# Rational Environment Reference Manual

Reference Summary (RS)

RATIONAL

**Copyright © 1985, 1986, 1987, 1990, 1993 by Rational**

Product Number: 4000-00486

Rev. 2.0, September 1985
Rev. 3.0, November 1985
Rev. 4.0, July 1986
Rev. 5.0, July 1987
Rev. 6.0, September 1990
Rev. 7.0, March 1993

This document is subject to change without notice.

Note the Reader's Comments forms at the end of this book, which request the user's evaluation to assist Rational in preparing future documentation.

AIX and RISC System/6000 are trademarks of International Business Machines Corporation.

DECstation, DECwindows, ULTRIX, VAXstation, VMS, and VT100 are trademarks of Digital Equipment Corporation.

Motorola is a registered trademark of Motorola, Inc.

PostScript is a registered trademark of Adobe Systems Incorporated.

Rational and R1000 are registered trademarks and Compilation Integrator and Rational Environment are trademarks of Rational.

Sun Workstation is a registered trademark and OpenWindows, SunOS, and X11/NeWS are trademarks of Sun Microsystems, Inc.

X Window System is a trademark of MIT.

**Rational, 3320 Scott Boulevard, Santa Clara, California 95054-3197**

# Contents


## INTRODUCTION TO THE RATIONAL DOCUMENTATION SET RS-1

The Rational Documentation Set RS-1
System Manager's Guide RS-2
Rational Environment User's Guide RS-3
Rational Environment Basic Operations RS-3
Rational Environment Reference Manual RS-3
  Volume 1: Reference Summary RS-3
  Volume 2: Editing Images (EI) and Editing Specific Types (EST) RS-5
  Volume 3: Debugging (DEB) RS-6
  Volume 4: Session and Job Management (SJM) RS-6
  Volume 5: Library Management (LM) RS-6
  Volume 6: Text Input/Output (TIO) RS-6
  Volume 7: Data and Device Input/Output (DIO) RS-7
  Volume 8: String Tools (ST) RS-7
  Volume 9: Programming Tools (PT) RS-7
  Volume 10: System Management Utilities (SMU) RS-7
  Volume 11: Project Management (PM) RS-7
Reference Manual for the Ada Programming Language RS-8
Environment Reference Manual: Organization and Conventions RS-8
Organization of Books in the Reference Manual RS-8
Cross-Reference Conventions in the Reference Manual RS-10
Finding Information about the Environment RS-11
Using the Manuals RS-11
Viewing Specifications Online RS-12
Using Online Help RS-12
Feedback to Rational: Reader's Comments Form RS-12

## PARAMETER-VALUE CONVENTIONS RS-13

Organization of Topics RS-13
Referencing Environment Objects: Overview RS-14
Designation RS-14
Special Names RS-15
Special Names for Specific Objects RS-16
Special Names for Designated Objects RS-16
Special Names for Default Objects RS-17
Special Names vs. Parameter Placeholders RS-18
Pathnames RS-18
Fully Qualified Pathnames RS-19
Relative Pathnames RS-19
Pathnames for Ada Objects RS-20
Context RS-20
Current Context RS-20

Current Library RS-21
How the Environment Resolves Names RS-21
Library.Resolve Command RS-21
Examples RS-22
Context Characters RS-23
Context Character ! RS-24
Context Character Pair !! RS-24
Context Character Pair [ ] RS-24
Context Character $ RS-25
Context Character $$ RS-25
Context Character ^ RS-26
Context Characters $[*name*] RS-27
Context Characters $$[*name*] RS-27
Context Characters ^[*name*] RS-27
Searchlist Context Character } RS-27
Ada Context Character ` RS-28
When ` Follows a Library Name RS-28
When ` Follows an Ada-Unit Name RS-29
Wildcards for Matching One or More Pathnames RS-29
Naming Wildcard # RS-30
Naming Wildcard @ RS-30
Naming Wildcard ? RS-31
Naming Wildcard ?? RS-31
Substitution Characters for Specifying Destination Names RS-32
Testing Substitution Characters RS-32
Substitution Character @ RS-33
Substitution Character # RS-33
Special Characters for Specifying Sets of Objects RS-34
Set-Notation Characters [ ] RS-34
Indirect-File Prefix _ RS-35
Special Characters for Specifying Deleted Objects RS-37
Attributes RS-37
Attribute Syntax RS-38
Specifying Shared Properties RS-39
Specifying Disjoint Properties RS-39
Excluding Properties RS-39
Ada-Part Attributes: 'Spec and 'Body RS-40
Version Attribute 'V RS-40
Class Attribute 'C RS-41
Compilation-State Attribute 'S RS-44
Nickname Attribute 'N RS-45
Link Attribute 'L RS-45
Target-Key Attribute 'T RS-46
View Attributes 'Spec_View and 'View RS-47
Conditional Attribute 'If RS-48
Context Characters for Use in the Debugger RS-49
Unit-Name Prefix . RS-49
Stack-Frame Prefix _ RS-49
Task-Identifier Prefix % RS-50
Restricted Naming Expressions RS-51
Specifying Other Command Inputs: Overview RS-51
Ada Identifiers RS-51
Special Values RS-52
Pattern-Matching Characters RS-52
Options Parameter RS-53

General Option Specifications RS-53
Options with Boolean Values RS-54
Options with Literal Values RS-54
Multiple Options RS-55
Response Parameter RS-55
Specifying a Predefined Response Profile RS-56
  Basic Response Profiles RS-56
  Special-Purpose Response Profiles RS-58
Specifying Individual Response Characteristics RS-59
  Syntax for Combining Options and Special Values RS-60
  Setting Error Reaction RS-60
  Filtering Messages RS-61
  Changing the Format of Message Prefixes RS-62
  Adjusting Output Width RS-62
  Redirecting Command Output RS-62
  Specifying a Nondefault Activity RS-63
  Specifying Nondefault Remote-Information Files RS-63
Summary of Response-Parameter Options RS-63

**ENVIRONMENT SPECIFICATIONS** **RS-65**

Map of the Rational Environment Library System RS-65
!Commands RS-75
!IO RS-325
!LRM RS-385
!Tools RS-393
Abbreviations RS-679

**TERMINAL TYPES AND KEYMAPS** **RS-685**

Predefined Terminal Types RS-685
Standard Terminal Types RS-685
Rational Windows Interface Terminal Types RS-685
Rational X Interface Terminal Types RS-686
Objects That Define Terminal Types RS-686
Key Naming and Character Recognition RS-687
  The *Terminal_Type*_Keys File RS-687
  The *Terminal_Type*_Key_Names Package RS-688
Key Binding RS-689
  Procedure *Terminal_Type*_Commands RS-689
  The *Terminal_Type*_User_Commands File RS-691
Terminal-Type Definition and Recognition RS-691
  The Terminal_Types File RS-691
  The Terminal_Recognition File RS-692

**GLOSSARY: RATIONAL AND DOD-STD-2167/2167A TERMS** **RS-695**

Rational Terms RS-695
DOD-STD-2167/2167A Terms RS-710

**MASTER INDEX** **RS-713**

# Introduction to the Rational Documentation Set

This tabbed section is divided into three subsections:

- Overview of the manuals in the Rational® documentation set, with a brief description of each manual and suggestions regarding the appropriate audience, starting on page 1. Topics include:


System Manager's Guide . . . . . . . . . . . . . . . . . . . . . . . . RS-2
Rational Environment™ User's Guide . . . . . . . . . . . . . . . . . RS-3
Rational Environment Basic Operations . . . . . . . . . . . . . . . . RS-3
Rational Environment Reference Manual . . . . . . . . . . . . . . . RS-3
Volume 1: Reference Summary (RS) . . . . . . . . . . . . . . . . RS-3
Volume 2: Editing Images (EI) and Editing Specific Types (EST) . . RS-5
Volume 3: Debugging (DEB) . . . . . . . . . . . . . . . . . . . . RS-6
Volume 4: Session and Job Management (SJM) . . . . . . . . . . . RS-6
Volume 5: Library Management (LM) . . . . . . . . . . . . . . . . RS-6
Volume 6: Text Input/Output (TIO) . . . . . . . . . . . . . . . . RS-6
Volume 7: Data and Device Input/Output (DIO) . . . . . . . . . . RS-7
Volume 8: String Tools (ST) . . . . . . . . . . . . . . . . . . . . RS-7
Volume 9: Programming Tools (PT) . . . . . . . . . . . . . . . . RS-7
Volume 10: System Management Utilities (SMU) . . . . . . . . . . RS-7
Volume 11: Project Management (PM) . . . . . . . . . . . . . . . RS-7
Reference Manual for the Ada Programming Language . . . . . . . . RS-8


- Description of how the *Rational Environment Reference Manual* is organized and some of the conventions used, starting on page 8. Topics include:


Organization of Books in the Reference Manual . . . . . . . . . . . RS-8
Cross-Reference Conventions in the Reference Manual . . . . . . . . RS-10


- Explanation of how to navigate through the manuals and how to use the Environment's online help facilities, starting on page 11. Topics include:


Using the Manuals . . . . . . . . . . . . . . . . . . . . . . . . . RS-11
Viewing Specifications Online . . . . . . . . . . . . . . . . . . . RS-12
Using Online Help . . . . . . . . . . . . . . . . . . . . . . . . . RS-12


## THE RATIONAL DOCUMENTATION SET

The Rational documentation set includes the following manuals (see Figure 1). All but the *System Manager's Guide* constitute the core Environment documentation set; the *System Manager's Guide* is delivered with each system.

*System Manager's Guide*
*Rational Environment User's Guide*
*Rational Environment Basic Operations*
*Rational Environment Reference Manual*
*Reference Manual for the Ada Programming Language*

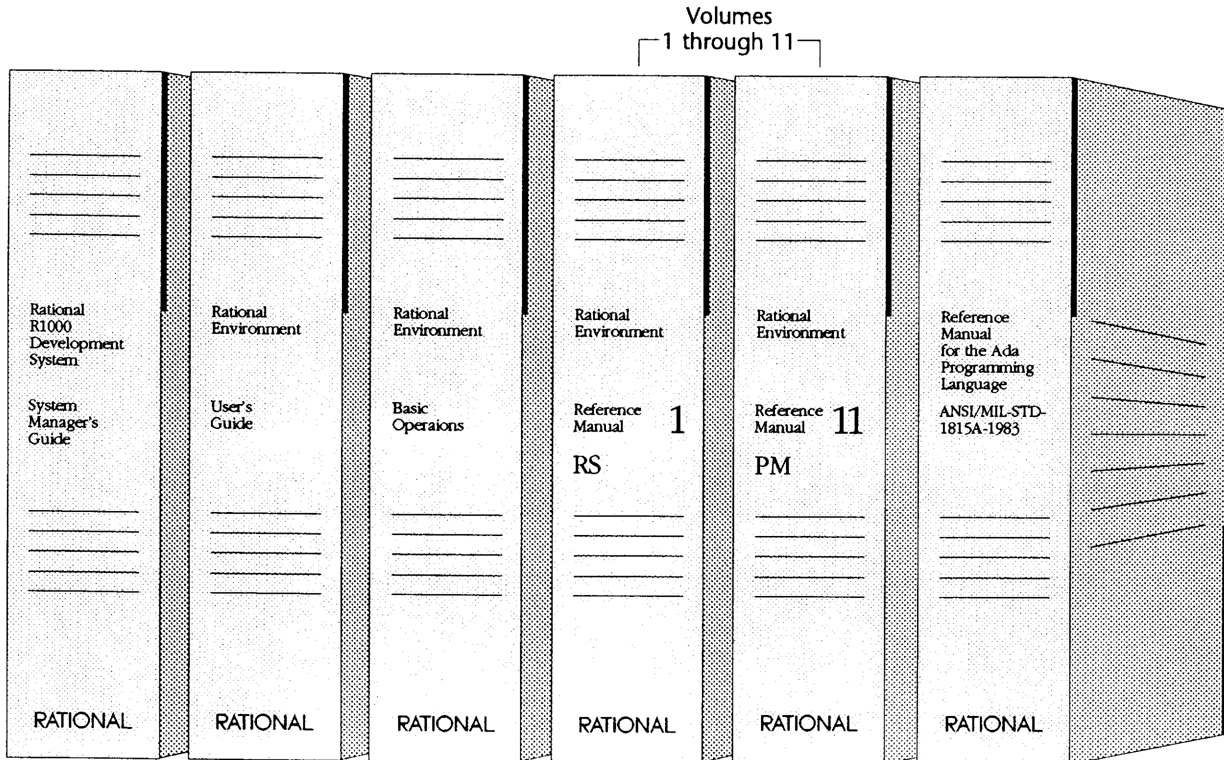



***Figure 1 Rational Documentation Set***

Each of these manuals consists of one volume, with the exception of the *Rational Environment Reference Manual*, which contains the following eleven volumes:

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

The following subsections describe each manual.

