File`:

This procedure has the effect of repeatedly calling the `Skip_Line` procedure (with a spacing of 1) until the current line number equals the value specified by `To`; there is no effect if the current line number already equals this value. (Short pages will be skipped until a page is reached that has a line at the specified line position.)

If a `File` parameter is omitted, the default file is the current output file.

The column, line, and page numbers are allowed to exceed `Count'Last` (as a consequence of the input or output of sufficiently many characters, lines, or pages).

---

### Parameters

`File` : `File_Type`;

Specifies the handle for the file.

`To` : `Positive_Count`;

Specifies the value of the line.

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the file mode is `Out_File`, the `Io_Exceptions.Layout_Error` exception is raised if the value specified by the `To` parameter exceeds `Page_Length` when the page length is bounded (that is, when it does not have the conventional value of 0).

If the file mode is `In_File`, the `Io_Exceptions.End_Error` exception is raised when an attempt is made to read a file terminator.

---

### References

procedure `New_Line`

procedure `New_Page`

procedure `Skip_Line`

---

```
procedure Set_Line_Length
package !Io.Text_Io
```

## procedure Set\_Line\_Length

---

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

```
procedure Set_Line_Length (To : Count);
```

---

### Description

Sets the maximum line length of the specified output file to the number of characters specified by the To parameter.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

To : Count;

Specifies the value of the line length.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the specified line length is inappropriate for the associated file, the Io\_Exceptions.Use\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---



## procedure Set\_Output

---

```
procedure Set_Output (File : File_Type);
```

---

### **Description**

Sets the current default output file to the specified file handle.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

```
procedure Set_Page_Length  
package !Io.Text_Io
```

## procedure Set\_Page\_Length

---

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

```
procedure Set_Page_Length (To : Count);
```

---

### **Description**

Sets the maximum page length of the specified output file to the number of lines specified by the To parameter.

If a File parameter is omitted, the current default file is understood to be specified.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

To : Count;  
Specifies the value of the page length.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the specified page length is inappropriate for the associated file, the Io\_Exceptions.Use\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

## procedure Skip\_Line

---

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

---

### Description

Skips all subsequent input until the next line terminator.

For a spacing of 1, this procedure reads and discards all characters until a line terminator has been read, and then it sets the current column number to 1. If the line terminator is not immediately followed by a page terminator, the current line number is increased by 1. Otherwise, if the line terminator is immediately followed by a page terminator, the page terminator is skipped, the current page number is increased by 1, and the current line number is set to 1.

For a spacing greater than 1, the above actions are performed the number of times specified by the Spacing parameter.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Spacing : Positive\_Count := 1;  
Specifies the value denoting the number of lines to be skipped.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If an attempt is made to read a file terminator, the Io\_Exceptions.End\_Error exception is raised.

If the mode of the file is not In\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

```
procedure Skip_Page
package !Io.Text_Io
```

## procedure Skip\_Page

---

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

---

### **Description**

Skips past all input until a page terminator is read.

This procedure reads and discards all characters and line terminators until a page terminator has been read. Then it adds 1 to the current page number and sets the current column and line numbers to 1.

If a File parameter is omitted, the current default file is understood to be specified.

---

### **Parameters**

File : File\_Type;  
Specifies the handle for the file.

---

### **Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If an attempt is made to read a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

## function Standard\_Input

---

function Standard\_Input return File\_Type;

---

### **Description**

Returns the handle to the standard input file.

---

### **Restrictions**

The standard input file cannot be opened, closed, reset, or deleted.

---

function Standard\_Output  
package !Io.Text\_Io

## function Standard\_Output

---

function Standard\_Output return File\_Type;

---

### **Description**

Returns the handle to the standard output file.

---

### **Restrictions**

The standard output file cannot be opened, closed, reset, or deleted.

---

## type Type\_Set

---

type Type\_Set is (Lower\_Case, Upper\_Case);

---

### **Description**

Specifies the case in which enumeration literals are to be displayed.

---

constant Unbounded  
package !Io.Text\_Io

## constant Unbounded

---

Unbounded: constant Count := 0;

---

### **Description**

Denotes an unbounded line and/or page length.

---



## generic package Enumeration\_Io

This package provides facilities for enumeration I/O.

Values are output using either uppercase or lowercase letters for identifiers. This is specified by the Set parameter, which is of the enumeration Type\_Set type.

The format (which includes any trailing spaces) can be specified by an optional field Width parameter.

constant Default\_Setting  
package !Io.Text\_Io.Enumeration\_Io

## constant Default\_Setting

---

Default\_Setting : Type\_Set := Upper\_Case;

---

### **Description**

Denotes the default type set of values to be output.

---

## constant Default\_Width

---

Default\_Width : Field := 0;

---

### **Description**

Denotes the default number of characters to be output.

---

generic formal type Enum  
package !Io.Text\_Io.Enumeration\_Io

## generic formal type Enum

---

type Enum is (<>);

---

### **Description**

Denotes the enumeration type to be used in instantiating this package.

---

## procedure Get

---

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

---

### Description

Reads an enumeration value from a file.

After skipping any leading blanks, line terminators, or page terminators, this procedure reads an enumeration literal of Enum type in either lowercase or uppercase. If Enum is of Character type, the literal value must include the apostrophes.

The procedure returns, in the Item parameter, the value of the Enum type that corresponds to the sequence input.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : out Enum;  
Specifies the object that receives the value read.

---

### Errors

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not In\_File, the Io\_Exceptions.Mode\_Error exception is raised.

If an attempt is made to skip a file terminator, the Io\_Exceptions.End\_Error exception is raised.

If the sequence input does not have the required syntax, or if the identifier or character literal does not correspond to a value of the Enum type, the Io\_Exceptions.Data\_Error exception is raised.

---

```
procedure Get
package !Io.Text_Io.Enumeration_Io
```

## procedure Get

---

```
procedure Get (From : String;
              Item : out Enum;
              Last : out Positive);
```

---

### Description

Reads an enumeration value from a string.

This procedure reads an enumeration value from the beginning of the specified string, following the same rule as the Get procedure that reads an enumeration value from a file treating the end of the string as a file terminator.

The procedure returns, in the Item parameter, the value of the Enum type that corresponds to the sequence input. It returns, in the Last parameter, the index value such that From(Last) is the last character read.

---

### Parameters

From : String;

Specifies the string to be read.

Item : out Enum;

Specifies the object that receives the value read.

Last : out Positive;

Specifies the index of the last character read.

---

### Errors

If the sequence input does not have the required syntax, or if the identifier or character literal does not correspond to a value of the Enum type, the Io\_Exceptions.Data\_Error exception is raised.

---

## procedure Put

---

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

---

### Description

Writes an enumeration value to a file.

This procedure outputs the value of the Item parameter as an enumeration literal (either an identifier or a character literal). The optional Set parameter indicates whether lowercase or uppercase letters are used for identifiers; it has no effect for character literals. If the sequence of characters produced has fewer than Width characters, trailing spaces are output to make up the difference.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;

Specifies the handle for the file.

Item : Enum;

Specifies the value to be written.

Width : Field := Default\_Width;

Specifies the number of characters to be written.

Set : Type\_Set := Default\_Setting;

Specifies the type set of the value to be written.

procedure Put  
package !Io.Text\_Io.Enumeration\_Io

---

**Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---



## procedure Put

---

```
procedure Put (To : out String;  
              Item : Enum;  
              Set : Type_Set := Default_Setting);
```

---

### Description

Writes an enumeration value to a string.

This procedure outputs the value of the Item parameter to the specified string, following the same rule as for output to a file, using the length of the specified string as the value for the Width parameter.

---

### Parameters

To : out String;

Specifies the string to which the value is to be written.

Item : Enum;

Specifies the value to be written.

Set : Type\_Set := Default\_Setting;

Specifies the type set of the value to be written.

---

### Errors

If the length of the actual string is insufficient for the output of the item, the Io\_Exceptions.Layout\_Error exception is raised.

---

---

end Enumeration\_Io;

---

RATIONAL

## generic package Fixed\_Io

This package provides facilities for fixed-point I/O.

Values are output as decimal literals without underline characters. The format of each value output consists of a Fore field, a decimal point, an Aft field, and (if a nonzero Exp parameter is supplied) the letter E and an Exp field. The two possible formats thus correspond to:

Fore . Aft

and:

Fore . Aft E Exp

with no spaces between these fields. The Fore field can include leading spaces and a minus sign for negative values. The Aft field includes only decimal digits (possibly with trailing zeros). The Exp field includes the sign (plus or minus) and the exponent (possibly with leading zeros).

constant Default\_Aft  
package !Io.Text\_Io.Fixed\_Io

## constant Default\_Aft

---

Default\_Aft : Field := Num'Aft;

---

### **Description**

Denotes the default number of characters to be used after the decimal point in output procedures.

---

## constant Default\_Exp

---

Default\_Exp : Field := 0;

---

### **Description**

Denotes the default number of exponent characters to be used in output procedures.

---

constant Default\_Fore  
package !Io.Text\_Io.Fixed\_Io

## constant Default\_Fore

---

Default\_Fore : Field := Num'Fore;

---

### **Description**

Denotes the default number of characters to be used after the decimal point in output procedures.

---

## procedure Get

---

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

---

### Description

Reads a fixed-point value from a file.

If the value of the Width parameter is 0, this procedure skips any leading blanks, line terminators, or page terminators, reads a plus or a minus sign if present, and then reads according to the syntax of a real literal (which may be a based literal). If a nonzero value of the Width parameter is supplied, then exactly Width characters are input, or the characters (possibly none) up to a line terminator are input, whichever comes first; any skipped leading blanks are included in the count.

The procedure returns, in the Item parameter, the value of the Num type that corresponds to the sequence input.

If a File parameter is omitted, the current default file is understood to be specified.

---

### Parameters

File : File\_Type;  
Specifies the handle for the file.

Item : out Num;  
Specifies the object that receives the value read.

Width : Field := 0;  
Specifies the number of characters to be read.

procedure Get  
package !Io.Text\_Io.Fixed\_Io

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the sequence input does not have the required syntax, or if the value obtained is not of the `Num` type, the `Io_Exceptions.Data_Error` exception is raised. For this test, leading blanks are ignored. When a sign is input, this rule applies to the succeeding numeric literal.

---



## procedure Get

---

```
procedure Get (From : String;  
              Item : out Num;  
              Last : out Positive);
```

---

### Description

Reads a fixed-point value from a string.

This procedure reads a real value from the beginning of the specified string, following the same rule as the Get procedure that reads a real value from a file but treating the end of the string as a file terminator.

The procedure returns, in the Item parameter, the value of the Num type that corresponds to the sequence input. It returns, in the Last parameter, the index value such that From(Last) is the last character read.

---

### Parameters

From : String;  
Specifies the string to be read.

Item : out Num;  
Specifies the object that receives the value read.

Last : out Positive;  
Specifies the index of the last character read.