## System Manager's Guide

The *System Manager's Guide* describes the Rational R1000® Development System and how to use it. It contains chapters on starting and stopping the system, emergency procedures, user accounts, access control, and disk- and tape-drive operations. The guide also gives valuable information on how to maintain system efficiency.

*Audience:* Refer to this guide if you are a system manager or are required to maintain R1000 systems or user accounts.

## Rational Environment User's Guide

The *Rational Environment User's Guide* describes the basic concepts and operations of the Environment, illustrated with sample scenarios. It is divided into three parts:

- Part I, "Getting Started," explains how to log in, use windows, execute commands, and get help.
- Part II, "Editing Text," describes how to create, edit, and save text files.
- Part III, "Developing Simple Ada Programs," explains how to create, edit, execute, debug, browse, and modify Ada programs.

*Audience:* If you are a new Environment user, you can use this guide to teach yourself basic Environment concepts and operations. If you have more experience with the Environment, you can consult this guide to resolve questions and to supplement existing knowledge.

## Rational Environment Basic Operations

The *Rational Environment Basic Operations* manual describes the steps required to perform common operations in the Environment, but it does not present context, as does the *User's Guide*, described above. *Basic Operations* focuses on fundamental areas of the Environment needed to begin work on small Ada programs in single libraries. This manual contains keymaps for some terminals, which indicate what keys have been bound to Environment commands, and explains how to modify these key bindings for other uses.

*Audience:* If you already understand Environment concepts, you can use *Basic Operations* as a reference for performing operations. It is also used for reference in training classes.

## Rational Environment Reference Manual

Each of the eleven volumes of the *Reference Manual* (see Figure 2) consists of one or more *books* that contain information on a particular feature or area of application in the Environment. The contents of a given book are arranged by Ada units (usually packages), which contain the declarations for sets of related Environment commands. 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 cross-references and in page numbers in the Master Index (described with volume 1, below).

For information on how to navigate in the *Reference Manual*, see "Environment Reference Manual: Organization and Conventions," on page 8, following this description of the individual books.

### Volume 1: Reference Summary

Volume 1 is divided into five major tabbed sections:

- **Introduction to the Rational Documentation Set:** This section describes the manuals and conventions in the documentation set and explains how to navigate through them and the online help.

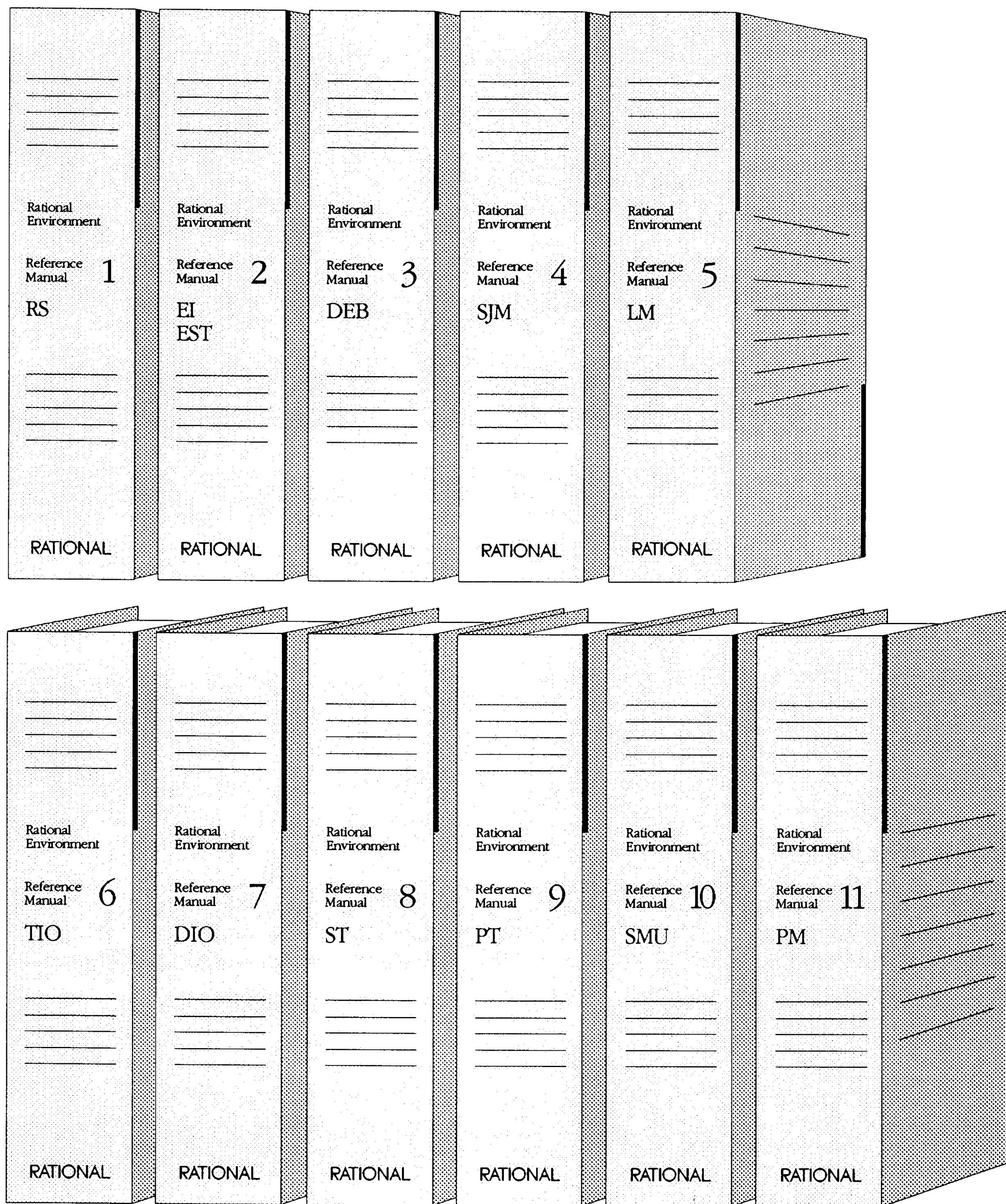



***Figure 2 Rational Environment Reference Manual***

- **Parameter-Value Conventions:** This section explains the kinds of values that most Environment commands accept. It is divided into two major parts: "Referencing Environment Objects," which explains naming and wildcards, and "Specifying Other Command Inputs," of which typical examples are the Options

and Response parameters. A brief summary of these conventions can be found in other volumes of the *Reference Manual.*

*Audience:* All Environment users need to be familiar with this material, since it applies to all Environment commands, regardless of the book in which they are documented. This section also contains useful information if you are writing tools that reference Environment objects.

- **Environment Specifications:** This section provides a quick reference to the resources provided by the Environment. It contains the full Ada specification for each unit in the standard Environment, organized by pathname. The section begins with a list of the units in the library system of the Environment, cross-referenced to the manual or book in which they are documented. This section also contains a list of the standard abbreviations for Environment commands and an overview of Environment keymaps.

  *Audience:* This section is designed for users who are familiar with scanning Ada specifications.

- **Glossary:** The Glossary defines terms specific to the Rational Environment as well as the Rational Design Facility: DOD-STD-2167/2167A.

  *Audience:* This section is designed to provide users of the Environment and the Design Facility a quick reference. New users are encouraged to look through the Glossary to familiarize themselves with the contents.

- **Master Index:** The Master Index combines the indexes for all the books in the *Rational Environment Reference Manual.* It contains entries on key concepts as well as entries for each unit and its declarations, exceptions, errors, enumerations, pragmas, switches, and so on.

  *Audience:* If you are using the Environment to develop programs, especially if you are new to the Environment, you will find this index useful. As you become more familiar with the contents of the *Reference Manual,* you will often go directly to the book containing the information you want. However, this index is likely to remain a valuable resource.

## Volume 2: Editing Images (EI) and Editing Specific Types (EST)

Volume 2 is divided into two books, Editing Images and Editing Specific Types.

- **Editing Images:** When working with the Environment, you use certain editing operations that do not depend on the type of image being edited. These operations include cursor movement, search-and-replace operations, editing, window management, screen management, and macros. These operations constitute the means of interacting with the Environment at the simplest level. The EI book presents introductory material on the logic and structure of the basic editing commands, followed by reference entries for all subpackages in package Editor.

  *Audience:* This book is designed for new Environment users who need to learn the basics of editing images.

- **Editing Specific Types:** The Environment uses different types of images, each of which possesses its own underlying form and structure. Each type of image, such as Ada units, command windows, menus, and text images, can require commands that differ from those defined in the EI book. The EST book contains reference information describing commands for editing these specific types of images, including commands in package Common.

  *Audience:* This book is intended for users who are familiar with the Environment and with Ada programming.

## Volume 3: Debugging (DEB)

The debugger provides facilities for analyzing the behavior of Ada programs running in the Environment. This book gives a detailed description of the debugger and its operation. The book provides an in-depth analysis, with examples, of debugging Environment programs, debugger interactions with the editor, debugger facilities, miscellaneous facilities, and debugger naming.

*Audience:* This information is intended for users who are familiar with the Environment and Ada programming. New Environment users can benefit by scanning this information as they begin to develop programs; additional information about debugging can be found in the *User's Guide.*

## Volume 4: Session and Job Management (SJM)

Two of the important concepts used by the Environment are *sessions* and *jobs.* Sessions (which are analogous to accounts on other systems) define your work environment; since you can have multiple sessions, you can create different ones for different projects. Jobs are what the system runs when you initiate an operation. The Session and Job Management book contains reference information describing commands and tools for managing sessions and jobs, including tailoring individual sessions for different kinds of work, starting jobs, stopping them, tailoring their behavior, and using profiles.

*Audience:* Although this book is valuable to almost all Environment users, new users can start with package Job for killing or interrupting jobs, package What for help information, and package Queue for print queues. (Packages Queue and Operator are also documented in volume 10, System Management Utilities.)

## Volume 5: Library Management (LM)

Libraries are the building blocks of the hierarchical Environment structure and can contain objects such as Ada units, files, and other libraries (which are synonymous with directories on other systems). Libraries also define the context for the compilation of Ada programs and manage visibility between Ada units. The Library Management book presents information on how libraries are organized, how to compile large systems, and how to control access to libraries and objects in them. LM also describes operations for copying, saving, and restoring objects; link operations; switch operations for setting library characteristics; operations for comparing, merging, and searching Ada units and files; and cross-reference lists.

*Audience:* If you are a new Environment user, see packages Archive and Library for how to manipulate objects in libraries. If you are a system manager, see also packages Access_List and Access_List_Tools for information on system access. If you are creating programs, see also package Compilation for information on compiling.

## Volume 6: Text Input/Output (TIO)

The Environment provides I/O packages for manipulating text files, which are files that contain ASCII characters intended for viewing, editing, and so on. The Text Input/Output book contains reference information documenting the Ada-defined packages Text_Io and Io_Exceptions, as well as Rational-developed I/O packages. Topics covered in this book include files, devices, windows, file handles, filenames, access control, concurrency, representations of terminators, exceptions, and error reactions. Other packages available in the Environment for performing I/O are documented in volume 7.

*Audience:* If you are a developer who is familiar with the Environment and with Ada programming, you can use this information when building programs.

## Volume 7: Data and Device Input/Output (DIO)

The Environment provides I/O packages for manipulating binary files, devices, and editor windows. The Data and Device Input/Output book contains reference information similar to that found in TIO. The explanations include information on the Ada-defined packages Direct_Io, Sequential_Io, and Io_Exceptions, as well as information on the Rational-developed I/O packages Polymorphic_Sequential_Io and Window_Io.

*Audience:* If you are a developer who is familiar with the Environment and with Ada programming, you can use this information when building programs.

## Volume 8: String Tools (ST)

The Environment provides tools for operations such as manipulating different types of strings in programs, mapping string values to values of other types, operating on tables of unique strings, and operating on the Ada-defined String type. The String Tools book contains reference information describing some of these tools. Reference information describing some related programming tools is in volume 9.

*Audience:* If you are a developer who is familiar with the Environment and with Ada programming, you can use this information when building programs.

## Volume 9: Programming Tools (PT)

Another set of tools offered by the Environment are the programming tools. The Programming Tools book contains reference information describing some of these programming tools, including reusable software components and various other useful programming tools, along with reference information on the Ada-defined packages Calendar, Standard, and System.

*Audience:* If you are a developer who is familiar with the Environment and with Ada programming, you can use this information when building programs.

## Volume 10: System Management Utilities (SMU)

To maintain an R1000, the system manager must perform operations such as creating and deleting user accounts, setting up print queues, performing backups, and configuring terminal ports. The System Management Utilities book contains reference information describing commands and tools for manipulating an R1000. Note that SMU differs from the *System Manager's Guide* in that it is organized by package rather than by system-management tasks and related topics.

*Audience:* This reference material is intended for system managers who are familiar with the Environment and with Ada programming. (Packages Operator and Queue are also documented in volume 4, Session and Job Management.)

## Volume 11: Project Management (PM)

To facilitate developing projects in Ada, the Environment provides commands and tools for partitioning a project into components, testing and releasing implemented components, tracking the history of Ada-unit versions and configurations, and coordinating multiple developers and development efforts. The Project Management book documents these operations, including package Cmvc (configuration management and version control).

*Audience:* If you are familiar with the Environment and with Ada programming, use this book to assist you in managing a project. If you are new to the Environment, look through the introduction to package Cmvc to familiarize yourself with its contents.

### Reference Manual for the Ada Programming Language

The *Reference Manual for the Ada Programming Language* (LRM) contains the Ada standard, as published by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO), against which compilers are verified. In Appendix F is the information required by ANSI and the ISO on the implementation-dependent features of the R1000 native compiler. An Appendix F for each of Rational's Cross-Development Facilities accompanies the manuals for those products.

*Audience:* If you are creating Ada programs, use this manual to ensure that the Ada grammar and syntax conform to the standard and to verify compiler operation.

## ENVIRONMENT REFERENCE MANUAL: ORGANIZATION AND CONVENTIONS

The following subsections describe the organization of the books in the *Rational Environment Reference Manual* and the cross-reference conventions used.

### Organization of Books in the Reference Manual

Each book in the *Rational Environment Reference Manual* begins with a large pink tab on which the name of the book appears. This is followed by a series of smaller pink and white tabs, respectively, indicating major and minor sections in the book. The organization of a sample book is shown in Figure 3 and described below.

- **Book:** As mentioned above, each book is preceded by a large pink tab, behind which are the title page, table of contents, and preface, if one exists.
- **Key concepts:** Most 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. If the Key Concepts section is long, it may be divided by additional tabs.
- **Reference sections:** The largest part of each book consists of tabbed sections that contain information about specialized topics or groups of commands. There are two types of reference sections:
  - **Unit sections:** Each command, tool, 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 the reference entries for the declarations (for example, procedures, functions, and types) within that unit.

    Each unit section is preceded by a white tab. The sections for units are alphabetized by the simple names of the units. For example, the section for package !Commands.Job is alphabetized under Job.

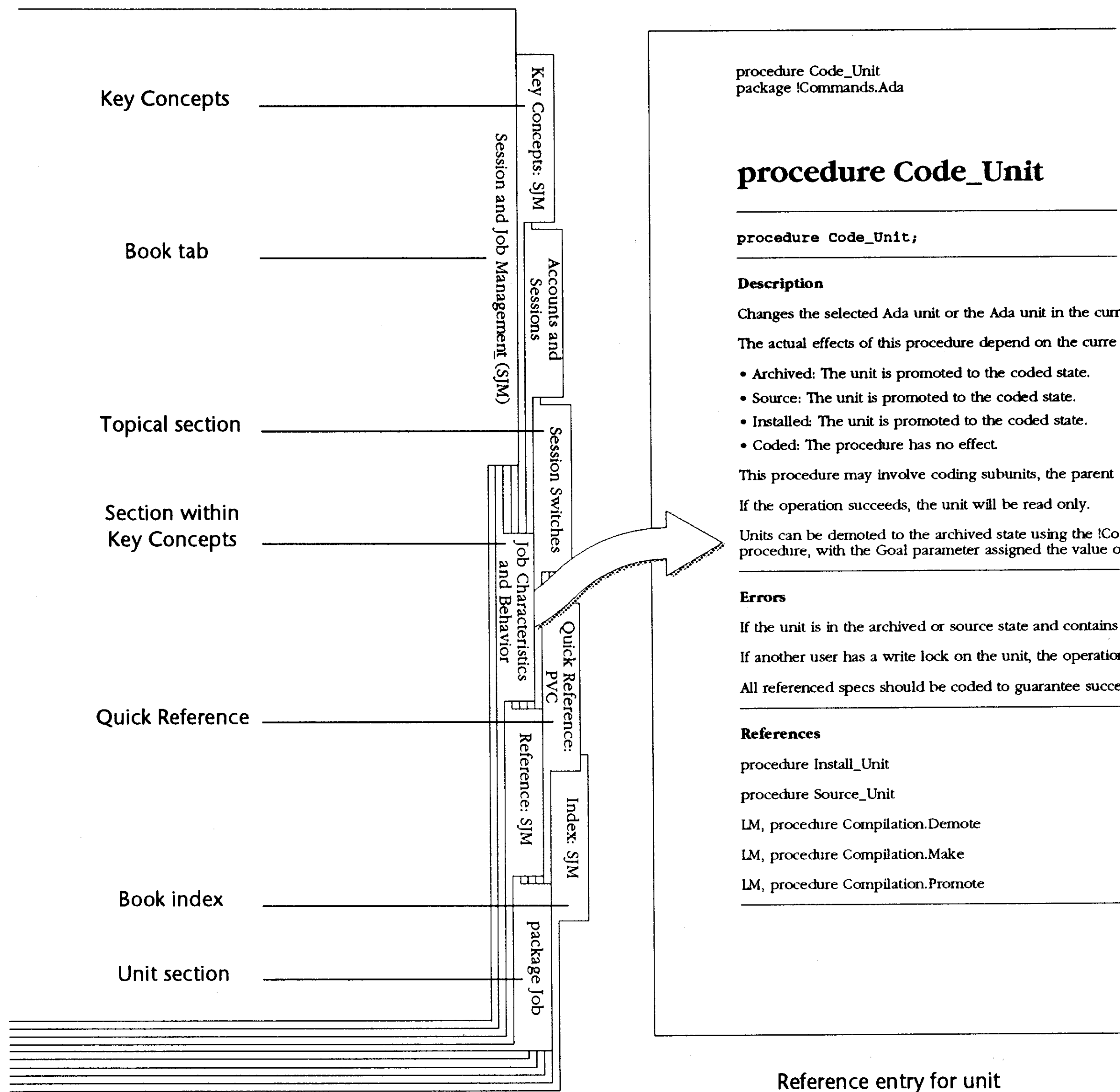



***Figure 3 A Sample Book***

For many units, introductory material and/or examples specific to the unit appear immediately after the section tabs. Introductory material is followed by *reference entries* describing the unit's declarations. The reference entries are organized alphabetically by the simple names of the declarations. The simple name of the given declaration and the fully qualified pathname of the enclosing unit appear at the top of each page in a reference entry.

- **Topical sections:** Like the unit sections, explanatory/topical sections are preceded by white tabs, and these sections are alphabetized with the unit sections. The topical sections, such as the Session Switches section in Session and Job Management (SJM), discuss Environment facilities that are not associated with specific packages.

- **Quick Reference for Parameter-Value Conventions:** Many books contain a brief summary of the Environment-defined conventions for parameter values. This section summarizes conventions for referencing Environment objects, as well as specifying other command inputs. Complete explanations and examples of all Environment-defined parameter-value conventions can be found in the Parameter-Value Conventions tabbed section of the Reference Summary.
- **Index:** The index is 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.*

  The index lists page numbers using the book abbreviation followed by the page number (for example, LM-322). Italic page numbers indicate primary reference entries for declarations; nonitalic page numbers indicate key concepts, defined terms, cross-references, raised exceptions, and so on.

## Cross-Reference Conventions in the Reference Manual

The following conventions are used in cross-references to information. Many references appear at the end of individual reference entries and direct you to other reference entries. Other references appear in text. The *Reference Manual* uses the following kinds of references:

- Simple name—for example:

  procedure Copy

  A reference such as this indicates that the declaration is in the same unit. If the unit contains nested packages, references to nested declarations use qualified pathnames.
- Qualified pathname—for example:

  procedure Library.Copy

  This kind of reference indicates that the declaration is in another unit in the same book. Behind that unit's tab, the declaration appears in alphabetical order.
- Declarations in other books—for example:

  the Library Management (LM) book, procedure Library.Copy
  Library Management (LM), procedure Library.Copy
  LM, procedure Library.Copy

  This kind of reference can be indicated by the addition of the complete book name or abbreviation for that book. The tabs for the units are in alphabetical order, as are the declarations within the units.

References to specific declarations in the Environment library system (rather than the documentation for them) typically are indicated by their fully qualified pathnames, such as "procedure !Commands.Library.Copy." When the context is clear, however, a shorter name may 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, you can either look at the unit's specification in the Reference Summary or view it online.(See "Finding Information about the Environment," below, for details about viewing specifications online.)

# FINDING INFORMATION ABOUT THE ENVIRONMENT

Rational provides a variety of resources for finding information about the Environment. The following subsections explain how to look for information using the manuals, online specifications, and online help.

## Using the Manuals

The *Rational Environment Reference Manual* is organized by Ada units, most of which are packages but some of which are procedures and functions. Related units are grouped by book. This organization follows the Ada practice of using packages to group related material together.

Each Ada unit contains declarations, most of which are procedures (commands) but which also include functions, types, constants, and, in some cases, smaller packages. Within the packages that it documents, the *Reference Manual* provides reference entries for each of these declarations. This information is explained more fully under "Organization of Books in the Reference Manual," on page 8.

When looking for information in the *Reference Manual*, you can take one of the following routes, depending on what you know and what you want to find:

- You know the simple name of a declaration:

  Look in the Master Index in volume 1 to find the book abbreviation and page number for the declaration.

- You know the Ada unit and declaration names:

  Find the book containing the unit by looking in either the Environment Specifications section or the Master Index in volume 1. You can also scan the titles on the binders to find a category related to the unit and then look at the unit names on the tabs. Within the unit, declarations are listed in alphabetical order.

- You want to know what kinds of operations an Ada unit contains:

  Start by looking at the introduction to the tabbed section that describes that unit, if such a section exists. You can also scan the Environment specification for the unit in the Reference Summary (volume 1).

- You know the kind of operation you want to perform:

  Look in the Master Index for a topical entry that corresponds to the operation (delete a file, for example). You can also look at the titles on the binders or at the specifications in volume 1 to find a book containing Ada units likely to be related to your operation. Once you have found a book of interest, look through the preface, table of contents, and index or read the Key Concepts section to see if the book documents the operation you want to perform.

- You want to know about available operations:

  Look through the Environment specifications and the Master Index in volume 1. You can also find valuable information in the Key Concepts section of each book.

If you cannot find an item in the Master Index, it may be documented in the manuals for a product layered on top of the Environment (for example, Rational Networking—TCP/IP or Rational's family of Compilation Integrator™ products).

## Viewing Specifications Online

In addition to the standard set of Environment specifications in volume 1, the complete set is present online in the !Commands and !Tools libraries. If you know the pathname of a declaration, you can view its specification in an Environment window by providing its pathname to the Common.Definition procedure:

```
Definition ("!Commands.Debug");
```

If you know the simple name of the unit in which the declaration appears, in most cases you can use searchlist naming, as follows, as a quick way to view the unit:

```
Definition ("\Debug");
```

From the Environment specification, you can obtain additional help on a unit by placing the cursor on the unit's declaration and pressing [Help].

## Using Online Help

Much of the information contained in the reference entries for each unit is available through the online help facilities in the Environment.