---

### Errors

If the sequence input does not have the required syntax, or if the value obtained is not of the Num type, the Io\_Exceptions.Data\_Error exception is raised. When a sign is input, this rule applies to the succeeding numeric literal.

---

generic formal type Num  
package !Io.Text\_Io.Fixed\_Io

## generic formal type Num

---

type Num is delta  $\diamond$ ;

---

### **Description**

Denotes the fixed-point type for which the package is being instantiated.

---

## procedure Put

---

```
procedure Put (File : File_Type;  
              Item : Num;  
              Fore : Field      := Default_Fore;  
              Aft  : Field      := Default_Aft;  
              Exp  : Field      := Default_Exp);
```

```
procedure Put (Item : Num;  
              Fore : Field := Default_Fore;  
              Aft  : Field := Default_Aft;  
              Exp  : Field := Default_Exp);
```

---

### Description

Writes a fixed-point value to a file.

The Put procedure outputs the value of the Item parameter as a decimal literal with the format defined by the Fore, Aft, and Exp parameters. If the value is negative, a minus sign is included in the integer part. If Exp has the value 0, the integer part to be output has as many digits as needed to represent the integer part of the value of the Item parameter, overriding Fore if necessary, or it consists of the digit 0 if the value of Item has no integer part.

If Exp has a value greater than 0, the integer part to be output has a single digit, which is nonzero except for the value 0.0 of the Item parameter.

In both cases, however, if the integer part to be output has fewer than Fore characters, including any minus sign, leading spaces are output to make up the difference. The number of digits of the fractional part is specified by Aft, or it is 1 if Aft equals 0. The value is rounded; a value of exactly one-half in the last place can be rounded either up or down.

If Exp has the value 0, there is no exponent part. If Exp has a value greater than 0, the exponent part to be output has as many digits as needed to represent the exponent part of the value of Item (for which a single-digit integer part is used), and it includes an initial sign (plus or minus). If the exponent part to be output has fewer than Exp characters, including the sign, leading zeros precede the digits to make up the difference. For the value 0.0 of Item, the exponent has the value 0.

If a File parameter is omitted, the current default file is understood to be specified.

procedure Put  
package !Io.Text\_Io.Fixed\_Io

---

### **Parameters**

File : File\_Type;

Specifies the handle for the file.

Item : Num;

Specifies the value to be written.

Fore : Field := Default\_Fore;

Specifies the number of characters to be output before the decimal point.

Aft : Field := Default\_Aft;

Specifies the number of characters to be output after the decimal point.

Exp : Field := Default\_Exp;

Specifies the number of exponent characters to be output.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---

## procedure Put

---

```
procedure Put (To    : out String;  
              Item  :   Num;  
              Aft   :   Field := Default_Aft;  
              Exp   :   Field := Default_Exp);
```

---

### Description

Writes a fixed-point value to a string.

This procedure outputs the value of the Item parameter to the specified string, following the same rule as for output to a file, using a value for the width of the Fore field such that the sequence of characters output exactly fills the string, including any leading spaces.

For an item with a positive value, if output to a string exactly fills the string without leading spaces, then output of the corresponding negative value raises the `Io_Exceptions.Layout_Error` exception.

---

### Parameters

To : out String;

Specifies the string to which the value is to be written.

Item : Num;

Specifies the value to be written.

Aft : Field := Default\_Aft;

Specifies the number of characters to be output after the decimal point.

Exp : Field := Default\_Exp;

Specifies the number of exponent characters to be output.

---

### Errors

If the length of the actual string is insufficient for the output of the item, the `Io_Exceptions.Layout_Error` exception is raised.

---

procedure Put  
package !Io.Text\_Io.Fixed\_Io

---

end Fixed\_Io;

---

## generic package Float\_Io

This package provides facilities for floating-point I/O.

Values are output as decimal literals without underline characters. The format of each value output consists of a Fore field, a decimal point, an Aft field, and (if a nonzero Exp parameter is supplied) the letter E and an Exp field. The two possible formats thus correspond to:

Fore . Aft

and:

Fore . Aft E Exp

with no spaces between these fields. The Fore field can include leading spaces and a minus sign for negative values. The Aft field includes only decimal digits (possibly with trailing zeros). The Exp field includes the sign (plus or minus) and the exponent (possibly with leading zeros).

constant Default\_Aft  
package !Io.Text\_Io.Float\_Io

## constant Default\_Aft

---

Default\_Aft : Field := Num'Digits - 1;

---

### **Description**

Denotes the default width to be used after the decimal point in output procedures.

---



## constant Default\_Exp

---

Default\_Exp : Field := 3;

---

### **Description**

Denotes the default exponent width to be used in output procedures.

---

constant Default\_Fore  
package !Io.Text\_Io.Float\_Io

## constant Default\_Fore

---

Default\_Fore : Field := 2;

---

### **Description**

Denotes the default width to be used before the decimal point in output procedures.

---

## procedure Get

---

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

---

### Description

Reads a floating-point number from a file.

If the value of the `Width` parameter is 0, this procedure skips any leading blanks, line terminators, or page terminators, reads a plus or a minus sign if present, and then reads according to the syntax of a real literal (which may be a based literal). If a nonzero value of the `Width` parameter is supplied, then exactly `Width` characters are input, or the characters (possibly none) up to a line terminator are input, whichever comes first; any skipped leading blanks are included in the count.

The procedure returns, in the `Item` parameter, the value of the `Num` type that corresponds to the sequence input.

If a `File` parameter is omitted, the current default file is understood to be specified.

---

### Parameters

`File` : `File_Type`;  
Specifies the handle for the file.

`Item` : `out Num`;  
Specifies the object that receives the value read.

`Width` : `Field := 0`;  
Specifies the number of characters to be read.

procedure Get  
package !Io.Text\_Io.Float\_Io

---

### Errors

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the sequence input does not have the required syntax, or if the value obtained is not of the `Num` type, the `Io_Exceptions.Data_Error` exception is raised. For this test, leading blanks are ignored. When a sign is input, this rule applies to the succeeding numeric literal.

---

## procedure Get

---

```
procedure Get (From : String;  
              Item : out Num;  
              Last : out Positive);
```

---

### Description

Reads a floating-point value from a string.

This procedure reads a real value from the beginning of the specified string, following the same rule as the Get procedure that reads a real value from a file but treating the end of the string as a file terminator.

The procedure returns, in the Item parameter, the value of the Num type that corresponds to the sequence input. It returns, in the Last parameter, the index value such that From(Last) is the last character read.

---

### Parameters

From : String;  
Specifies the string to be read.

Item : out Num;  
Specifies the object that receives the value read.

Last : out Positive;  
Specifies the index of the last character read.

---

### Errors

If the sequence input does not have the required syntax, or if the value obtained is not of the Num type, the Io\_Exceptions.Data\_Error exception is raised. When a sign is input, this rule applies to the succeeding numeric literal.

---

generic formal type Num  
package !Io.Text\_Io.Float\_Io

## generic formal type Num

---

type Num is digits <>;

---

### **Description**

Defines the floating-point type of the items that form the elements in the I/O stream.

---

## procedure Put

---

```
procedure Put (File : File_Type;  
              Item : Num;  
              Fore : Field      := Default_Fore;  
              Aft  : Field      := Default_Aft;  
              Exp  : Field      := Default_Exp);
```

```
procedure Put (Item : Num;  
              Fore : Field := Default_Fore;  
              Aft  : Field := Default_Aft;  
              Exp  : Field := Default_Exp);
```

---

### Description

Writes a floating-point value to a file.

This procedure outputs the value of the Item parameter as a decimal literal with the format defined by the Fore, Aft, and Exp parameters. If the value is negative, a minus sign is included in the integer part. If Exp has the value 0, the integer part to be output has as many digits as needed to represent the integer part of the value of the Item parameter, overriding Fore if necessary, or it consists of the digit 0 if the value of Item has no integer part.

If Exp has a value greater than 0, the integer part to be output has a single digit, which is nonzero except for the value 0.0 of the Item parameter.

In both cases, however, if the integer part to be output has fewer than Fore characters, including any minus sign, leading spaces are output to make up the difference. The number of digits of the fractional part is specified by Aft, or it is 1 if Aft equals 0. The value is rounded; a value of exactly one-half in the last place can be rounded either up or down.

If Exp has the value 0, there is no exponent part. If Exp has a value greater than 0, the exponent part to be output has as many digits as needed to represent the exponent part of the value of Item (for which a single-digit integer part is used), and it includes an initial sign (plus or minus). If the exponent part to be output has fewer than Exp characters, including the sign, leading zeros precede the digits to make up the difference. For the value 0.0 of Item, the exponent has the value 0.

If a File parameter is omitted, the current default file is understood to be specified.

procedure Put  
package !Io.Text\_Io.Float\_Io

---

### **Parameters**

File : File\_Type;

Specifies the handle for the file.

Item : Num;

Specifies the value to be written.

Fore : Field := Default\_Fore;

Specifies the number of characters to be output before the decimal point.

Aft : Field := Default\_Aft;

Specifies the number of characters to be output after the decimal point.

Exp : Field := Default\_Exp;

Specifies the number of exponent characters to be output.

---

### **Errors**

If the file is not open, the Io\_Exceptions.Status\_Error exception is raised.

If the mode of the file is not Out\_File, the Io\_Exceptions.Mode\_Error exception is raised.

---



## procedure Put

---

```
procedure Put (To : out String;  
              Item : Num;  
              Aft : Field := Default_Aft;  
              Exp : Field := Default_Exp);
```

---

### Description

Writes a floating-point value to a string.

This procedure outputs the value of the Item parameter to the specified string, following the same rule as for output to a file, using a value for the width of the Fore field such that the sequence of characters output exactly fills the string, including any leading spaces.

For an item with a positive value, if output to a string, the parameter exactly fills the string without leading spaces, then output of the corresponding negative value raises the Io\_Exceptions.Layout\_Error exception.

---

### Parameters

To : out String;

Specifies the string to which the value is to be written.

Item : Num;

Specifies the value to be written.

Aft : Field := Default\_Aft;

Specifies the number of characters to be output after the decimal point.

Exp : Field := Default\_Exp;

Specifies the number of exponent characters to be output.

---

### Errors

If the length of the actual string is insufficient for the output of the item, the Io\_Exceptions.Layout\_Error exception is raised.

```
procedure Put
package !Io.Text_Io.Float_Io
```

---

**Example**

```
package Real_Io is new Float_Io(Real); use Real_Io;
-- default format used at instantiation, Default_Exp = 3

X : Real := -123.4567; -- digits 8      (see 3.5.7)

Put(X); -- default format                "-1.2345670E+02"
Put(X, Fore => 5, Aft => 3, Exp => 2); -- "bbb-1.235E+2"
Put(X, 5, 3, 0); -- "b-123.457"
```

---

---

```
end Float_Io;
```

---

## generic package Integer\_Io

This package provides facilities for integer I/O.

Values are output as decimal or based literals, without underline characters or exponents, and preceded by a minus sign if negative. The format (which includes any leading spaces and a minus sign) can be specified by an optional field Width parameter. Values of widths of fields in output formats are of the nonnegative integer Field subtype. Values of bases are of the integer Number\_Base subtype.

constant Default\_Base  
package !Io.Text\_Io.Integer\_Io

## constant Default\_Base

---

Default\_Base : Number\_Base := 10;

---

### **Description**

Denotes the default radix base to be used in output procedures.

---

## constant Default\_Width

---

Default\_Width : Field := Num\_Width;

---

### **Description**

Denotes the default width to be used in output procedures.

---

```
procedure Get
package !Io.Text_Io.Integer_Io
```

## procedure Get

---

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

procedure Get (Item : out Num;
              Width : Field := 0);
```

---

### Description

Reads an integer value from a file.

If the value of the **Width** parameter is 0, this procedure skips any leading blanks, line terminators, or page terminators, reads a plus or a minus sign if present, and then reads according to the syntax of an integer literal (which may be a based literal). If a nonzero value of the **Width** parameter is supplied, then exactly **Width** characters are input, or the characters (possibly none) up to a line terminator are input, whichever comes first; any skipped leading blanks are included in the count.

The procedure returns, in the **Item** parameter, the value of the **Num** type that corresponds to the sequence input.

If a **File** parameter is omitted, the current default file is understood to be specified.

---

### Parameters

**File** : File\_Type;

Specifies the handle for the file.

**Item** : out Num;

Specifies the object that receives the value read.

**Width** : Field := 0;

Specifies the number of characters to be read.

---

**Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `In_File`, the `Io_Exceptions.Mode_Error` exception is raised.

If an attempt is made to skip a file terminator, the `Io_Exceptions.End_Error` exception is raised.

If the sequence input does not have the required syntax, or if the value obtained is not of the `Num` type, the `Io_Exceptions.Data_Error` exception is raised. When a sign is input, this rule applies to the succeeding numeric literal.

---

## procedure Get

---

```
procedure Get (From : String;  
              Item : out Num;  
              Last : out Positive);
```

---

### Description

Reads an integer value from a string.

This procedure reads an integer value from the beginning of the specified string, following the same rules as the Get procedure that reads an integer value from a file but treating the end of the string as a file terminator.

The procedure returns, in the Item parameter, the value of the Num type that corresponds to the sequence input. It returns, in the Last parameter, the index value such that From(Last) is the last character read.

---

### Parameters

From : String;

Specifies the string to be read.

Item : out Num;

Specifies the object that receives the value read.

Last : out Positive;

Specifies the index of the last character read.

---

### Errors

If the sequence input does not have the required syntax, or if the value obtained is not of the Num type, the Io\_Exceptions.Data\_Error exception is raised. When a sign is input, this rule applies to the succeeding numeric literal.

---



## generic formal type Num

---

type Num is range <>;

---

### **Description**

**Defines the integer type of the items that form the elements in the I/O stream.**

---

```
procedure Put
package !Io.Text_Io.Integer_Io
```

## procedure Put

---

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

procedure Put (Item  : Num;
              Width : Field      := Default_Width;
              Base  : Number_Base := Default_Base);
```

---

### Description

Writes an integer value to a file.

This procedure outputs the value of the `Item` parameter as an integer literal, with no underlines, no exponent, and no leading zeros (but a single zero for the value 0), and with a preceding minus sign for a negative value.

If the resulting sequence of characters to be output has fewer characters than specified by the `Width` parameter, leading spaces are output to make up the difference.

The procedure uses the syntax for decimal literal if the `Base` parameter has the value 10 (either explicitly or through `Default_Base`); otherwise, it uses the syntax for based literal, with any letters in uppercase.

If a `File` parameter is omitted, the current default file is understood to be specified.

---

### Parameters

`File` : `File_Type`;

Specifies the handle for the file.

`Item` : `Num`;

Specifies the value to be written.

`Width` : `Field` := `Default_Width`;

Specifies the number of characters to be output.

`Base` : `Number_Base` := `Default_Base`;

Specifies the radix base with which to output the form of the number.

---

**Errors**

If the file is not open, the `Io_Exceptions.Status_Error` exception is raised.

If the mode of the file is not `Out_File`, the `Io_Exceptions.Mode_Error` exception is raised.

---

```
procedure Put
package !Io.Text_Io.Integer_Io
```

## procedure Put

---

```
procedure Put (To : out String;
              Item : Num;
              Base : Number_Base := Default_Base);
```

---

### Description

Writes an integer value to a string.

This procedure outputs the value of the Item parameter to the specified string, following the same rule as for output to a file, using the length of the specified string as the value for the Width parameter.

---

### Parameters

To : out String;

Specifies the string to which the integer is to be written.

Item : Num;

Specifies the value of the integer to be written.

Base : Number\_Base := Default\_Base;

Specifies the radix base to be used in writing the number.

---

### Errors

If the length of the actual string is insufficient for the output of the item, the `Io_Exceptions.Layout_Error` exception is raised.

---

**Example**

```
package Int_Io is new Integer_Io(Small_Int); use Int_Io;
-- default format used at instantiation, Default_Width = 4,
--                                     Default_Base = 10

    Put(126);                               -- "b126"
    Put(-126, 7);                           -- "bbb-126"
    Put(126, Width => 13, Base => 2);       -- "bbb2#1111110#"

```

---

---

end Integer\_Io;

---

package !Io.Text\_Io

---

end Text\_Io;

---

## Index

This index contains entries for each unit and its declarations as well as definitions, topical cross-references, exceptions raised, errors, enumerations, pragmas, switches, and the like. The entries for each unit are arranged alphabetically by simple name. An italic page number indicates the primary reference for an entry.

<b>!Io.Device_Independent_Io</b> package . . . . .	TIO-3
Io.Convert function . . . . .	TIO-17
<b>!Io.Device_Independent_Io.File_Type</b> type	
Io.Convert function . . . . .	TIO-17
Io.Convert procedure . . . . .	TIO-18
<b>!Io.Terminal_Specific</b> package . . . . .	TIO-3
<b>!Machine.Devices.Terminal_n</b> . . . . .	TIO-3
< function	
Io.< . . . . .	TIO-10
= function	
Io.= . . . . .	TIO-9
> function	
Io.> . . . . .	TIO-11

### A

access control . . . . .	TIO-5
<b>Access_Error</b>	
Io.Exceptions.Use_Error exception . . . . .	TIO-163
add, <i>see</i> Insert	
add to end, <i>see</i> Append	
<b>Already_Open_Error</b>	
Io.Exceptions.Status_Error exception . . . . .	TIO-162
<b>Ambiguous_Name_Error</b>	
Io.Exceptions.Name_Error exception . . . . .	TIO-161

Append procedure	
Io.Append	TIO-12
ASCII characters	TIO-1
Ascii.Ff	TIO-5
Ascii.Lf	TIO-5

## B

base	
default	
Io.Integer_Io.Default_Base constant	TIO-144
Text_Io.Integer_Io.Default_Base constant	TIO-252
number	
Io.Number_Base subtype	TIO-62
Text_Io.Number_Base subtype	TIO-192
Boolean value	
read from file	
Io.Get procedure	TIO-49
write to current error file (Message window)	
Io.Echo procedure	TIO-31
write to file	
Io.Put procedure	TIO-78
buffer	
force characters to file	
Io.Flush procedure	TIO-39

## C

Capacity_Error	
Io_Exceptions.Use_Error exception	TIO-163
case	
lowercase	
Io.Lower_Case constant	TIO-57
uppercase	
Io.Upper_Case constant	TIO-104
characters	
force from buffer to file	
Io.Flush procedure	TIO-39
read current line except terminator	
Io.Get_Line function	TIO-50
read from file	
Io.Get procedure	TIO-41
Text_Io.Get procedure	TIO-181
write to current error file (Message window)	
Io.Echo procedure	TIO-26
write to file	
Io.Put procedure	TIO-72
Io.Text_Io.Put procedure	TIO-198



Check_Out_Error	
Io_Exceptions.Use_Error exception . . . . .	TIO-163
Class_Error	
Io_Exceptions.Use_Error exception . . . . .	TIO-163
Close procedure	
Io.Close . . . . .	TIO-14
Text_Io.Close . . . . .	TIO-166
Col function	
Io.Col . . . . .	TIO-15
Text_Io.Col . . . . .	TIO-167
column	
current number	
Io.Col function . . . . .	TIO-15
Text_Io.Col function . . . . .	TIO-167
number . . . . .	TIO-7
set number	
Io.Set_Col procedure . . . . .	TIO-85
Text_Io.Set_Col procedure . . . . .	TIO-203
Column_Error	
Io_Exceptions.Layout_Error exception . . . . .	TIO-159
comparison	
equal	
Io.= function . . . . .	TIO-9
greater than	
Io.> function . . . . .	TIO-11
less than	
Io.< function . . . . .	TIO-10
concurrency . . . . .	TIO-5
Convert function	
Io.Convert . . . . .	TIO-16, TIO-17
Convert procedure	
Io.Convert . . . . .	TIO-18
count	
Io.Positive_Count subtype . . . . .	TIO-71
Text_Io.Positive_Count subtype . . . . .	TIO-197
Count subtype	
Io.Count . . . . .	TIO-7, TIO-19
Count type	
Text_Io.Count . . . . .	TIO-165, TIO-168
Create procedure	
Io.Create . . . . .	TIO-20
Form function . . . . .	TIO-40
Text_Io.Create . . . . .	TIO-169
Form function . . . . .	TIO-180

Current_Error function	
Io.Current_Error . . . . .	TIO-22
Current_Input function	
Io.Current_Input . . . . .	TIO-23
Text_Io.Current_Input . . . . .	TIO-171
Current_Output function	
Io.Current_Output . . . . .	TIO-24
Text_Io.Current_Output . . . . .	TIO-172

D

Data_Error exception	
Io package	
Get procedure . . . . .	TIO-46, TIO-48, TIO-49
Io.Enumeration_Io generic package	
Get procedure . . . . .	TIO-113, TIO-114
Io.Fixed_Io generic package	
Get procedure . . . . .	TIO-124, TIO-125
Io.Float_Io generic package	
Get procedure . . . . .	TIO-136, TIO-137
Io.Integer_Io generic package	
Get procedure . . . . .	TIO-147, TIO-148
Io.Exceptions.Data_Error	TIO-156
Text_Io.Enumeration_Io generic package	
Get procedure . . . . .	TIO-221, TIO-222
Text_Io.Fixed_Io generic package	
Get procedure . . . . .	TIO-232, TIO-233
Text_Io.Float_Io generic package	
Get procedure . . . . .	TIO-244, TIO-245
Text_Io.Integer_Io generic package	
Get procedure . . . . .	TIO-255, TIO-256
decimal point	
after	
Io.Fixed_Io.Default_Aft constant . . . . .	TIO-120
Io.Float_Io.Default_Aft constant . . . . .	TIO-132
Text_Io.Fixed_Io.Default_Aft constant . . . . .	TIO-228
Text_Io.Float_Io.Default_Aft constant . . . . .	TIO-240
before	
Io.Fixed_Io.Default_Fore constant . . . . .	TIO-122
Io.Float_Io.Default_Fore constant . . . . .	TIO-134
Text_Io.Fixed_Io.Default_Fore constant . . . . .	TIO-230
Text_Io.Float_Io.Default_Fore constant . . . . .	TIO-242
<DEFAULT> . . . . .	TIO-6
default profile . . . . .	TIO-6