To obtain online help:

- From an Environment specification, place the cursor on the unit in question and press [Help].
- From any command window, execute the What.Does command, specifying the name of the command for which you want information. You can specify fully qualified pathnames, simple names, or fragments of names.

  If you specify a simple name or fragment that is ambiguous, the Environment displays a menu of help topics from which you can choose. From this menu, place the cursor on the name of the unit for which you want information, and:

  - Press [Explain] or [Help] to obtain information on that unit.
  - Press [Definition] to traverse to the specification for that unit.

You can also traverse directly from the help window to the specification for the unit being described. To do so, place the cursor on the unit's specification at the top of the window and press [Definition].

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 online help facility.

# 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 pages of each book contain reader's comments forms that you can use to send us comments or to report errors. You can also submit problem reports and make suggestions electronically with the Rational problem-reporting system. When you submit reports and suggestions, please indicate the manual name, book name, and page number.

# Parameter-Value Conventions

Parameters of Rational Environment™ commands accept values that conform both to Ada and to Environment conventions. This tabbed section explains these conventions with examples. Introductory material describing the syntax of Ada parameters is located in the *Rational Environment User's Guide*.

A brief summary of these conventions can be found in the Quick Reference for Parameter-Value Conventions tabbed section of other volumes in the *Rational Environment Reference Manual*.

## ORGANIZATION OF TOPICS

This tabbed section divides the various parameter-value conventions into two groups:

- Conventions for referencing Environment objects, starting on page 14. Topics include:


Attributes . . . . . . . . . . . . . . . . . . . . . RS-37
Context and name resolution . . . . . . . . RS-20
Context characters . . . . . . . . . . . . . . . RS-23
Debugger context characters . . . . . . . . . RS-49
Deleted objects . . . . . . . . . . . . . . . . . RS-37
Designation . . . . . . . . . . . . . . . . . . . RS-14
Indirect files . . . . . . . . . . . . . . . . . . RS-35
Parameter placeholders . . . . . . . . . . . RS-18
Pathnames . . . . . . . . . . . . . . . . . . . . RS-18
Restricted naming expressions . . . . . . . . RS-51
Set notation . . . . . . . . . . . . . . . . . . . RS-34
Special names . . . . . . . . . . . . . . . . . . RS-15
Substitution characters . . . . . . . . . . . . RS-32
Wildcard characters for pathnames . . . . . . RS-29


- Conventions for specifying other command inputs, starting on page 51. Topics include:


Ada identifiers . . . . . . . . . . . . . . . . . RS-51
Options-parameter syntax . . . . . . . . . . RS-53
Response-parameter values . . . . . . . . . RS-55
Special values . . . . . . . . . . . . . . . . . RS-52
Wildcard characters for searches . . . . . . . RS-52

## REFERENCING ENVIRONMENT OBJECTS: OVERVIEW

Many Environment commands operate on one or more Environment objects. Depending on the command, objects can be specified using one or both of the following strategies:

- You can *designate* ("point to") the objects or their representations on the screen.
- You can enter *pathnames* as values to the command's parameters.

*Designation* refers to using cursor position and/or selection (highlighting) to specify objects for command operation. Some commands, such as Common.Promote, always accept designation as a means of specifying objects; other commands, such as Common.Definition, can be made to accept a particular form of designation through the use of *special names* as parameter values.

*Pathnames* are strings that explicitly reference Environment objects:

- *Fully qualified pathnames* contain all the information necessary to specify objects uniquely.
- *Relative pathnames* derive some naming information from the *context* in which they are used—that is, the Environment maps them onto unique objects by *resolving* them relative to the nearest enclosing library. Relative pathnames are typically shorter and more convenient to use than fully qualified pathnames.

You can abbreviate any pathname by using *context characters* and *wildcard characters* to stand for particular portions of the name.

You can specify groups of objects in several ways:

- Wildcard characters allow you to construct a single general name that references multiple objects by *matching* multiple pathnames.
- *Set notation* allows you to enter a list of pathnames as a single string.
- *Indirect files* allow you to save sets of pathnames for repeated use. Specifying an indirect file is equivalent to specifying its contents in set notation.

When moving, copying, or renaming multiple objects, you can use wildcards to specify a set of *source* (or input) names and *substitution characters* to specify a set of *destination* (or output) names.

You can specify objects by their properties using *attributes*. Whereas pathnames identify objects by their location in the Environment library hierarchy, attributes match object properties such as class, version, Ada part, nickname, and compilation state.

Finally, you can use special characters to reference objects that have been deleted but not expunged or to reference objects in the context of the Rational debugger.

## DESIGNATION

*Designation* means referencing an object or structure by "pointing to" it on the screen in one of the following ways:

- Positioning the cursor on its name or in its image
- Selecting its name or image so that it appears in a highlighted font on the screen (only one selection can exist on the screen at any given time)

The following packages contain selection commands (most of these commands are bound to keys):

- Package !Commands.Common.Object (Object.Parent, Object.Child, and so on)
- Package !Commands.Editor.Region (Region.Start and Region.Finish)

■ Selecting its name or image *and* positioning the cursor in the highlighted area

The type of designation you can use with a given command depends primarily on the command itself:

- Some commands are "hard-wired" to accept one of the three types of designation listed above. For example, !Commands.Common.Object.Delete requires a selection that contains the cursor to specify what to delete.
- Other commands have explicit parameters that can be made sensitive to designation through the use of *special names*. For example, !Commands.Common-.Definition displays the object under the cursor when its Name parameter has the value "<CURSOR>".

  The choice of special name determines which type of designation is required. (See "Special Names," below.)

The three types of designation are of different strengths, corresponding to the amount of effort required. Cursor positioning (the weakest) requires the least effort; selection containing the cursor (the strongest) requires the most effort.

In general, Environment commands are designed so that the accepted strength of designation corresponds to how benign or destructive the command is. Commands that display objects (such as !Commands.Common.Definition) by default accept the weakest type of designation. Commands that remove objects from the Environment (such as !Commands.Common.Object.Delete) generally accept only the strongest type of designation.

## SPECIAL NAMES

*Special names* are Environment-defined strings that provide a shorthand way to reference various kinds of objects. Some special names resolve to specific objects, such as your home world; others resolve to objects that have been designated in various ways (see also "Designation," above); yet others resolve to certain default objects associated with the current image.

Special names:

- Have the form "<*special name*>". Values for *special name* are listed in Tables 1, 2, and 3 in the following subsections.
- Must be enclosed in quotation marks because they are strings.
- Can contain upper- and lowercase letters, although they are conventionally written in uppercase letters (for example, "<CURSOR>").
- Can be abbreviated, down to the shortest unambiguous string within the angle brackets (for example, "<c>", "<CUR>", and "<Curs>").

Special names are used as default parameter values in many Environment commands. You can replace a default special name with a different special name or with a pathname, as accepted by the command. Special names can be used alone or as components of larger pathnames (see Examples 2, 4, and 7 in the following subsections; see also "Pathnames," page 18).

Note that the Environment also recognizes *special values* that are enclosed in angle brackets—for example, "<DEFAULT>". Instead of referring to Environment objects, these values affect command behavior; see "Special Values," page 52.

## Special Names for Specific Objects

Three special names exist for specifying objects that users frequently need to reference, especially when working in subsystems. These special names are summarized in Table 1.

***Table 1 Special Names for Specific Objects***

| Shortest Form | Full Form | Description |
|---|---|---|
| `"<h>"` | `"<HOME>"` | Resolves to the user's home world, if any; otherwise, fails with an error message. |
| `"<su>"` | `"<SUBSYSTEM>"` | Resolves to the enclosing subsystem, if any; otherwise, fails with an error message. |
| `"<vi>"` | `"<VIEW>"` | Resolves to the enclosing view, if any; otherwise, fails with an error message. |

***Example 1.*** Assume that user Anderson wants to traverse from an arbitrary location in the Environment to his home world. The following command accomplishes this:

```
Definition ("<h>");
```

***Example 2.*** Assume that user Miyata wants to traverse from an arbitrary Environment location to a directory called Memos in his home world. The following command accomplishes this:

```
Definition ("<h>.memos");
```

***Example 3.*** Assume that you are working in an object in the Units directory of a view and you want to traverse to the view itself. To do this, you can enter the following command:

```
Definition ("<view>");
```

***Example 4.*** Assume again that you are working in an object in a view's Units directory, but now you want to traverse to the Configurations directory in the enclosing subsystem. The following command accomplishes this:

```
Definition ("<sub>.configurations");
```

Note that in Examples 2 and 4, special names are used as components of larger pathnames. As described in "Pathnames," below, name components are generally separated by periods; however, the period can be omitted between a special name and an adjacent name component—for example, `"<sub>configurations"`.

## Special Names for Designated Objects

Several special names make commands sensitive to designation. Each such special name resolves to an object that is designated in a particular way, enabling you to reference objects by "pointing to" them on the screen (see "Designation," above). These special names are listed in Table 2.

***Table 2 Special Names for Designated Objects***

| Shortest Form | Full Form | Description |
|---|---|---|
| "<c>" | "<CURSOR>" | Resolves to the object on which the cursor is located; any highlighted area is ignored. |
| "<r>" | "<REGION>" | Resolves to the highlighted object; cursor can be anywhere. Often specifies a highlighted source object in a copy command, where cursor specifies the destination. |
| "<s>" | "<SELECTION>" | Resolves to the highlighted object; the highlight must contain the cursor, or else an error results. |
| "<i>" | "<IMAGE>" | Resolves to the highlighted object containing the cursor (if any); otherwise, resolves to the object whose image contains the cursor. |
| "<t>" | "<TEXT>" | Resolves to the object whose name is a highlighted string containing the cursor. Equivalent to copying the highlighted string directly into the parameter prompt. |

***Example 5.*** Consider the default value ("<CURSOR>") for the Name parameter of the Common.Definition command. This special name allows you to traverse from a library to an object in it by putting the cursor anywhere on the object's library entry and entering the command as follows:

```
Definition (Name => "<CURSOR>");
```

***Example 6.*** Consider the default values for the From and To parameters of the Library.Copy command. These special names allow you to copy an object by highlighting its library entry, moving the cursor into the window containing the destination library, opening a command window there, and entering the command as follows:

```
Library.Copy (From => "<REGION>",
              To   => "<IMAGE>");
```

The use of "<REGION>" instead of "<SELECTION>" allows the cursor to be moved out of the highlighted area so that it can be used to specify the destination. In contrast, the use of "<SELECTION>" would require you to leave the cursor in the highlighted area.

***Example 7.*** Assume that user Miyata is in his home world and he wants to display a file called Restaurant_Review that exists in the Memos subdirectory. He can traverse directly to the file by highlighting the library entry for Memos, opening a command window, and entering the following command:

```
Definition (Name => "<sel>.restaurant_review");
```

Note that using the special name "<SELECTION>" means that the cursor must be in the highlighted area when the command window is created. This is true because a command executed in a command window uses the cursor's most recent position in the attached window.

## Special Names for Default Objects

Two special names resolve to a default object associated with the current session or image, unless a nondefault object has been highlighted. These special objects are listed in Table 3.

***Table 3 Special Names for Default Objects***

| Shortest Form | Full Form | Description |
|---|---|---|
| "<a>" | "<ACTIVITY>" | Resolves to the highlighted activity containing the cursor (if any); otherwise, resolves to the current activity (the activity for the currently executing job). When a job begins, the current activity is the same as the session default activity. |
| "<sw>" | "<SWITCH>" | Resolves to the highlighted library switch file; otherwise, resolves to the library switch file associated with the highlighted library; otherwise, resolves to the library switch file associated with the current image. The highlighted area (if any) must contain the cursor. |

## Special Names vs. Parameter Placeholders

Although similar in form, special names should not be confused with *parameter placeholders*, which appear as default parameter values for many commands. Whereas a special name actually resolves to an Environment object, a parameter placeholder serves only to indicate the type of object name you should enter in its place.

Parameter placeholders take the form ">>*clue*<<", where *clue* is a short phrase describing the name to be specified. Unlike special names, parameter placeholders *must* be replaced with pathnames or other string values. Executing a command without replacing a parameter placeholder results in an error. For example, the following command will execute successfully only when the parameter placeholder has been replaced with a valid name for a library:

```
Library.Create (Name => ">>LIBRARY NAME<<");
```

# PATHNAMES

Environment objects can be referenced explicitly using strings called *pathnames*. Pathnames:

- Consist of one or more *name components* separated by periods—for example, "!Users.Anderson.Calculation"
- Must be enclosed in quotation marks because they are strings
- Are especially useful for referencing objects that are not currently on the screen (and so cannot easily be designated)
- Can be fully qualified or relative to the context in which the command is entered, as described in the following subsections

Pathnames differ from Ada names that are used as parameter values. Pathnames are enclosed in quotation marks; Ada names are not. Every object in the Environment library system has a string pathname. Ada names typically are not used to specify particular Environment objects (although Ada names are used as parameter values for specifying type enumerations, constants, or functions). See "Ada Identifiers," page 51, for a description of Ada name usage.

## Fully Qualified Pathnames

Every Environment object has a unique, *fully qualified pathname* that reflects the object's unique location in the hierarchy of Environment libraries. A fully qualified pathname traces the path from the Environment's root world (!) to the named object.

***Example.*** Consider the portion of the Environment hierarchy given in Figure 1. This figure shows a package called Calculation in user Anderson's home library; the fully qualified pathname for this package is !Users.Anderson.Calculation.

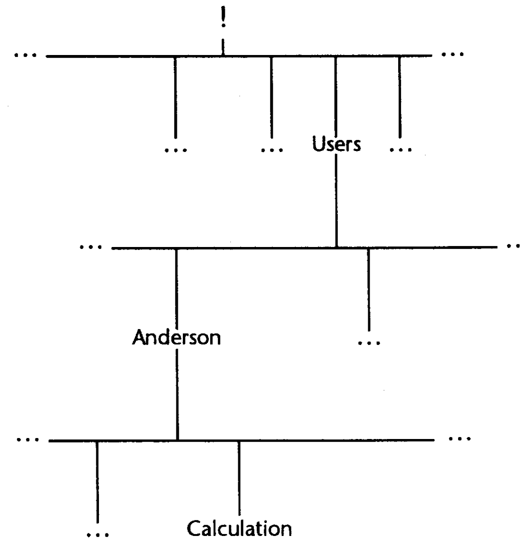



***Figure 1 !Users.Anderson.Calculation in the Environment Library Hierarchy***

As shown in this example, fully qualified pathnames:

- Begin with the reserved character !, which represents the root world
- Contain one or more name components separated by periods
  - The rightmost name component (Calculation) is the object's *simple name.*
  - The remaining name components (Users and Anderson) represent the nested objects between ! and the named object.

## Relative Pathnames

Although you can always refer to an object using its fully qualified pathname, it is often more convenient to use a *relative pathname* instead. A relative pathname traces the path to an object starting from some Environment location other than the root world. Thus, if you want to reference an object that is "in the neighborhood" of your current location in the Environment, you can use a relative pathname to specify the object relative to your current location.

***Example.*** Consider package !Users.Anderson.Calculation, shown in Figure 1 above. If you are in the library !Users, you can refer to this package as Anderson.Calculation. If you are in the library !Users.Anderson, you can refer to this package using its simple name, Calculation. Thus, a given relative pathname for an object:

- Starts with a name component or a context character other than the reserved character !
- Contains only a portion of the object's fully qualified name, omitting the portion that references the context

Relative pathnames are shorter than fully qualified pathnames because the Environment uses the context to fill in the missing name components. The process of mapping a context-dependent, relative pathname to the fully qualified pathname that references a unique Environment object is called *name resolution*; see "Context," below.

Note that special names can be used as components of pathnames—for example, user Anderson might use "<HOME>.Calculation" to reference !Users.Anderson.Calculation. When this is the case, the object referenced by the special name is the starting point for resolving any subsequent name components (see also "Special Names," page 15).

### Pathnames for Ada Objects

Pathnames can reference Ada subunits as well as Ada units. Thus, if package !Users.Anderson.Calculation has a subunit called Add, then the fully qualified pathname for this subunit is !Users.Anderson.Calculation.Add.

Names for Ada unit specifications and bodies can be explicitly differentiated using *attributes*—for example, the specification for package Calculation is more completely specified as !Users.Anderson.Calculation'Spec. See "Attributes," page 37.

## CONTEXT

*Context* is a location in the Environment library hierarchy that provides information for the resolution of relative pathnames. In general, two distinct but related contexts are of interest:

- The *current context*, which is "where you are" in the Environment when you enter a command.
- The *current library*, which is the nearest library to "where you are" when you enter a command. By default, the current library is the context for interpreting any relative pathnames used in the command.

The next subsections provide the details about current context and current library.

### Current Context

Current context is established by the object or image in the *current window*, which is the window that either contains the Environment cursor or is attached to a command window that contains the Environment cursor.

The current context is always either a library or an Ada unit. That is, if the current window contains:

- A library, the current context is that library.
- An Ada unit, the current context is that Ada unit.
- A file, the current context is the library that contains that file.
- Anything else, the current context is a library assigned on a case-by-case basis. For example, the current context for a user's mailbox is the user's home library.

You can reference the current context using a special pair of characters—namely, the brackets ([ ]) (see "Context Character Pair [ ]," page 24).

## Current Library

As the name implies, the current library is always a library (a directory, world, view, or subsystem). If the current context is:

- A library, the current library is that library
- An Ada unit, the current library is the library that contains that Ada unit

You can reference the current library using the dollar-sign context character ($) by itself (see "Context Character $," page 25).

## How the Environment Resolves Names

When you specify a string name as a parameter value in a command, the Environment must interpret that name by mapping it onto a unique object in the library hierarchy. The process of interpreting string names as Environment objects is called *name resolution.* The role of context in name resolution depends on the kind of string name:

- Fully qualified pathnames contain enough information to resolve to unique Environment objects, independent of the context in which these names are used. (These names are said to make *absolute* reference to objects.)
- Relative pathnames (and pathnames starting with context characters) are interpreted relative the current library—that is, the Environment uses the current library to map a context-dependent relative pathname onto a fully qualified pathname that references a unique Environment object.

### Library.Resolve Command

You can use the Library.Resolve command to see how the Environment resolves a given string name in a given context. This command is useful for testing string names before you use them in other commands in the same context. You can use this command to evaluate strings containing special names, context characters, wildcard characters, substitution characters, set notation, indirect file notation, and attributes.

The Library.Resolve command displays the fully qualified pathname(s) to which a given string expands. For example, assume that you enter the following command in a command window attached to user Anderson's home library:

```
Library.Resolve (Name_Of => "$");
```

The context character $ resolves to the current library, which is !Users.Anderson, as indicated in the command's output:

```
91/06/26 18:45:28 ::: [Library.Resolve ("$", "", TRUE, PERSEVERE);].
!USERS.ANDERSON
91/06/26 18:45:28 ::: [End of Library.Resolve command -- No errors detected].
```

***Figure 2 Resolution of Context Character $***

The Library.Resolve command can also be used to display the fully qualified pathname(s) to which a pair of source and destination names resolve (for example, in commands like Library.Move and Library.Copy). See "Substitution Characters for Specifying Destination Names," page 32.

## Examples

Following are some examples that show how relative pathnames are resolved in various Environment commands.

***Example 1.*** Assume that the Environment cursor is in a window containing the world !Users.Anderson, where you want to create a file called My_Notes. The following command, entered from a command window off the current window, creates !Users.Anderson.My_Notes. In this command, the simple name My_Notes is resolved relative to the world !Users.Anderson, which is the current context and hence the current library:

```
Text.Create (Image_Name => "My_Notes");
```

***Example 2.*** Assume that the cursor is in a window containing the file !Users.Anderson.My_Notes and that you want to view the specification of !Users.Anderson-.Calculation. The following command, entered from a command window off the current window, displays the visible part for the desired package. In this command, the simple name Calculation is resolved relative to the world !Users.Anderson, which is the current context and hence the current library:

```
Definition (Name => "Calculation", Visible => True);
```

***Example 3.*** Assume that the cursor is in a window containing the package specification !Users.Anderson.Calculation and that you want to create a file called !Users.Anderson.Temp. The following command, entered from a command window off the current window, creates the desired file. In this command, the simple name Temp is resolved relative to the world !Users.Anderson, because this world is the closest library that encloses the current context !Users.Anderson.Calculation:

```
Text.Create (Image_Name => "Temp");
```

***Example 4.*** Assume that the cursor is in a window that contains the body of !Users.Anderson.Calculation and that you want to display a subunit called Add. The following command, entered from a command window attached to the current window, displays the desired subunit. In this command, the qualified name Calculation.Add is resolved relative to the world !Users.Anderson because this world is the closest library that encloses the current context !Users.Anderson-.Calculation:

```
Definition (Name => "Calculation.Add");
```

Note that the simple name Add is not sufficient in this command, because this name cannot be resolved in !Users.Anderson. It is possible, however, to use the name [ ]Add in this context; see "The Character Pair [ ]," below.

## CONTEXT CHARACTERS

*Context characters* can be used to abbreviate pathnames. In general, they serve as a shorthand way to specify the context in which subsequent portions of a pathname are resolved. You can use context characters either to explicitly specify the default resolution context or to specify some related context. Different context characters specify different kinds of contexts:

- Some context characters specify an object that is related to the current context via the library hierarchy. These context characters "climb the tree structure" from the current context (or the library that encloses it) to arrive at the new context for resolving subsequent name components.
- Other context characters specify the Environment mechanisms for command-name resolution and Ada-name resolution. In particular, the backslash and the grave context characters cause subsequent name components to be resolved relative to the current session's searchlist and the current library's links, respectively. (For more information about searchlists and links, see the Session and Job Management (SJM) book and the Library Management (LM) book, respectively.)

Context characters are summarized in Table 4.

***Table 4 Context Characters***

| Character | Description |
|---|---|
| `!` | Resolves to the Environment's root world. |
| `!!`*`name`* | Resolves to the Environment's root world on an R1000® called *name* (only for commands in packages Archive and Queue). |
| `[]` | Resolves to the current context (either a library or an Ada unit). |
| `$` | Resolves to the current library, when used alone or at the beginning of a name. (The current library either is or encloses the current context.) |
| `$$` | Resolves to the current world, when used alone or at the beginning of a name. (The current world is a world, view, or subsystem that either is or encloses the current context.) |
| `^` | Resolves to the closest enclosing object (either a library or an Ada unit), when used alone or at the beginning of a name. |
| `$[`*`name`*`]` | Resolves to the closest enclosing library whose simple name is *name*. |
| `$$[`*`name`*`]` | Resolves to the closest enclosing world whose simple name is *name*. |
| `^[`*`name`*`]` | Resolves to the closest enclosing object whose simple name is *name*. |
| `\`*`foo`* | Evaluates the name *foo* relative to the searchlist for the current session. |
| *`library`*`` ` ``*`foo`* | Evaluates the simple name *foo* relative to the links associated with *library*. |
| *`unit`*`` ` ``*`foo`* | Evaluates the simple name *foo* relative to the objects that are directly visible in *unit*. |

## Context Character !

The exclamation mark (!) represents the root world in the Environment library system. The exclamation mark can be used alone or in front of other name components. A pathname that begins with the exclamation mark is a *fully qualified name*, which always references the same unique object, regardless of the context in which the command is entered. (See "Fully Qualified Pathnames," above.)

***Example.*** The following two commands, entered from any context, display the Environment's root world and the world that contains the Ada specifications for many Environment commands:

```
Definition ("!");
Definition ("!Commands");
```

## Context Character Pair !!

The double exclamation mark (!!), when followed by the machine name of an R1000 on the same network, represents the Environment's root world on that R1000. That is, a pathname that begins with !!*Machine_Name* is resolved relative to the root world on the named R1000. A name component following the machine name can be preceded by either a single exclamation mark (!) or a period (.).

Note that the double exclamation mark is interpreted only by selected commands that communicate to other R1000s across the network—namely, commands in packages Archive and Queue.

***Example.*** The following equivalent commands, entered from any context, copy the object !Projects.Tools from an R1000 called M_1 into same location on the current R1000:

```
Archive.Copy ("!!M_1!Projects.Tools");
Archive.Copy ("!!M_1.Projects.Tools");
```

## Context Character Pair [ ]

The special character pair [ ] represents the current context, which is either a library (directory, world, view, or subsystem) or an Ada unit. The current context is established by the object or image in the *current window*, which is the window that either contains the Environment cursor or is attached to a command window that contains the Environment cursor.