<b>Default_Aft constant</b>	
Io.Fixed_Io.Default_Aft . . . . .	TIO-120
Io.Float_Io.Default_Aft . . . . .	TIO-132
Text_Io.Fixed_Io.Default_Aft . . . . .	TIO-228
Text_Io.Float_Io.Default_Aft . . . . .	TIO-240
<b>Default_Base constant</b>	
Io.Integer_Io.Default_Base . . . . .	TIO-144
Text_Io.Integer_Io.Default_Base . . . . .	TIO-252
<b>Default_Exp constant</b>	
Io.Fixed_Io.Default_Exp . . . . .	TIO-121
Io.Float_Io.Default_Exp . . . . .	TIO-133
Text_Io.Fixed_Io.Default_Exp . . . . .	TIO-229
Text_Io.Float_Io.Default_Exp . . . . .	TIO-241
<b>Default_Fore constant</b>	
Io.Fixed_Io.Default_Fore . . . . .	TIO-122
Io.Float_Io.Default_Fore . . . . .	TIO-134
Text_Io.Fixed_Io.Default_Fore . . . . .	TIO-230
Text_Io.Float_Io.Default_Fore . . . . .	TIO-242
<b>Default_Setting constant</b>	
Io.Enumeration_Io.Default_Setting . . . . .	TIO-110
Text_Io.Enumeration_Io.Default_Setting . . . . .	TIO-218
<b>Default_Width constant</b>	
Io.Enumeration_Io.Default_Width . . . . .	TIO-111
Io.Integer_Io.Default_Width . . . . .	TIO-145
Text_Io.Enumeration_Io.Default_Width . . . . .	TIO-219
Text_Io.Integer_Io.Default_Width . . . . .	TIO-253
<b>Delete procedure</b>	
Io.Delete . . . . .	TIO-25
Text_Io.Delete . . . . .	TIO-173
<b>Device_Data_Error</b>	
Io_Exceptions.Device_Error exception . . . . .	TIO-157
<b>Device_Error exception</b>	
Io_Exceptions.Device_Error . . . . .	TIO-157
devices . . . . .	TIO-2
directory error, <i>see</i> Nonexistent_Directory_Error	
display, <i>see</i> Default_Font	
display image, <i>see</i> Designation, Font	

E

Echo procedure	
Io.Echo . . . . .	<i>TIO-26, TIO-27, TIO-28, TIO-29, TIO-31</i>
Echo_Line procedure	
Io.Echo_Line . . . . .	<i>TIO-32</i>
End_Error exception	
Io package	
Get procedure . . . . .	<i>TIO-41, TIO-42, TIO-44, TIO-46, TIO-48, TIO-49</i>
Get_Line procedure . . . . .	<i>TIO-50, TIO-52</i>
Set_Col procedure . . . . .	<i>TIO-86</i>
Set_Line procedure . . . . .	<i>TIO-92</i>
Skip_Line procedure . . . . .	<i>TIO-97</i>
Skip_Page procedure . . . . .	<i>TIO-98</i>
Io.Enumeration_Io generic package	
Get procedure . . . . .	<i>TIO-113</i>
Io.Fixed_Io generic package	
Get procedure . . . . .	<i>TIO-124</i>
Io.Float_Io generic package	
Get procedure . . . . .	<i>TIO-136</i>
Io.Integer_Io generic package	
Get procedure . . . . .	<i>TIO-147</i>
Io_Exceptions.End_Error . . . . .	<i>TIO-158</i>
Text_Io package	
Get procedure . . . . .	<i>TIO-181, TIO-182</i>
Get_Line procedure . . . . .	<i>TIO-184</i>
Set_Col procedure . . . . .	<i>TIO-204</i>
Set_Line procedure . . . . .	<i>TIO-207</i>
Skip_Line procedure . . . . .	<i>TIO-211</i>
Skip_Page procedure . . . . .	<i>TIO-212</i>
Text_Io.Enumeration_Io generic package	
Get procedure . . . . .	<i>TIO-221</i>
Text_Io.Fixed_Io generic package	
Get procedure . . . . .	<i>TIO-232</i>
Text_Io.Float_Io generic package	
Get procedure . . . . .	<i>TIO-244</i>
Text_Io.Integer_Io generic package	
Get procedure . . . . .	<i>TIO-255</i>
End_Of_File function	
Io.End_Of_File . . . . .	<i>TIO-33</i>
Text_Io.End_Of_File . . . . .	<i>TIO-174</i>
end-of-file terminator . . . . .	<i>TIO-5</i>
End_Of_Line function	
Io.End_Of_Line . . . . .	<i>TIO-34</i>
Text_Io.End_Of_Line . . . . .	<i>TIO-175</i>

End_Of_Page function	
Io.End_Of_Page . . . . .	TIO-35
Text_Io.End_Of_Page . . . . .	TIO-176
Enum generic formal type	
Io.Enumeration_Io.Enum . . . . .	TIO-112
Get procedure . . . . .	TIO-113, TIO-114
Text_Io.Enumeration_Io.Enum . . . . .	TIO-220
Get procedure . . . . .	TIO-221, TIO-222
enumeration literals	
Io.Lower_Case constant . . . . .	TIO-57
Io.Type_Set subtype . . . . .	TIO-102
Io.Upper_Case constant . . . . .	TIO-104
Text_Io.Type_Set type . . . . .	TIO-215
enumeration value	
read from file	
Io.Enumeration_Io.Get procedure . . . . .	TIO-113
Text_Io.Enumeration_Io.Get procedure . . . . .	TIO-221
read from string	
Io.Enumeration_Io.Get procedure . . . . .	TIO-114
Text_Io.Enumeration_Io.Get procedure . . . . .	TIO-222
write to file	
Io.Enumeration_Io.Put procedure . . . . .	TIO-115
Text_Io.Enumeration_Io.Put procedure . . . . .	TIO-223
write to string	
Io.Enumeration_Io.Put procedure . . . . .	TIO-117
Text_Io.Enumeration_Io.Put procedure . . . . .	TIO-225
Enumeration_Io generic package	
Io.Enumeration_Io . . . . .	TIO-109
Text_Io.Enumeration_Io . . . . .	TIO-217
EOF, <i>see</i> End_Error, End_Of_File	
EOL, <i>see</i> End_Of_Line	
equal	
Io.= function . . . . .	TIO-9
error	
Io.Current_Error function . . . . .	TIO-22
Io.Note_Error generic formal procedure . . . . .	TIO-106
Io.Pop_Error procedure . . . . .	TIO-68
Io.Reset_Error procedure . . . . .	TIO-81
Io.Set_Error procedure . . . . .	TIO-87
Io.Standard_Error function . . . . .	TIO-99
Io_Exceptions.Data_Error exception	
Input_Syntax_Error . . . . .	TIO-156
Input_Value_Error . . . . .	TIO-156
Output_Type_Error . . . . .	TIO-156
Output_Value_Error . . . . .	TIO-156

error, continued

Io_Exceptions.Device_Error exception	
Device_Data_Error . . . . .	TIO-157
Illegal_Heap_Access_Error . . . . .	TIO-157
Illegal_Reference_Error . . . . .	TIO-157
Page_Nonexistent_Error . . . . .	TIO-157
Write_To_Read_Only_Page_Error . . . . .	TIO-157
Io_Exceptions.Layout_Error exception	
Column_Error . . . . .	TIO-159
Illegal_Position_Error . . . . .	TIO-159
Item_Length_Error . . . . .	TIO-159
Io_Exceptions.Mode_Error exception	
Illegal_Operation_On_Infile . . . . .	TIO-160
Illegal_Operation_On_Outfile . . . . .	TIO-160
Io_Exceptions.Name_Error exception	
Ambiguous_Name_Error . . . . .	TIO-161
Illformed_Name_Error . . . . .	TIO-161
Nonexistent_Directory_Error . . . . .	TIO-161
Nonexistent_Object_Error . . . . .	TIO-161
Nonexistent_Version_Error . . . . .	TIO-161
Io_Exceptions.Status_Error exception	
Already_Open_Error . . . . .	TIO-162
Not_Open_Error . . . . .	TIO-162
Io_Exceptions.Use_Error exception	
Access_Error . . . . .	TIO-163
Capacity_Error . . . . .	TIO-163
Check_Out_Error . . . . .	TIO-163
Class_Error . . . . .	TIO-163
Frozen_Error . . . . .	TIO-163
Line_Page_Length_Error . . . . .	TIO-163
Lock_Error . . . . .	TIO-163
Reset_Error . . . . .	TIO-163
Unsupported_Error . . . . .	TIO-163

error, *see also* Bell, Unknown\_Key

error file

current	
Io.Pop_Error procedure . . . . .	TIO-68
Io.Reset_Error procedure . . . . .	TIO-81
current default	
Io.Current_Error function . . . . .	TIO-22
Io.Set_Error procedure . . . . .	TIO-87
standard . . . . .	TIO-3
Io.Standard_Error function . . . . .	TIO-99

error reactions . . . . .	TIO-6
---------------------------	-------

exceptions . . . . .	TIO-6
Io_Exceptions package	
Data_Error exception . . . . .	TIO-156
Device_Error exception . . . . .	TIO-157
End_Error exception . . . . .	TIO-158
Layout_Error exception . . . . .	TIO-159
Mode_Error exception . . . . .	TIO-160
Name_Error exception . . . . .	TIO-161
Status_Error exception . . . . .	TIO-162
Use_Error exception . . . . .	TIO-163

exponent	
Io.Fixed_Io.Default_Exp constant . . . . .	TIO-121
Io.Float_Io.Default_Exp constant . . . . .	TIO-133
Text_Io.Fixed_Io.Default_Exp constant . . . . .	TIO-229
Text_Io.Float_Io.Default_Exp constant . . . . .	TIO-241

F

Field subtype	
Io.Field . . . . .	TIO-96
Echo procedure . . . . .	TIO-28
Io.Integer_Io generic package . . . . .	TIO-143
Put procedure . . . . .	TIO-74
Text_Io.Field . . . . .	TIO-177
Text_Io.Integer_Io generic package . . . . .	TIO-251

file . . . . .	TIO-1
association	
Io.Close procedure . . . . .	TIO-14
Text_Io.Close procedure . . . . .	TIO-166
create	
Io.Create procedure . . . . .	TIO-20
Text_Io.Create procedure . . . . .	TIO-169
current default error	
Io.Current_Error function . . . . .	TIO-22
current default input	
Io.Current_Input function . . . . .	TIO-23
Text_Io.Current_Input function . . . . .	TIO-171
current default output	
Io.Current_Output function . . . . .	TIO-24
Text_Io.Current_Output function . . . . .	TIO-172
delete	
Io.Delete procedure . . . . .	TIO-25
Text_Io.Delete procedure . . . . .	TIO-173
end of	
Io.End_Of_File function . . . . .	TIO-33
Text_Io.End_Of_File function . . . . .	TIO-174
force characters to	
Io.Flush procedure . . . . .	TIO-39

file, continued	
handle	TIO-4
Io package	TIO-7
Io.File_Type type	TIO-38
in	
Io.In_File constant	TIO-53
name	TIO-4
Io.Name function	TIO-59
Text_Io.Name function	TIO-189
out	
Io.Out_File constant	TIO-65
processing multiple files	
Io.Wildcard_Iterator generic procedure	TIO-105
read Boolean value from	
Io.Get procedure	TIO-49
read character from	
Io.Get procedure	TIO-41
Text_Io.Get procedure	TIO-181
read enumeration value from	
Io.Enumeration_Io.Get procedure	TIO-113
Text_Io.Enumeration_Io.Get procedure	TIO-221
read fixed-point value from	
Io.Fixed_Io.Get procedure	TIO-123
Text_Io.Fixed_Io.Get procedure	TIO-231
read floating-point value from	
Io.Float_Io.Get procedure	TIO-135
Io.Get procedure	TIO-47
Text_Io.Float_Io.Get procedure	TIO-243
read integer value from	
Io.Get procedure	TIO-45
Io.Integer_Io.Get procedure	TIO-146
Text_Io.Integer_Io.Get procedure	TIO-254
read line from	
Io.Get procedure	TIO-43
read remaining characters except terminator	
Io.Get_Line function	TIO-50
read string from	
Io.Get procedure	TIO-42
Text_Io.Get procedure	TIO-182
read string from single line except terminator	
Io.Get_Line procedure	TIO-51
Text_Io.Get_Line procedure	TIO-183
read-only access	
Io.File_Mode subtype	TIO-37
Text_Io.File_Mode type	TIO-178



file, continued

stack	TIO-8
Io.Pop_Error procedure	TIO-68
Io.Pop_Input procedure	TIO-69
Io.Pop_Output procedure	TIO-70
Io.Reset_Error procedure	TIO-81
Io.Reset_Input procedure	TIO-82
Io.Reset_Output procedure	TIO-83
Io.Set_Error procedure	TIO-87
Io.Set_Input procedure	TIO-89
Io.Set_Output procedure	TIO-94
standard error	TIO-7
Io.Standard_Error function	TIO-99
standard input	TIO-3, TIO-7
Io.Standard_Input function	TIO-100
Text_Io.Standard_Input function	TIO-213
standard output	TIO-3, TIO-7
Io.Standard_Output function	TIO-101
Text_Io.Standard_Output function	TIO-214
storage	TIO-2
text	TIO-7
wildcards	
Io.Wildcard_Iterator generic procedure	TIO-105
write Boolean value to	
Io.Put procedure	TIO-78
write character to	
Io.Put procedure	TIO-72
Io.Text_Io.Put procedure	TIO-198
write enumeration value to	
Io.Enumeration_Io.Put procedure	TIO-115
Text_Io.Enumeration_Io.Put procedure	TIO-223
write fixed-point value to	
Io.Fixed_Io.Put procedure	TIO-127
Text_Io.Fixed_Io.Put procedure	TIO-235
write floating-point value to	
Io.Float_Io.Put procedure	TIO-139
Io.Put procedure	TIO-76
Text_Io.Float_Io.Put procedure	TIO-247
write integer value to	
Io.Integer_Io.Put procedure	TIO-150
Io.Put procedure	TIO-74
Text_Io.Integer_Io.Put procedure	TIO-258
write string to	
Io.Put procedure	TIO-73
Text_Io.Put procedure	TIO-199
write string to/advance line	
Io.Put_Line procedure	TIO-79
Text_Io.Put_Line procedure	TIO-200

file, continued	
write-only access	
Io.File_Mode subtype . . . . .	TIO-37
Text_Io.File_Mode type . . . . .	TIO-178
file handle, get, <i>see</i> Open	
file stack	
Io package . . . . .	TIO-7
File_Mode subtype	
Io.File_Mode . . . . .	TIO-37
File_Mode type	
Text_Io.File_Mode . . . . .	TIO-178
File_Type type	
Io.File_Type . . . . .	TIO-7, TIO-38
Convert function . . . . .	TIO-16, TIO-17
Text_Io.File_Type . . . . .	TIO-165, TIO-179
Io.Convert function . . . . .	TIO-16
Io.Convert procedure . . . . .	TIO-18
fixed-point type, <i>see</i> Num	
fixed-point value	
read from file	
Io.Fixed_Io.Get procedure . . . . .	TIO-123
Text_Io.Fixed_Io.Get procedure . . . . .	TIO-231
read from string	
Io.Fixed_Io.Get procedure . . . . .	TIO-125
Text_Io.Fixed_Io.Get procedure . . . . .	TIO-233
write to file	
Io.Fixed_Io.Put procedure . . . . .	TIO-127
Text_Io.Fixed_Io.Put procedure . . . . .	TIO-235
write to string	
Io.Fixed_Io.Put procedure . . . . .	TIO-129
Text_Io.Fixed_Io.Put procedure . . . . .	TIO-237
Fixed_Io generic package	
Io.Fixed_Io . . . . .	TIO-119
Text_Io.Fixed_Io . . . . .	TIO-227
Float_Io generic package	
Io.Float_Io . . . . .	TIO-131
Text_Io.Float_Io . . . . .	TIO-239
floating-point value	
read from file	
Io.Float_Io.Get procedure . . . . .	TIO-135
Io.Get procedure . . . . .	TIO-47
Text_Io.Float_Io.Get procedure . . . . .	TIO-243
read from string	
Io.Float_Io.Get procedure . . . . .	TIO-137
Text_Io.Float_Io.Get procedure . . . . .	TIO-245

floating-point value, continued	
write to current error file (Message window)	
Io.Echo procedure . . . . .	TIO-29
write to file	
Io.Float_Io.Put procedure . . . . .	TIO-139
Io.Put procedure . . . . .	TIO-76
Text_Io.Float_Io.Put procedure . . . . .	TIO-247
write to string	
Io.Float_Io.Put procedure . . . . .	TIO-141
Text_Io.Float_Io.Put procedure . . . . .	TIO-249
Flush procedure	
Io.Flush . . . . .	TIO-39
Convert function . . . . .	TIO-17
Save procedure . . . . .	TIO-84
Form function	
Io.Form . . . . .	TIO-40
Text_Io.Form . . . . .	TIO-180
Frozen_Error	
Io.Exceptions.Use_Error exception . . . . .	TIO-163

G

Get procedure	
Io.Enumeration_Io.Get . . . . .	TIO-113, TIO-114
Io.Fixed_Io.Get . . . . .	TIO-123, TIO-125
Io.Float_Io.Get . . . . .	TIO-135, TIO-137
Io.Get . . . . .	TIO-41, TIO-42, TIO-43, TIO-45, TIO-47, TIO-49
Set_Col procedure . . . . .	TIO-85
Io.Integer_Io.Get . . . . .	TIO-146, TIO-148
Text_Io.Enumeration_Io.Get . . . . .	TIO-221, TIO-222
Text_Io.Fixed_Io.Get . . . . .	TIO-231, TIO-233
Text_Io.Float_Io.Get . . . . .	TIO-243, TIO-245
Text_Io.Get . . . . .	TIO-181, TIO-182
Set_Col procedure . . . . .	TIO-203
Text_Io.Integer_Io.Get . . . . .	TIO-254, TIO-256
Get_Line function	
Io.Get_Line . . . . .	TIO-50
Get_Line procedure	
Io.Get_Line . . . . .	TIO-51
Set_Col procedure . . . . .	TIO-85
Text_Io.Get_Line . . . . .	TIO-183
Set_Col procedure . . . . .	TIO-203
greater than	
Io.> function . . . . .	TIO-11

H

handle, file . . . . .	TIO-4
Io package . . . . .	TIO-7
Io.File_Type type . . . . .	TIO-38
handle, file, get, <i>see</i> Open	
hardware error, <i>see</i> Device_Data_Error	

I

I/O window	
Io.Current_Input function . . . . .	TIO-23
Io.Current_Output function . . . . .	TIO-24
Io.Standard_Input function . . . . .	TIO-100
Io.Standard_Output function . . . . .	TIO-101
Text_Io.Current_Input function . . . . .	TIO-171
Text_Io.Current_Output function . . . . .	TIO-172
Text_Io.Standard_Input function . . . . .	TIO-213
Text_Io.Standard_Output function . . . . .	TIO-214
Illegal_Heap_Access_Error	
Io_Exceptions.Device_Error exception . . . . .	TIO-157
Illegal_Operation_On_Infile	
Io_Exceptions.Mode_Error exception . . . . .	TIO-160
Illegal_Operation_On_Outfile	
Io_Exceptions.Mode_Error exception . . . . .	TIO-160
Illegal_Position_Error	
Io_Exceptions.Layout_Error exception . . . . .	TIO-159
Illegal_Reference_Data_Error	
Io_Exceptions.Device_Error exception . . . . .	TIO-157
Illformed_Name_Error	
Io_Exceptions.Name_Error exception . . . . .	TIO-161
image, display, <i>see</i> Designation, Font	
In_File constant	
Io.In_File . . . . .	TIO-53
input file . . . . .	TIO-3
current	
Io.Pop_Input procedure . . . . .	TIO-69
Io.Reset_Input procedure . . . . .	TIO-82
current default	
Io.Current_Input function . . . . .	TIO-23
Io.Set_Input procedure . . . . .	TIO-89
Text_Io.Current_Input function . . . . .	TIO-171
Text_Io.Set_Input procedure . . . . .	TIO-205

input file, continued	
standard	
Io.Standard_Input function . . . . .	TIO-100
Text_Io.Standard_Input function . . . . .	TIO-213
input window	
Io.Current_Input function . . . . .	TIO-23
Io.Standard_Input function . . . . .	TIO-100
Text_Io.Current_Input function . . . . .	TIO-171
Text_Io.Standard_Input function . . . . .	TIO-213
Input_Syntax_Error	
Io_Exceptions.Data_Error exception . . . . .	TIO-156
Input_Value_Error	
Io_Exceptions.Data_Error exception . . . . .	TIO-156
integer value	
read from file	
Io.Get procedure . . . . .	TIO-45
Io.Integer_Io.Get procedure . . . . .	TIO-146
Text_Io.Integer_Io.Get procedure . . . . .	TIO-254
read from string	
Io.Integer_Io.Get procedure . . . . .	TIO-148
Text_Io.Integer_Io.Get procedure . . . . .	TIO-256
write to current error file (Message window)	
Io.Echo procedure . . . . .	TIO-28
write to file	
Io.Integer_Io.Put procedure . . . . .	TIO-150
Io.Put procedure . . . . .	TIO-74
Text_Io.Integer_Io.Put procedure . . . . .	TIO-258
write to string	
Io.Integer_Io.Put procedure . . . . .	TIO-152
Text_Io.Integer_Io.Put procedure . . . . .	TIO-260
Integer_Io generic package	
Io.Integer_Io . . . . .	TIO-143
Text_Io.Integer_Io . . . . .	TIO-251
Io package . . . . .	TIO-7
Io_Exceptions package . . . . .	TIO-6, TIO-155
Is_Open function	
Io.Is_Open . . . . .	TIO-54
Text_Io.Is_Open . . . . .	TIO-185
Item_Length_Error	
Io_Exceptions.Layout_Error exception . . . . .	TIO-159
iterator, wildcard	
Io.Wildcard_Iterator generic procedure . . . . .	TIO-105
Io.Wildcard_Iterator procedure . . . . .	TIO-108