If the current window contains:

- A library, the current context is that library.
- An Ada unit, the current context is that Ada unit.
- A file, the current context is the library that contains that file.
- Anything else, the current context is a library assigned on a case-by-case basis. For example, the current context for a user's mailbox is the user's home library.

When the character pair [ ] precedes another name component, that name component is resolved relative to the current context. This is especially useful when the current context is an Ada unit.

***Example.*** Assume that the current window contains the body of package Calculation, so that the current context is !Users.Anderson.Calculation. In a command window created off the current window, you can use the following command to display the subunit Add:

```
Definition ("[]Add");
```

Note that [ ] causes the relative name Add to be resolved in the *current context* (!Users.Anderson.Calculation) rather than the *current library* (!Users.Anderson). Omitting [ ] results in an error, because the subunit Add is not defined directly in !Users.Anderson.

## Context Character $

The dollar sign ($) represents the current library, when used alone or at the beginning of a name. The current library is either the current context itself or the closest library enclosing the current context. The dollar sign thus resolves to a directory, world, view, or subsystem.

The dollar sign can be omitted at the beginning of a relative name, because it represents the library in which relative names are resolved. Put another way, every relative pathname is effectively preceded by an ***implicit*** $.

***Example.*** Assume, as in the previous section, that the current context is the body of package Calculation, so that the current library is !Users.Anderson. In a command window created off the current window, either of the following commands will display the subunit Add:

```
Definition ("Calculation.Add");
Definition ("$Calculation.Add");
```

In this context, Calculation.Add and $Calculation.Add are equivalent to each other (and to [ ]Add); all three resolve to !Users.Anderson.Calculation.Add.

## Context Character $$

The double dollar sign ($$) represents the *current world*, which is the world, view, or subsystem nearest to the current context. This is either the current context itself or the closest world, view, or subsystem enclosing the current context. Because the double dollar sign does not resolve to a directory, it is more restrictive than the single dollar sign ($), which can resolve to any library.

The double dollar sign is especially useful when you are working in views of subsystems (see Project Management (PM)). In particular, when you are working within subdirectories of a view, you can reference the enclosing view using the double dollar sign as an alternative to the special name "<VIEW>". The double dollar sign is usefully combined with the caret (^), as shown in Examples 3 through 5 in the next section.

***Example.*** Assume that you are editing an Ada unit in the Units subdirectory of a view named Rev1_Working. Now you want to display the Exports file in the State subdirectory of the same view. You can do this from the current context by entering the following command:

```
Definition ("$$State.Exports");
```

The double dollar sign "climbs up" the Environment hierarchy to the closest enclosing world, which is the view Rev1_Working. (Note that $$ skips over Units, which is a directory.) The subsequent name components (State.Exports) are then resolved relative to Rev1_Working.

## Context Character ^

The caret (^) represents the immediately enclosing (or parent) object; thus, the caret can resolve to a library or an Ada unit. The caret can be used repeatedly to "climb up" successive objects in the Environment library hierarchy, ultimately reaching the root of the library hierarchy (!). The parent of ! is defined to be !.

The caret is especially useful for referencing parents and siblings in the Environment library hierarchy (see Example 1), parents and siblings in a hierarchy of nested Ada subunits (see Example 2), and various objects within subsystems (see Examples 3 through 5).

***Example 1.*** Assume that the current window contains the home world, !Users-.Anderson, which is both the current context and the current library. In this context, the first of the following commands displays the parent world !Users, and the second command displays another user's home world (in this case, !Users.Miyata). In both commands, the caret climbs to the object that encloses the current library, resolving to !Users:

```
Definition ("^");
Definition ("^Miyata");
```

***Example 2.*** Assume that the current window contains the subunit !Users.Anderson.Calculation.Add, which is the current context. In this context, the first of the following commands displays the parent unit Calculation, and the second command displays a sibling subunit (in this case, Calculation.Subtract). In both commands, the character pair [ ] causes the caret to be resolved in the current context rather than the current library:

```
Definition ("[]^");
Definition ("[]^Subtract");
```

***Example 3.*** Assume that you are editing an Ada unit in the Units subdirectory of a view named Rev1_Working. Now you want to display the subsystem that contains Rev1_Working. You can do this using the special name "<SUBSYSTEM>"; alternatively, you can use the context character $$^, as in the following command. In this command, the double dollar sign climbs up to the closest enclosing world (the view Rev1_Working) and then the caret climbs up one step further to the enclosing subsystem:

```
Definition ("$$^");
```

***Example 4.*** Assume that you are editing an Ada unit in the Units subdirectory of a view named Rev1_Working. You want to display another view in the same subsystem—namely, Rev1_0_Spec. You can do this from the current context by entering the following command. Once again, the character string $$^ climbs to the enclosing subsystem, in which the name Rev1_0_Spec is resolved:

```
Definition ("$$^Rev1_0_Spec");
```

***Example 5.*** Assume that you are editing an Ada unit in the Units subdirectory of a view named Rev1_Working and that this view is in a subsystem called Mid_Layer. Now you want to display another subsystem in the same project library—namely, Top_Layer. You can do this from the current context by entering the following

command. In this command, the character string $$^^ climbs to the project library (the parent of subsystems Mid_Layer and Top_Layer):

```
Definition ("$$^^Top_Layer");
```

## Context Characters $[*name*]

The dollar sign can be used in combination with a bracketed simple name. The resulting combination resolves to the closest enclosing library with the specified simple name. In other words, $[*name*] climbs up the library hierarchy until it finds a directory, world, view, or subsystem with the specified name.

***Example.*** Assume that the current window contains the unit !Users.Anderson.Calculation. Then the following command, entered from the current context, displays the world !Users:

```
Definition ("$[Users]");
```

## Context Characters $$[*name*]

The double dollar sign can be used in combination with a bracketed simple name. The resulting combination resolves to the closest enclosing world, view, or subsystem with the specified simple name. That is, $$[*name*] climbs up the library hierarchy, skipping directories, until it finds a world, view, or subsystem with the specified name.

## Context Characters ^[*name*]

The caret can be used in combination with a bracketed simple name. The resulting combination resolves to the closest enclosing object with the specified simple name. That is, ^[*name*] climbs up the library hierarchy until it finds a library or Ada unit with the specified name.

## Searchlist Context Character

The backslash (\) causes the next name component to be evaluated in the searchlist for the current session. A searchlist is the list of libraries that is searched during command-name resolution; thus the backslash causes object names to be resolved as if they were command names.

The Environment resolves *name* by looking for an Ada unit with the specified name in each of the libraries listed in the current searchlist. If more than one of these libraries contains an Ada unit with this name, *name* resolves only to the first unit; however, some commands, such as Common.Definition, respond to this situation by displaying a menu of all the units that can be found via the searchlist. (Searchlists are described in the Session and Job Management (SJM) book.)

When the current session has the standard searchlist, the backslash provides a convenient way of referencing:

- The Ada specifications of Environment commands, particularly when you do not know the specific library in which to look (see Examples 1 through 3). This is true because the standard searchlist contains components for the libraries in

which Environment commands are defined (for example, !Commands, !Io, !Tools, and so on).

- Any Ada units for which links exist in the current library. This is true because the standard searchlist contains the component $`, which stands for the current library and its associated links (see "Ada Context Character `," below).

Note that you can use the !Commands.What.Search_List_Resolution command as a diagnostic tool if a name fails to resolve relative to the searchlist or if it resolves in an unexpected way.

***Example 1.*** Assume that the current window contains !Users.Anderson and that you want to view the Ada specification for the Environment command package Compilation. You can do this from the current context by entering the following command:

```
Definition ("\Compilation");
```

Because the standard searchlist lists the world !Commands, the above command displays !Commands.Compilation.

***Example 2.*** Assume the same current window as Example 1; this time you are specifically interested in the Compilation.Make command. You can display the Ada specification for this command as follows:

```
Definition ("\Compilation.Make");
```

This command displays !Commands.Compilation, highlighting the reference to procedure Make.

***Example 3.*** Assume that you are writing a program in !Users.Anderson and that you want to *with* the Environment's Text_Io package. To do this, you need to add a link to !Io.Text_Io. You can do this conveniently using the following command, provided that the current searchlist contains a component for !Io:

```
Links.Add ("\Text_Io");
```

## Ada Context Character `

The grave (`) causes the next name component to be evaluated using Ada semantics, either at the library level or at the unit level. The two general forms for use are *Library_Name`Simple_Name* and *Ada_Unit_Name`Simple_Name.*

### When ` Follows a Library Name

When the grave follows a library name and precedes a simple name, the simple name is resolved as if it were referenced in a *with* clause in an Ada unit in the specified library. Because visibility is supported at the library level by links, the simple name is looked up in the specified library's links and mapped onto the corresponding fully qualified pathname.

Note that units are always visible in the library in which they are defined, whether or not internal links exist for them. Therefore, the grave can resolve to Ada units declared in the specified library even if internal links are not created there (that is, even if the Create_Internal_Links library switch is set to False).

See also the 'L attribute, which searches external, internal, or both types of links, described in "Link Attribute 'L," below.

***Example 1.*** Assume that the current context is !Users.Anderson, which contains a link to the Ada unit !Users.Miyata.Factorial. Then the following command, entered in the current context, displays !Users.Miyata.Factorial:

```
Definition ("`Factorial");
```

***Example 2.*** Many components in the standard searchlist end with the grave—for example, $`. Such components cause the named library and its associated links to be searched when command names are resolved.

### When ` Follows an Ada-Unit Name

When the grave follows an Ada-unit name and precedes a simple name, the simple name is resolved exactly as it would be resolved during compilation. The simple name is looked up among the objects that are directly visible in the specified Ada unit, where direct visibility is determined by Ada visibility rules. Note that the grave does not "look through" renaming declarations or generic instantiations.

***Example 3.*** Assume that the current context is !Users.Anderson, which contains the Ada unit Calculation. Assume further that this unit *withs* package !Commands-.Text_Io but does not *with* !Users.Miyata.Factorial. Then the first of the following two commands displays !Commands.Text_Io, because Text_Io is directly visible in Calculation. However, the second of these two commands will quit, because Factorial is not directly visible in Calculation:

```
Definition ("Calculation`Text_Io");
Definition ("Calculation`Factorial");
```

## WILDCARDS FOR MATCHING ONE OR MORE PATHNAMES

*Wildcards* allow you to:

- Abbreviate long pathnames
- Specify multiple objects using a single general name

Using wildcards, you can construct a name that matches one or more pathnames based on spelling and name components. Wildcards can match portions of fully qualified or relative pathnames; furthermore, you can use wildcards in combination with context characters.

Wildcards can be used in any pathname. However, names that resolve to multiple objects are accepted only by parameters that can reference multiple objects (for example, the Existing parameter in the Library.Delete command). A parameter that must reference a unique object (such as the Name parameter of the Common-.Definition command) can accept a name containing wildcards only if it resolves to a unique object.

You can use the Library.Resolve command to test strings containing wildcards to ensure that they resolve to the desired names.

Note that wildcard characters differ from *restricted naming expressions* and *pattern-matching characters* (see "Restricted Naming Expressions," page 51, and "Pattern-Matching Characters," page 52).

Table 5 summarizes the four wildcards, which are described in detail in the following subsections.

*Table 5 Wildcard Characters*

| Character | Description |
|---|---|
| # | Matches a single character: T#### matches Tools. |
| @ | Matches zero or more characters: !U@.@.Tools matches !Users.Anderson.Tools. |
| ? | Matches zero or more nonworld name components: !Users.Anderson? matches !Users.Anderson and all objects in it except worlds. |
| ?? | Matches zero or more name components, including worlds and their contents: !Users?? matches !Users, all user home worlds, and their contents. |

Examples in the following subsections assume the library structure shown in Figure 3.

```
!Users.Anderson
   Calculation     : I Ada (Pack_Spec);
   Calculation     : I Ada (Pack_Body);
     .Add          : I Ada (Proc_Body);
     .Subtract     : I Ada (Proc_Body);
   Documentation   : Library (Directory);
   Login           : C Ada (Proc_Spec);
   Login           : C Ada (Proc_Body);
   Memo            : File (Text);
   Memo_Ps         : File (Text);
   Note            : File (Text);
   Note_Ps         : File (Text);
   S_1             : Session;
   To_Do           : File (Text);
   Tools           : Library (World);
```

*Figure 3 World !Users.Anderson*

## Naming Wildcard #

The pound sign (#) represents any single identifier character in a name, including the underscore (_). The pound sign does not match the name-component separator (.). You can use the pound sign more than once within a single name.

***Example 1.*** T#### matches any five-character name starting with T, such as Tools or To_Do.

***Example 2.*** If the world !Users.Anderson contains the files Memo, Memo_Ps, Note, and Note_Ps, then you can use the string !Users.Anderson.####_Ps to refer just to the two PostScript® files.

## Naming Wildcard @

The *at* sign (@) represents zero or more identifier characters in a name, including the underscore (_). You can use the *at* sign more than once within a single name. The *at* sign does not match the name-component separator (.), so the characters matched by a single instance of @ are always in the same name component. Thus, a given use of @ that matches an Ada-unit name includes no reference to subunits of the matched unit; referencing a subunit requires a further name component.

***Example 1.*** The name !Users.Anderson.Documentation can be abbreviated !U@.And@.Doc@.

***Example 2.*** If the current library is !Users.Anderson, the string @ references the set of objects in that library. However, the string @ does not match any subunits of those objects (that is, Calculation.Add and Calculation.Subtract are not referenced by @).

## Naming Wildcard ?

The question mark (?) matches zero or more contiguous whole name components but does not match name components for worlds (or the objects contained by those worlds). The question mark therefore can be used to match nested directories and their contents or Ada units and their subunits. The periods before and after the question mark are optional, so that the name A.?.B is equivalent to the name A?B. You can use the question mark more than once in a single name.

Although ? does not match name components for worlds, the name "?" appears to match a world name when the current library is a world. This occurs because relative pathnames are implicitly preceded by $, making "?" equivalent to "$?" (see "Context Character $"). When "?" evaluates to zero name components, the resulting naming expression is equivalent to "$", which is the current library.

***Example 1.*** In the current library !Users.Anderson, the name Calculation? references the package spec and body for Calculation and the subunits Calculation.Add and Calculation.Subtract.

***Example 2.*** The name !Users.Anderson? references !Users.Anderson and all objects in it except the world Tools and its contents. (The world !Users.Anderson is matched because ? can represent zero name components.)

***Example 3.*** Assume that the current context is the world !Users.Anderson.Tools. Then the command Delete ("?") deletes this world, because ? matches !Users.Anderson.Tools as well as its nonworld contents. To delete just the contents of this world, use the name @?. To delete just the nonworld contents of this world, use the name @'C(~World)?.

## Naming Wildcard ??

The double question mark (??) matches zero or more contiguous whole name components, including the name components for worlds and for the objects in those worlds. You can use the double question mark more than once in a single name. Note that the periods before and after the double question mark are optional, so that the name A.??.B is equivalent to the name A??B.

***Example 1.*** The name !Users?? matches the world !Users as well as the home worlds of all users and the contents of those worlds.

***Example 2.*** The name !Users.Anderson?? references all the objects in !Users.Anderson, including the world Tools and its contents. The name also references package Calculation and its subunits, as well as the directory Documentation and any of its contents. Furthermore, the name references the world !Users.Anderson because ?? can represent zero name components.

***Example 3.*** The name !?? matches all objects in the library system on a given machine.

## SUBSTITUTION CHARACTERS FOR SPECIFYING DESTINATION NAMES

Operations that move, copy, or rename objects usually have parameters for specifying:

- The names of the *source* objects on which to operate
- The names of the *destination* objects that will result from the operation

*Substitution characters* can be used as a shorthand way to specify destination names in move and copy operations. Substitution characters:

- Allow you to derive one or more destination names from portions of the specified source names
- Make it possible to move, copy, or rename multiple objects with a single operation
- Can save you typing if the source and destination names have name components in common

When you specify a destination name containing substitution characters, the Environment:

1. Maps these characters, from right to left, onto particular strings in the source name.
2. Expands the destination name by replacing each substitution character with the corresponding string from the source name, as summarized in Table 6.

***Table 6 Substitution Characters***

| Character | Description |
|---|---|
| @ | Expands to the string or strings matched by a wildcard (#, @, ?, ??) in the source name; allows you to copy, move, or rename multiple objects in a single operation. |
| # | Expands to a name component from the source name; saves typing when the source and destination names have components in common. |

### Testing Substitution Characters

You can use the Library.Resolve command to test a pair of source and destination names to ensure that the substitution characters expand as desired. Testing such pairs is recommended when you use them in the Library.Move command, which deletes the source objects.

To use the Library.Resolve command to test a pair of source and destination names:

- Specify the source name as the value of the Name_Of parameter.
- Specify the destination name as the value of the Target_Name parameter.

For example:

```
Library.Resolve (Name_Of     => "File@",
                 Target_Name => "Old_File@");
```

## Substitution Character @

You can use the substitution character *at* sign (@) to specify a single destination name that will expand to multiple destination names. The @ is therefore useful when copying, moving, or renaming multiple objects in a single operation.

The @ can be used in a destination name only if wildcards (#, @, ?, ??) are used in the source name. Starting from right to left, each @ in the destination name is mapped onto a wildcard in the source name. The strings matched by the wildcards are inserted in place of the corresponding instances of @. Multiple destination names result if a single wildcard matches multiple strings. If an @ in a destination name does not map onto a wildcard in the source name, then the @ in the destination name is replaced with the null string.

***Example 1.*** Assume that the current library contains files File1 through File50 and that you want to rename these objects Old_File1 through Old_File50. The following command does this:

```
Library.Rename (From => "File@",          -- wildcard @
                To   => "Old_File@");     -- substitution @
```

In each of the 50 destination names generated by this command, the substitution character @ in the destination (Old_File@) is replaced by one of the numbers matched by the wildcard @ in the source (File@). *Note that omitting the @ after Old_File would create a single file that would be overwritten by each file matched by File@, resulting in loss of data.*

***Example 2.*** Assume that user Anderson has several files called To_Do, one in his home world, one in his Documentation directory, and one in his Tools world. From the context !Users.Anderson, the following command copies two of these files (the one in the home world and the one in the Documentation directory):

```
Library.Copy (From => "?To_Do",
              To   => "@.Done");
```

The two destination names generated by this command are !Users.Anderson.Done and !Users.Anderson.Documentation.Done. In each name, the substitution character @ is replaced with the name components, if any, that are matched by the wildcard ?. The string !Users.Anderson is contributed by the current context.

Note that the To_Do file in the world Tools is not copied because the wildcard ? does not match world names. (If the source string were "??To_Do", then Tools.To_Do would be copied.) Note further that the substitution character # is not used in the destination string because it would match the first name component on the right (To_Do) instead of the first wildcard.

## Substitution Character #

The pound sign (#) can save you typing when a pair of source and destination names share common components. The # is mapped onto the next whole name component in the source name, counting from right to left. The destination name is expanded by inserting the name component in place of the #.

***Example.*** Assume that user Anderson wants to copy the file Memo from !Users.Anderson into the directory !Users.Anderson.Documentation. The following com-

mand accomplishes this, because the source and destination strings together construct the destination name !Users.Anderson.Documentation.Memo:

```
Library.Copy (From => "!Users.Anderson.Memo",
              To   => "!#.#.Documentation.#");
```

## SPECIAL CHARACTERS FOR SPECIFYING SETS OF OBJECTS

You can specify groups of objects using *set notation* or *indirect files*:

- Set notation allows you to supply a list of pathnames directly as a single parameter value. You can also use set notation to list the contrasting segments within pathnames that are otherwise the same.
- Indirect files allow you to save lists of pathnames in files and then reference those pathnames indirectly by specifying the appropriate filenames. Specifying an indirect file is:
  - Useful when you need to reference the same set of objects in multiple commands or in commands that are entered multiple times.
  - Equivalent to specifying the contained list of pathnames in set notation.

Tools often build indirect files for later use.

Note that wildcards serve as a third way of specifying multiple objects (see "Wildcards for Matching One or More Pathnames," above). Wildcards are especially useful when the pathnames to be specified share name components or characters within name components. Set notation and indirect files allow you to list unrelated pathnames.

### Set-Notation Characters [ ]

Enclosing brackets ([ ]) allow you to specify a set of pathnames with a single string. Brackets can enclose:

- A series of whole pathnames, which may be fully qualified or relative (see Example 1).
- A series of partial pathnames or *name segments*. Name segments include whole name components, attributes, wildcards, and even other bracketed sets. This is useful for specifying contrasting name segments within a set of pathnames that are otherwise the same (see Examples 2 and 3).

Elements enclosed in brackets can be separated by either commas or semicolons:

- When commas are used, the set is valid even if some of the elements cannot be resolved.
- When semicolons are used, the set is valid only if every element in the set resolves to at least one object. A naming error is raised and the command quits if any set element fails to resolve to an Environment object.

Using the semicolon serves as a safeguard against typographical errors made anywhere in the naming string (see Example 4). You cannot mix commas and semicolons within a single set of brackets.

You can use the tilde (~) to exclude elements from a set (see Example 5). The tilde is especially useful in sets that contain wildcards.

***Example 1.*** Assume that you want to archive the home worlds !Users.Anderson, !Users.Miyata, and !Users.Chavez. You can specify these names using set notation as in the following two commands. The first command can be entered from any context, because the brackets enclose fully qualified pathnames; the second command must be entered in the context !Users, because the brackets enclose relative pathnames.

```
Archive.Save
    (Objects => "[!Users.Anderson,!Users.Miyata,!Users.Chavez]");

Archive.Save (Objects => "[Anderson,Miyata,Chavez]");
```

***Example 2.*** Assume the same situation as Example 1. Whereas the commands in Example 1 listed whole pathnames in set notation, the following equivalent command lists only the contrasting name components in set notation:

```
Archive.Save (Objects => "!Users[Anderson,Miyata,Chavez]");
```

Note that the period separator can be omitted around name components that are enclosed in brackets.

***Example 3.*** Assume that users Anderson, Miyata, and Chavez all have a Tools library. It can be archived with the following command:

```
Archive.Save (Objects => "!Users[Anderson,Miyata,Chavez]Tools");
```

Note that name segments within brackets can contain multiple name components, as shown in the following command:

```
Archive.Save
    (Objects => "!Users[Anderson.Tools,Miyata.Utilities]");
```

***Example 4.*** Assume as in Example 1 that you want to archive the home worlds for Anderson, Miyata, and Chavez, but that you mistakenly misspell the username Anderson, so only two of the three elements resolve to Environment objects:

```
Archive.Save (Objects => "[Aderson,Miyata,Chavez]");
```

Because the comma separator was used, the command simply does what it can and archives only !Users.Miyata and !Users.Chavez.

If you would like a warning when a set element does not resolve, you can use the semicolon separator instead of the comma. The use of semicolons causes the following command to quit, because not all elements can be resolved:

```
Archive.Save (Objects => "[Aderson;Miyata;Chavez]");
```

***Example 5.*** Assume that you want to archive all home worlds *except* those for Anderson, Miyata, and Chavez. To do this, you can use the @ to match all usernames and the tilde to indicate exceptions:

```
Archive.Save (Objects => "!Users[@,~Anderson,~Miyata,~Chavez]");
```

## Indirect-File Prefix _

When the underscore (_) precedes a pathname, that name is interpreted as the name of an *indirect file*. An indirect file is an ordinary text file that contains a list of one or more pathnames and/or naming expressions. When an indirect file is given as a parameter value, the Environment internally converts the file's contents into set notation. An indirect file is thus a way of maintaining a list of objects that then can be referenced using a single name.

Note that activities can be used as indirect files, provided that you specify the activity name with the underscore prefix. (See the Project Management (PM) book for more information about activities.)

Indirect files can contain any of the following:

- Fully qualified and/or relative pathnames. Note that fully qualified pathnames are recommended, because their resolution is not affected by the context in which you reference the indirect file.
- The names of other indirect files, provided that each filename has the underscore prefix.
- Wildcards, attributes, and bracketed set notation.

Multiple names within an indirect file may be placed:

- In a single column, one to a line. All lines in the file must end with either a newline or a semicolon, as described below.
- On the same line. All names must be separated either by commas or semicolons, as described below.