	<b>J</b>	
job . . . . .		TIO-5
job response profile . . . . .		TIO-6
	<b>K</b>	
key concepts . . . . .		TIO-1
	<b>L</b>	
Layout_Error exception		
Io package		
Col function . . . . .		TIO-15
Line function . . . . .		TIO-55
Page function . . . . .		TIO-66
Set_Col procedure . . . . .		TIO-86
Set_Line procedure . . . . .		TIO-92
Io.Enumeration_Io generic package		
Put procedure . . . . .		TIO-117
Io.Fixed_Io generic package		
Put procedure . . . . .	TIO-129, TIO-130	
Io.Float_Io generic package		
Put procedure . . . . .		TIO-141
Io.Integer_Io generic package		
Put procedure . . . . .		TIO-152
Io_Exceptions.Layout_Error . . . . .		TIO-159
Text_Io package		
Col function . . . . .		TIO-167
Line function . . . . .		TIO-186
Page function . . . . .		TIO-195
Set_Col procedure . . . . .		TIO-204
Set_Line procedure . . . . .		TIO-207
Text_Io.Enumeration_Io generic package		
Put procedure . . . . .		TIO-225
Text_Io.Fixed_Io generic package		
Put procedure . . . . .		TIO-237
Text_Io.Float_Io generic package		
Put procedure . . . . .		TIO-249
Text_Io.Integer_Io generic package		
Put procedure . . . . .		TIO-260
length		
line . . . . .		TIO-8
Io.Line_Length function . . . . .		TIO-56
Io.Set_Line_Length procedure . . . . .		TIO-93
Io.Unbounded constant . . . . .		TIO-103
Text_Io.Line_Length function . . . . .		TIO-187
Text_Io.Set_Line_Length procedure . . . . .		TIO-208
Text_Io.Unbounded constant . . . . .		TIO-216

length, continued	
page	TIO-8
Io.Page_Length function	TIO-67
Io.Set_Page_Length procedure	TIO-96
Io.Unbounded constant	TIO-103
Text_Io.Page_Length function	TIO-196
Text_Io.Set_Page_Length procedure	TIO-210
Text_Io.Unbounded constant	TIO-216
length error, <i>see</i> Item_Length_Error	
less than	
Io.< function	TIO-10
line	TIO-7
current number	
Io.Line function	TIO-55
Io.Set_Line procedure	TIO-91
Text_Io.Line function	TIO-186
Text_Io.Set_Line procedure	TIO-206
echo	
Io.Echo_Line procedure	TIO-32
end of	
Io.End_Of_Line function	TIO-34
Text_Io.End_Of_Line function	TIO-175
get	
Io.Get_Line function	TIO-50
Io.Get_Line procedure	TIO-51
Text_Io.Get_Line procedure	TIO-183
length	TIO-8
Io.Set_Line_Length procedure	TIO-93
Io.Unbounded constant	TIO-103
Text_Io.Set_Line_Length procedure	TIO-208
Text_Io.Unbounded constant	TIO-216
new	
Io.New_Line procedure	TIO-60
Text_Io.New_Line procedure	TIO-190
put	
Io.Put_Line procedure	TIO-79
Text_Io.Put_Line procedure	TIO-200
read from file	
Io.Get procedure	TIO-43
set	
Io.Set_Line procedure	TIO-91
Text_Io.Set_Line procedure	TIO-206
skip	
Io.Skip_Line procedure	TIO-97
Text_Io.Skip_Line procedure	TIO-211
terminator (Ascii.Lf)	TIO-5

Line function	
Io.Line . . . . .	TIO-55
Text_Io.Line . . . . .	TIO-186
Line_Length function	
Io.Line_Length . . . . .	TIO-56
Text_Io.Line_Length . . . . .	TIO-187
Line_Page_Length_Error	
Io_Exceptions.Use_Error exception . . . . .	TIO-163
Lock_Error	
Io_Exceptions.Use_Error exception . . . . .	TIO-163
Lower_Case constant	
Io.Lower_Case . . . . .	TIO-57

M

make changes permanent	
Io.Save procedure . . . . .	TIO-84
merging files, <i>see</i> Append	
Message window	
Io package . . . . .	TIO-7
write to	
Io.Current_Error function . . . . .	TIO-22
Io.Echo procedure . . . . .	TIO-26, TIO-27, TIO-28, TIO-29, TIO-31
Io.Echo_Line procedure . . . . .	TIO-32
Io.Standard_Error function . . . . .	TIO-99
Mode function	
Io.Mode . . . . .	TIO-58
Text_Io.Mode . . . . .	TIO-188
mode, file	
Io.File_Mode subtype . . . . .	TIO-37
Text_Io.File_Mode type . . . . .	TIO-178
Mode_Error exception	
Io package	
End_Of_File function . . . . .	TIO-33
End_Of_Line function . . . . .	TIO-34
End_Of_Page function . . . . .	TIO-35
Get procedure . . . . .	TIO-41, TIO-42, TIO-44, TIO-46, TIO-48, TIO-49
Get_Line procedure . . . . .	TIO-50, TIO-52
Line_Length function . . . . .	TIO-56
New_Line procedure . . . . .	TIO-60
New_Page procedure . . . . .	TIO-61
Page_Length function . . . . .	TIO-67
Put procedure . . . . .	TIO-72, TIO-73, TIO-75, TIO-77, TIO-78
Put_Line procedure . . . . .	TIO-79
Reset procedure . . . . .	TIO-80



Mode_Error exception, continued	
Io package, continued	
Set_Error procedure . . . . .	TIO-88
Set_Input procedure . . . . .	TIO-90
Set_Line_Length procedure . . . . .	TIO-93
Set_Output procedure . . . . .	TIO-95
Set_Page_Length procedure . . . . .	TIO-96
Skip_Line procedure . . . . .	TIO-97
Skip_Page procedure . . . . .	TIO-98
Io.Enumeration_Io generic package	
Get procedure . . . . .	TIO-113
Put procedure . . . . .	TIO-116
Io.Fixed_Io generic package	
Get procedure . . . . .	TIO-124
Put procedure . . . . .	TIO-128
Io.Float_Io generic package	
Get procedure . . . . .	TIO-136
Put procedure . . . . .	TIO-140
Io.Integer_Io generic package	
Get procedure . . . . .	TIO-147
Put procedure . . . . .	TIO-151
Io.Exceptions.Mode_Error	TIO-160
Text_Io package	
End_Of_File function . . . . .	TIO-174
End_Of_Line function . . . . .	TIO-175
End_Of_Page function . . . . .	TIO-176
Get procedure . . . . .	TIO-181, TIO-182
Get_Line procedure . . . . .	TIO-184
Line_Length function . . . . .	TIO-187
New_Line procedure . . . . .	TIO-190
New_Page procedure . . . . .	TIO-191
Page_Length function . . . . .	TIO-196
Put procedure . . . . .	TIO-198, TIO-199
Put_Line procedure . . . . .	TIO-200
Reset procedure . . . . .	TIO-202
Set_Input procedure . . . . .	TIO-205
Set_Line_Length procedure . . . . .	TIO-208
Set_Output procedure . . . . .	TIO-209
Set_Page_Length procedure . . . . .	TIO-210
Skip_Line procedure . . . . .	TIO-211
Skip_Page procedure . . . . .	TIO-212
Text_Io.Enumeration_Io generic package	
Get procedure . . . . .	TIO-221
Put procedure . . . . .	TIO-224
Text_Io.Fixed_Io generic package	
Get procedure . . . . .	TIO-232
Put procedure . . . . .	TIO-236
Text_Io.Float_Io generic package	
Get procedure . . . . .	TIO-244
Put procedure . . . . .	TIO-248

Mode_Error exception, continued	
Text_Io.Integer_Io generic package	
Get procedure . . . . .	TIO-255
Put procedure . . . . .	TIO-259
multiple files, processing	
Io.Wildcard_Iterator generic procedure . . . . .	TIO-105

N

name error, *see* Ambiguous\_Name\_Error, Illformed\_Name\_Error

Name function	
Io.Name . . . . .	TIO-59
Text_Io.Name . . . . .	TIO-189

Name_Error exception	
Io package	
Append procedure . . . . .	TIO-13
Create procedure . . . . .	TIO-21
Open procedure . . . . .	TIO-64
Set_Error procedure . . . . .	TIO-88
Set_Input procedure . . . . .	TIO-90
Set_Output procedure . . . . .	TIO-95
Io_Exceptions.Name_Error . . . . .	TIO-161
Text_Io package	
Create procedure . . . . .	TIO-170
Open procedure . . . . .	TIO-194

naming files . . . . .	TIO-4
------------------------	-------

New_Line procedure	
Io.New_Line . . . . .	TIO-60
Echo procedure . . . . .	TIO-26
Echo_Line procedure . . . . .	TIO-32
Put procedure . . . . .	TIO-72
Put_Line procedure . . . . .	TIO-79
Set_Col procedure . . . . .	TIO-85
Set_Line procedure . . . . .	TIO-91
Text_Io.New_Line . . . . .	TIO-190
Put procedure . . . . .	TIO-198
Put_Line procedure . . . . .	TIO-200
Set_Col procedure . . . . .	TIO-203
Set_Line procedure . . . . .	TIO-206

New_Page procedure	
Io.New_Page . . . . .	TIO-61
Close procedure . . . . .	TIO-14
Set_Line procedure . . . . .	TIO-91
Text_Io.New_Page . . . . .	TIO-191
Close procedure . . . . .	TIO-166
Reset procedure . . . . .	TIO-201
Set_Line procedure . . . . .	TIO-206

Nonexistent_Directory_Error		
Io_Exceptions.Name_Error exception . . . . .		TIO-161
Nonexistent_Object_Error		
Io_Exceptions.Name_Error exception . . . . .		TIO-161
Nonexistent_Version_Error		
Io_Exceptions.Name_Error exception . . . . .		TIO-161
Not_Open_Error		
Io_Exceptions.Status_Error exception . . . . .		TIO-162
Note_Error generic formal procedure		
Io.Note_Error . . . . .		TIO-106
Io.Wildcard_Iterator generic procedure . . . . .		TIO-105
Io.Wildcard_Iterator procedure . . . . .		TIO-108
Num generic formal type		
Io.Fixed_Io.Num . . . . .		TIO-126
Get procedure . . . . .	TIO-123, TIO-124,	TIO-125
Io.Float_Io.Num . . . . .		TIO-138
Get procedure . . . . .	TIO-135, TIO-136,	TIO-137
Io.Integer_Io.Num . . . . .		TIO-149
Get procedure . . . . .	TIO-146, TIO-147,	TIO-148
Text_Io.Fixed_Io.Num . . . . .		TIO-234
Get procedure . . . . .	TIO-231, TIO-232,	TIO-233
Text_Io.Float_Io.Num . . . . .		TIO-246
Get procedure . . . . .	TIO-243,	TIO-245
Text_Io.Integer_Io.Num . . . . .		TIO-257
Get procedure . . . . .	TIO-254, TIO-255,	TIO-256
Number_Base subtype		
Io.Number_Base . . . . .		TIO-62
Echo procedure . . . . .		TIO-28
Io.Integer_Io generic package . . . . .		TIO-143
Put procedure . . . . .		TIO-74
Text_Io.Number_Base . . . . .		TIO-192
Text_Io.Integer_Io generic package . . . . .		TIO-251

O

object error, *see* Nonexistent\_Object\_Error

open

Io.Is_Open function . . . . .		TIO-54
Text_Io.Is_Open function . . . . .		TIO-185

open error, *see* Already\_Open\_Error, Not\_Open\_Error

Open procedure

Io.Open . . . . .		TIO-63
Form function . . . . .		TIO-40
Text_Io.Open . . . . .		TIO-193
Form function . . . . .		TIO-180

Out_File constant	
Io.Out_File . . . . .	TIO-65
output file . . . . .	TIO-3
current	
Io.Pop_Output procedure . . . . .	TIO-70
Io.Reset_Output procedure . . . . .	TIO-83
current default	
Io.Current_Output function . . . . .	TIO-24
Io.Set_Output procedure . . . . .	TIO-94
Text_Io.Current_Output function . . . . .	TIO-172
Text_Io.Set_Output procedure . . . . .	TIO-209
standard	
Io.Standard_Output function . . . . .	TIO-101
Text_Io.Standard_Output function . . . . .	TIO-214
output window	
Io.Current_Output function . . . . .	TIO-24
Io.Standard_Output function . . . . .	TIO-101
Text_Io.Current_Output function . . . . .	TIO-172
Text_Io.Standard_Output function . . . . .	TIO-214
Output_Type_Error	
Io_Exceptions.Data_Error exception . . . . .	TIO-156
Output_Value_Error	
Io_Exceptions.Data_Error exception . . . . .	TIO-156
	P
page . . . . .	TIO-7
current number	
Io.Page function . . . . .	TIO-66
Text_Io.Page function . . . . .	TIO-195
end of	
Io.End_Of_Page function . . . . .	TIO-35
Text_Io.End_Of_Page function . . . . .	TIO-176
length . . . . .	TIO-8
Io.Set_Page_Length procedure . . . . .	TIO-96
Io.Unbounded constant . . . . .	TIO-103
Text_Io.Set_Page_Length procedure . . . . .	TIO-210
Text_Io.Unbounded constant . . . . .	TIO-216
new	
Io.New_Page procedure . . . . .	TIO-61
Text_Io.New_Page procedure . . . . .	TIO-191
skip	
Io.Skip_Page procedure . . . . .	TIO-98
Text_Io.Skip_Page procedure . . . . .	TIO-212
terminator (Ascii.Ff) . . . . .	TIO-5
Page function	
Io.Page . . . . .	TIO-66
Text_Io.Page . . . . .	TIO-195

page length error, <i>see</i> Line_Page_Length_Error	
Page_Length function	
Io.Page_Length . . . . .	TIO-67
Text_Io.Page_Length . . . . .	TIO-196
Page_Nonexistent_Error	
Io_Exceptions.Device_Error exception . . . . .	TIO-157
permanent, make changes	
Io.Save procedure . . . . .	TIO-84
pop . . . . .	TIO-8
Pop_Error procedure	
Io.Pop_Error . . . . .	TIO-68
Pop_Input procedure	
Io.Pop_Input . . . . .	TIO-69
Pop_Output procedure	
Io.Pop_Output . . . . .	TIO-70
position, <i>see</i> Column_Number, Index, Set_Index	
Positive_Count subtype	
Io.Positive_Count . . . . .	TIO-7, TIO-71
Text_Io.Positive_Count . . . . .	TIO-165, TIO-197
<PROFILE> . . . . .	TIO-6
Process generic formal procedure	
Io.Process . . . . .	TIO-107
Io.Wildcard_Iterator generic procedure . . . . .	TIO-105
Io.Wildcard_Iterator procedure . . . . .	TIO-108
put, <i>see</i> Write	
Put procedure	
Io.Enumeration_Io.Put . . . . .	TIO-115, TIO-117
Io.Fixed_Io.Put . . . . .	TIO-127, TIO-129
Io.Float_Io.Put . . . . .	TIO-139, TIO-141
Io.Integer_Io.Put . . . . .	TIO-150, TIO-152
Io.Put . . . . .	TIO-72, TIO-73, TIO-74, TIO-76, TIO-78
Echo_Line procedure . . . . .	TIO-32
Put_Line procedure . . . . .	TIO-79
Text_Io.Enumeration_Io.Put . . . . .	TIO-223, TIO-225
Text_Io.Fixed_Io.Put . . . . .	TIO-235, TIO-237
Text_Io.Float_Io.Put . . . . .	TIO-247, TIO-249
Text_Io.Integer_Io.Put . . . . .	TIO-258, TIO-260
Text_Io.Put . . . . .	TIO-198, TIO-199
Put_Line procedure . . . . .	TIO-200

Put_Line procedure	
Io.Put_Line . . . . .	TIO-79
Note_Error generic formal procedure . . . . .	TIO-106
Text_Io.Put_Line . . . . .	TIO-200

R

read	
open to	
Io.In_File constant . . . . .	TIO-53
read, <i>see also</i> Get, Get_Line	
read-only access	
Io.File_Mode subtype . . . . .	TIO-37
Text_Io.File_Mode type . . . . .	TIO-178
remove, <i>see</i> Delete	
reset . . . . .	TIO-8
Reset procedure	
Io.Reset . . . . .	TIO-80
Text_Io.Reset . . . . .	TIO-201
Reset_Error	
Io.Exceptions.Use_Error exception . . . . .	TIO-163
Reset_Error procedure	
Io.Reset_Error . . . . .	TIO-81
Reset_Input procedure	
Io.Reset_Input . . . . .	TIO-82
Reset_Output procedure	
Io.Reset_Output . . . . .	TIO-83

S

Save procedure	
Io.Save . . . . .	TIO-84
screen	
input/output . . . . .	TIO-3
<SESSION> . . . . .	TIO-6
session response profile . . . . .	TIO-6
set position, <i>see</i> Set_Index	
Set_Col procedure	
Io.Set_Col . . . . .	TIO-85
Text_Io.Set_Col . . . . .	TIO-203

Set_Error procedure	
Io.Set_Error . . . . .	TIO-8, TIO-87
Pop_Error procedure . . . . .	TIO-68
Reset_Error procedure . . . . .	TIO-81
Set_Input procedure	
Io.Set_Input . . . . .	TIO-8, TIO-89
Pop_Input procedure . . . . .	TIO-69
Reset_Input procedure . . . . .	TIO-82
Text_Io.Set_Input . . . . .	TIO-205
Set_Line procedure	
Io.Set_Line . . . . .	TIO-91
Text_Io.Set_Line . . . . .	TIO-206
Set_Line_Length procedure	
Io.Set_Line_Length . . . . .	TIO-93
Text_Io.Set_Line_Length . . . . .	TIO-208
Set_Output procedure	
Io.Set_Output . . . . .	TIO-8, TIO-94
Pop_Output procedure . . . . .	TIO-70
Reset_Output procedure . . . . .	TIO-83
Text_Io.Set_Output . . . . .	TIO-209
Set_Page_Length procedure	
Io.Set_Page_Length . . . . .	TIO-96
Text_Io.Set_Page_Length . . . . .	TIO-210
setting, default	
Io.Enumeration_Io.Default_Setting constant . . . . .	TIO-110
Text_Io.Enumeration_Io.Default_Setting constant . . . . .	TIO-218
Skip_Line procedure	
Io.Skip_Line . . . . .	TIO-97
Get_Line procedure . . . . .	TIO-51
Set_Line procedure . . . . .	TIO-91
Text_Io.Skip_Line . . . . .	TIO-211
Get_Line procedure . . . . .	TIO-183
Set_Line procedure . . . . .	TIO-206
Skip_Page procedure	
Io.Skip_Page . . . . .	TIO-98
Text_Io.Skip_Page . . . . .	TIO-212
special names . . . . .	TIO-4
stack . . . . .	TIO-8

stack, file	
Io.Pop_Error procedure . . . . .	TIO-68
Io.Pop_Input procedure . . . . .	TIO-69
Io.Pop_Output procedure . . . . .	TIO-70
Io.Reset_Error procedure . . . . .	TIO-81
Io.Reset_Input procedure . . . . .	TIO-82
Io.Reset_Output procedure . . . . .	TIO-83
Io.Set_Error procedure . . . . .	TIO-87
Io.Set_Input procedure . . . . .	TIO-89
Io.Set_Output procedure . . . . .	TIO-94
standard error file . . . . .	TIO-3, TIO-7
Standard_Error function	
Io.Standard_Error . . . . .	TIO-99
standard input file . . . . .	TIO-3, TIO-7, TIO-165
Standard_Input function	
Io.Standard_Input . . . . .	TIO-100
Text_Io.Standard_Input . . . . .	TIO-219
standard output file . . . . .	TIO-3, TIO-7, TIO-165
Standard_Output function	
Io.Standard_Output . . . . .	TIO-101
Text_Io.Standard_Output . . . . .	TIO-214
Status_Error exception	
Io package	
Append procedure . . . . .	TIO-13
Close procedure . . . . .	TIO-14
Col function . . . . .	TIO-15
Create procedure . . . . .	TIO-21
Delete procedure . . . . .	TIO-25
End_Of_File function . . . . .	TIO-33
End_Of_Line function . . . . .	TIO-34
End_Of_Page function . . . . .	TIO-35
Form function . . . . .	TIO-40
Get procedure . . . . .	TIO-41, TIO-42, TIO-44, TIO-46, TIO-48, TIO-49
Get_Line procedure . . . . .	TIO-50, TIO-52
Line function . . . . .	TIO-55
Line_Length function . . . . .	TIO-56
Mode function . . . . .	TIO-58
Name function . . . . .	TIO-59
New_Line procedure . . . . .	TIO-60
New_Page procedure . . . . .	TIO-61
Open procedure . . . . .	TIO-64
Page function . . . . .	TIO-66
Page_Length function . . . . .	TIO-67
Put procedure . . . . .	TIO-72, TIO-73, TIO-75, TIO-77, TIO-78
Put_Line procedure . . . . .	TIO-79



Status\_Error exception, continued

Io package, continued

Reset procedure . . . . .	TIO-80
Set_Col procedure . . . . .	TIO-86
Set_Error procedure . . . . .	TIO-88
Set_Input procedure . . . . .	TIO-90
Set_Line procedure . . . . .	TIO-92
Set_Line_Length procedure . . . . .	TIO-93
Set_Output procedure . . . . .	TIO-95
Set_Page_Length procedure . . . . .	TIO-96
Skip_Line procedure . . . . .	TIO-97
Skip_Page procedure . . . . .	TIO-98
Io.Enumeration_Io generic package	
Get procedure . . . . .	TIO-113
Put procedure . . . . .	TIO-116
Io.Fixed_Io generic package	
Get procedure . . . . .	TIO-124
Put procedure . . . . .	TIO-128
Io.Float_Io generic package	
Get procedure . . . . .	TIO-136
Put procedure . . . . .	TIO-140
Io.Integer_Io generic package	
Get procedure . . . . .	TIO-147
Put procedure . . . . .	TIO-151
Io.Exceptions.Status_Error . . . . .	TIO-162
Text_Io package	
Close procedure . . . . .	TIO-166
Col function . . . . .	TIO-167
Create procedure . . . . .	TIO-170
Delete procedure . . . . .	TIO-173
End_Of_File function . . . . .	TIO-174
End_Of_Line function . . . . .	TIO-175
End_Of_Page function . . . . .	TIO-176
Get procedure . . . . .	TIO-181, TIO-182
Get_Line procedure . . . . .	TIO-184
Line function . . . . .	TIO-186
Line_Length function . . . . .	TIO-187
Mode function . . . . .	TIO-188
Name function . . . . .	TIO-189
New_Line procedure . . . . .	TIO-190
New_Page procedure . . . . .	TIO-191
Open procedure . . . . .	TIO-194
Page function . . . . .	TIO-195
Page_Length function . . . . .	TIO-196
Put procedure . . . . .	TIO-198, TIO-199
Put_Line procedure . . . . .	TIO-200
Reset procedure . . . . .	TIO-202
Set_Col procedure . . . . .	TIO-204
Set_Input procedure . . . . .	TIO-205
Set_Line procedure . . . . .	TIO-207

Status_Error exception, continued	
Text_Io package, continued	
Set_Line_Length procedure . . . . .	TIO-208
Set_Output procedure . . . . .	TIO-209
Set_Page_Length procedure . . . . .	TIO-210
Skip_Line procedure . . . . .	TIO-211
Skip_Page procedure . . . . .	TIO-212
Text_Io.Enumeration_Io generic package	
Get procedure . . . . .	TIO-221
Put procedure . . . . .	TIO-224
Text_Io.Fixed_Io generic package	
Get procedure . . . . .	TIO-232
Put procedure . . . . .	TIO-236
Text_Io.Float_Io generic package	
Get procedure . . . . .	TIO-244
Put procedure . . . . .	TIO-248
Text_Io.Integer_Io generic package	
Get procedure . . . . .	TIO-255
Put procedure . . . . .	TIO-259
strings	
read enumeration value from	
Io.Enumeration_Io.Get procedure . . . . .	TIO-114
Text_Io.Enumeration_Io.Get procedure . . . . .	TIO-222
read fixed-point value from	
Io.Fixed_Io.Get procedure . . . . .	TIO-125
Text_Io.Fixed_Io.Get procedure . . . . .	TIO-233
read floating-point value from	
Io.Float_Io.Get procedure . . . . .	TIO-137
Text_Io.Float_Io.Get procedure . . . . .	TIO-245
read from file	
Io.Get procedure . . . . .	TIO-42
Text_Io.Get procedure . . . . .	TIO-182
read integer value from	
Io.Integer_Io.Get procedure . . . . .	TIO-148
Text_Io.Integer_Io.Get procedure . . . . .	TIO-256
read single line except terminator	
Io.Get_Line procedure . . . . .	TIO-51
Text_Io.Get_Line procedure . . . . .	TIO-183
write enumeration value to	
Io.Enumeration_Io.Put procedure . . . . .	TIO-117
Text_Io.Enumeration_Io.Put procedure . . . . .	TIO-225
write fixed-point value to	
Io.Fixed_Io.Put procedure . . . . .	TIO-129
Text_Io.Fixed_Io.Put procedure . . . . .	TIO-237
write floating-point value to	
Io.Float_Io.Put procedure . . . . .	TIO-141
Text_Io.Float_Io.Put procedure . . . . .	TIO-249

strings, continued	
write integer value to	
Io.Integer_Io.Put procedure . . . . .	TIO-152
Text_Io.Integer_Io.Put procedure . . . . .	TIO-260
write to current error file (Message window)	
Io.Echo procedure . . . . .	TIO-27
write to current error file (Message window)/advance line	
Io.Echo_Line procedure . . . . .	TIO-32
write to file	
Io.Put procedure . . . . .	TIO-73
Text_Io.Put procedure . . . . .	TIO-199
write to file/advance line	
Io.Put_Line procedure . . . . .	TIO-79
Text_Io.Put_Line procedure . . . . .	TIO-200
synchronization . . . . .	TIO-5
syntax error, <i>see</i> Input_Syntax_Error	

T

tapes . . . . .	TIO-2
terminal	
input/output . . . . .	TIO-3
terminators . . . . .	TIO-5
text file . . . . .	TIO-1, TIO-7
Text_Io package . . . . .	TIO-165
throw away, <i>see</i> Delete	
type	
file	
Io.File_Type type . . . . .	TIO-38
Text_Io.File_Type type . . . . .	TIO-179
type error, <i>see</i> Output_Type_Error	
Type_Set subtype	
Io.Type_Set . . . . .	TIO-102
Io.Enumeration_Io generic package . . . . .	TIO-109
Type_Set type	
Text_Io.Type_Set . . . . .	TIO-215
Text_Io.Enumeration_Io generic package . . . . .	TIO-217

U

Unbounded constant	
Io.Unbounded . . . . .	TIO-8, TIO-103
Text_Io.Unbounded . . . . .	TIO-165, TIO-216

Unsupported_Error	
Io_Exceptions.Use_Error exception . . . . .	TIO-163
Upper_Case constant	
Io.Upper_Case . . . . .	TIO-104
Use_Error exception	
Io package . . . . .	TIO-2, TIO-3, TIO-5
Append procedure . . . . .	TIO-13
Create procedure . . . . .	TIO-21
Delete procedure . . . . .	TIO-25
Open procedure . . . . .	TIO-64
Reset procedure . . . . .	TIO-80
Set_Error procedure . . . . .	TIO-88
Set_Input procedure . . . . .	TIO-90
Set_Line_Length procedure . . . . .	TIO-93
Set_Output procedure . . . . .	TIO-95
Set_Page_Length procedure . . . . .	TIO-96
Io_Exceptions.Use_Error . . . . .	TIO-163
Text_Io package	
Create procedure . . . . .	TIO-170
Delete procedure . . . . .	TIO-173
Open procedure . . . . .	TIO-194
Reset procedure . . . . .	TIO-202
Set_Line_Length procedure . . . . .	TIO-208
Set_Page_Length procedure . . . . .	TIO-210

V

value error, *see* Input\_Value\_Error, Output\_Value\_Error

version error, *see* Nonexistent\_Version\_Error

W

width	
default	
Io.Enumeration_Io.Default_Width constant . . . . .	TIO-111
Io.Integer_Io.Default_Width constant . . . . .	TIO-145
Text_Io.Enumeration_Io.Default_Width constant . . . . .	TIO-219
Text_Io.Integer_Io.Default_Width constant . . . . .	TIO-253
Wildcard_Iterator generic procedure	
Io.Wildcard_Iterator . . . . .	TIO-105
Wildcard_Iterator procedure	
Io.Wildcard_Iterator . . . . .	TIO-8, TIO-108
Note_Error generic formal procedure . . . . .	TIO-106
Process generic formal procedure . . . . .	TIO-107
wildcards . . . . .	TIO-4

window . . . . .	TIO-2
Editor window	
Io.Current_Error function . . . . .	TIO-22
Io.Current_Input function . . . . .	TIO-23
Io.Current_Output function . . . . .	TIO-24
Io.Standard_Error function . . . . .	TIO-99
Io.Standard_Input function . . . . .	TIO-100
Io.Standard_Output function . . . . .	TIO-101
Text_Io.Current_Input function . . . . .	TIO-171
Text_Io.Current_Output function . . . . .	TIO-172
Text_Io.Standard_Input function . . . . .	TIO-213
Text_Io.Standard_Output function . . . . .	TIO-214
write	
Boolean value to Message window	
Io.Echo procedure . . . . .	TIO-31
character to Message window	
Io.Echo procedure . . . . .	TIO-26
floating-point value to Message window	
Io.Echo procedure . . . . .	TIO-29
integer value to Message window	
Io.Echo procedure . . . . .	TIO-28
open to	
Out_File constant . . . . .	TIO-65
string to Message window	
Io.Echo procedure . . . . .	TIO-27
string to Message window/advance line	
Io.Echo_Line procedure . . . . .	TIO-32
write, <i>see also</i> Put, Put_Line	
write-only access	
Io.File_Mode subtype . . . . .	TIO-37
Text_Io.File_Mode type . . . . .	TIO-178
Write_To_Read_Only_Page_Error	
Io_Exceptions.Device_Error exception . . . . .	TIO-157



# RATIONAL

## READER'S COMMENTS

**Note:** This form is for documentation comments only. You can also submit problem reports and comments electronically by using the SIMS problem-reporting system. If you use SIMS to submit documentation comments, please indicate the manual name, book name, and page number.

Did you find this book understandable, usable, and well organized? Please comment and list any suggestions for improvement.

---

---

---

---

---

If you found errors in this book, please specify the error and the page number. If you prefer, attach a photocopy with the error marked.

---

---

---

---

Indicate any additions or changes you would like to see in the index.

---

---

---

---

How much experience have you had with the Rational Environment?

6 months or less \_\_\_\_\_ 1 year \_\_\_\_\_ 3 years or more \_\_\_\_\_

How much experience have you had with the Ada programming language?

6 months or less \_\_\_\_\_ 1 year \_\_\_\_\_ 3 years or more \_\_\_\_\_

Name (optional) \_\_\_\_\_ Date \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ ZIP Code \_\_\_\_\_

**Please return this form to:**

**Publications Department  
Rational  
1501 Salado Drive  
Mountain View, CA 94043**