Separators for multiple names in indirect files affect command behavior, similar to separators used in set notation:

- Newlines or commas cause unresolvable names to be ignored without error messages.
- Semicolons require that every name in the file resolve to at least one object, or else a naming error is raised. The semicolon is useful for detecting misspelled names or for verifying that all of the named objects do in fact exist.

Commas and semicolons cannot be mixed in the same indirect file.

***Example 1.*** Assume that you regularly need to archive the home worlds for users Chavez, Anderson, Miyata, and Katz. In addition, you want to archive a set of project tools whose fully qualified names are listed in an indirect file called Project_Tools that is maintained by user Miyata.

You can create a text file with an appropriate name (for example, Archive_List) and enter the desired pathnames as shown. Fully qualified pathnames are listed so that the indirect file references the same objects independent of the context in which it is used:

```
!Users.Chavez
!Users.Anderson
!Users.Miyata
!Users.Katz
_!Users.Miyata.Project_Tools
```

Note that it is possible to enter multiple names on the same line using explicit name separators (in this case, commas)—for example:

```
!Users.Chavez,!Users.Anderson
!Users.Miyata,!Users.Katz,_!Users.Miyata.Project_Tools
```

You can now enter the Archive.Save command, specifying Archive_List as an indirect file. As shown, you can use the fully qualified name or a relative name for the indirect file, depending on the context in which you enter the command:

```
Archive.Save ("_!Users.Operator.Archive_List");
Archive.Save ("_Archive_List");
```

***Example 2.*** Assume that user Anderson wants to archive two libraries in his home world—namely, !Users.Anderson.Documentation and !Users.Anderson.Tools. To do this, he creates in his home world a text file called My_List that contains the following two relative pathnames:

```
Documentation.@
Tools.?
```

He has used relative pathnames because he knows he is likely to enter the Archive.Save command from the context of his home world, causing My_List and its contents to be resolved in !Users.Anderson:

```
Archive.Save ("_My_List");
```

If, instead, user Anderson were to perform the archive operation from some other context, say !Commands, he might try to use a fully qualified name for the indirect file, as follows:

```
Archive.Save ("_!Users.Anderson.My_List");
```

However, although the indirect file's name will resolve correctly, the names inside the file will not—relative to the current library, they will resolve to !Commands-.Documentation.@ and !Commands.Tools.?. The following command, in which brackets enclose the indirect file's simple name, will allow both the indirect file's name and the names inside the file to resolve correctly:

```
Archive.Save ("!Users.Anderson.[_My_List]");
```

## SPECIAL CHARACTERS FOR SPECIFYING DELETED OBJECTS

Braces ({ }) allow you to reference objects that have been deleted but not expunged. Such objects can be referenced only if they have a nonzero retention count. Deleting an object whose retention count is zero effectively expunges it as well. (For commands that delete, expunge, and set retention counts, see the Library Management (LM) book.)

An object is deleted when you delete its default version. Library entries for deleted objects can be displayed by expanding the library image. Entries for deleted objects are enclosed in braces.

Names in braces can resolve to nondeleted objects as well as deleted objects. Thus, wildcards can be used within braces to reference a set of objects that includes both deleted and nondeleted objects. Furthermore, commands such as Library.Undelete, which operate on deleted objects, implicitly insert braces around the name you supply. Note that some commands, such as Library.Copy, accept names only for nondeleted objects.

***Example.*** Assume that you delete a file named To_Do that has a retention count greater than zero. Until the deleted file is expunged, you can view it by entering the following command:

```
Definition ("{To_Do}");
```

## ATTRIBUTES

Attributes are qualifiers that allow you to specify objects using properties such as class, version, Ada part, compilation state, and so on. Attributes are useful for:

- Clarifying otherwise ambiguous pathnames. For example, a name such as !Users.Anderson.Calculation can refer either to the visible part or to the body of an Ada unit. The 'Spec or 'Body attribute can be used to make the name unambiguous, as in !Users.Anderson.Calculation'Body.
- Specifying nondefault properties of objects. For example, pathnames such as !Users.Anderson.Calculation'Body reference default versions of objects. You can request a nondefault version using the version attribute 'V, as in !Users.Anderson.Calculation'Body'V(2).
- Specifying sets of objects that have particular properties (especially when used along with wildcards). For example, a name like ?'S(Installed) matches all installed units in the current library.

The properties you can specify via attributes are summarized in Table 7.

***Table 7 Attributes***

| Attribute | Description |
|---|---|
| `'Body` | Resolves to the body of an Ada unit: `"Calculation'Body"` |
| `'C(`*arguments*`)` | Specifies an object's class or subclass: `"@'C(Library)"` |
| `'If(`*arguments*`)` | Specifies whether an object is controlled, checked in, checked out, or frozen: `"@'If(Frozen)"` |
| `'L(`*argument*`)` *or* `'L` | Specifies a library's links for resolution of subsequent name components. Omitting *argument* specifies entire set of links: `"Tools'L.Text_Io"` |
| `'N(`*nickname*`)` | Specifies an Ada subprogram's nickname: `"Example.Overloaded'N(First)"` |
| `'S(`*arguments*`)` | Specifies an Ada unit's compilation state: `"@'S(Source)"` |
| `'Spec` | Resolves to the visible part (specification) of an Ada unit: `"Calculation'Spec"` |
| `'Spec_View(`*activity*`)` *or* `'Spec_View` | Specifies the spec view listed for a given subsystem in *activity*. Omitting *activity* uses the default activity: `"!Project.Interface.@.Units'Spec_View"` |
| `'T(`*target_key*`)` | Specifies a library's target key: `"@_Working'T(Mc68020_Bare)"` |
| `'V(`*arguments*`)` | Specifies an object's version: `"My_File'V(5)"` |
| `'View(`*activity*`)` *or* `'View` | Specifies the load view listed for a given subsystem in *activity*. Omitting *activity* uses the default activity: `"Command_Interpreter'View"` |

## Attribute Syntax

Attributes on pathnames are syntactically similar to Ada attributes. Pathname attributes are postfixes of the form *'attribute* or *'attribute(arguments)*, where *attribute* is one of the strings listed in the "Attribute" column in Table 7. (Predefined arguments for each attribute are listed in separate sections below.)

Although all pathname attributes are postfixes:

- Some attributes must be postfixed to entire pathnames.
- Other attributes can be postfixed to individual name components within pathnames, thus affecting the resolution of subsequent name components.

Multiple attributes can be used in a single name (see "Specifying Shared Properties," below). No separator characters are required between adjacent attributes.

Among attributes that accept arguments:

- Some attributes accept multiple arguments separated by commas (see "Specifying Disjoint Properties," below).
- All attributes allow you to precede an argument with a tilde (~) to denote an excluded property; see "Excluding Properties," below.

## Specifying Shared Properties

You can use multiple attributes to specify a set of objects that share certain properties. When multiple attributes are used in a name, the name resolves to the *intersection* of the sets of objects that have each property.

***Example 1.*** The name @'Body'S(Installed) matches Ada-unit bodies in the installed state.

***Example 2.*** The name @'C(World)'T(R1000) matches worlds with target key R1000.

***Example 3.*** The name @'C(File)'C(Ada) matches no objects, because no object is both a file and an Ada unit.

## Specifying Disjoint Properties

You can use multiple arguments in a single attribute to specify disjoint properties —that is, properties that may but need not be shared. When multiple arguments are specified in a name, the name resolves to the *union* of the sets of objects that have each property.

***Example 1.*** The name @'C(Ada,File) matches all Ada units and all files in the current library.

***Example 2.*** The name @'S(Installed, Coded) matches either installed or coded Ada units in the current library.

Alternatively, disjoint properties can be expressed using set notation, so that the name in Example 2 is equivalent to the name @['S(Installed),'S(Coded)]. Set notation can also be used to specify arbitrary combinations of attributes. For example, @['Spec,'C(File)] matches the set of Ada-unit specifications and files.

## Excluding Properties

You can use the tilde (~) with attribute arguments to match objects that do not have particular properties.

***Example 1.*** The name @'C(~Binary) matches all objects in the current library except binary files. This name can match Ada units, text files, switch files, libraries, and so on.

***Example 2.*** The name @'C(File)'C(~Binary) matches all files except binary files in the current library. That is, this expression matches the intersection of the set of all files and the set of all nonbinary objects. This name can match text files and switch files, but not Ada units and libraries.

Note that this name is not equivalent to @'C(File,~Binary), which matches the union of files and all nonbinary objects. Because the set of files contains binary objects, @'C(File,~Binary) is equivalent to @.

***Example 3.*** The name ?'Body'S(~Coded) matches all Ada-unit bodies that are not coded.

## Ada-Part Attributes: 'Spec and 'Body

By default, Ada-unit names reference both visible parts and bodies. The 'Spec and 'Body attributes restrict name resolution to either the visible part or the body, respectively.

Typically, Ada-part attributes are postfixed to entire pathnames, as in !Users.Anderson.Calculation'Body. However, Ada-part attributes can be postfixed to individual name components within a pathname, provided that these name components reference Ada units. Name components to the right of such an attribute resolve to objects in the specified Ada part. In most cases, putting the attribute at the end of the name is equivalent to putting it within the name, because the Environment automatically searches both Ada parts until it finds the desired object.

***Example.*** Assume that package !Users.Anderson.Calculation contains a unique subprogram called Plus. Either of the following two commands highlights Plus in the visible part of Calculation:

```
Definition ("!Users.Anderson.Calculation'Spec.Plus");
Definition ("!Users.Anderson.Calculation.Plus'Spec");
```

Note that these two commands yield different results if Plus is overloaded, with one declaration in the visible part and one declaration in the body of Calculation. The first command will search the visible part of Calculation and find the Plus declared there. The second command will search both the visible part and the body of Calculation and display a menu listing the specs for both Plus declarations.

## Version Attribute 'V

The Environment retains multiple versions of objects such as files and Ada units, provided that the retention count for those objects is nonzero. Of these versions, one is the *default version* and the others are *nondefault versions.* Normally, pathnames reference default versions of objects. You can reference retained nondefault versions of objects by using the version attribute 'V. (For more information about retention counts, see Library Management (LM) book. The 'V attribute can take one or more of the arguments listed in Table 8.

***Table 8 Arguments for the Version Attribute 'V***

| Argument | Description |
|---|---|
| All | Matches *all* versions of the object. |
| Any | Matches only the *default* version of the object, which may but need not be the newest version. |
| Max | Matches the *newest* version of the object, which may but need not be the default version. |
| Min | Matches the *oldest* retained version of the object. |
| *n* | Matches the version with version number *n*. Multiple version numbers can be listed, separated with commas. |
| −*n* | Matches the *n*th version preceding the newest version. |

***Example.*** Assume that !Users.Anderson.Calculation has retention count 3 and that the default version is version 7 (so that the three retained nondefault versions are versions 6, 5, and 4).

- The name Calculation'V(All) references versions 7, 6, 5, and 4.
- The name Calculation'V(4,6) references versions 4 and 6.
- The names Calculation'V(Any) and Calculation'V(Max) both reference version 7.
- The name Calculation'V(Min) references version 4.
- The name Calculation'V(−2) references version 5, whereas the name Calculation'V(2) cannot be resolved (version 2 is not retained).

## Class Attribute 'C

Environment objects are grouped into classes, which are further subdivided into subclasses. You can reference specific classes or subclasses of objects using the class attribute 'C with one or more arguments. Arguments can be class or subclass names, which are listed in the following tables. Many subclass names have both a fully explicit form and an abbreviated form. Note that object classes and subclasses appear in expanded library displays.

***Example 1.*** Assume that every subdirectory in a given world has a file called To_Do in it and that you want to delete those files. An easy way to do this is to enter the following command from the context of the enclosing world:

```
Library.Delete ("@'C(Directory).Foo");
```

***Example 2.*** The following command produces a list of all subsystems, including combined subsystems, on a given R1000:

```
Library.Resolve ("!??'C(Subsystem, Comb_Ss)");
```

***Example 3.*** The following command destroys all the views in the subsystem in which it is entered:

```
Cmvc.Destroy_View ("@'C(Load_View, Spec_View, Comb_View)");
```

***Example 4.*** The following command lists all spec views in all subsystems in a program library called !Projects:

```
Library.Resolve ("!Projects.@.@'C(Spec_View)");
```

***Example 5.*** The following command lists all subsystems in the current library that contain a view called Rev11_0_Spec:

```
Library.Resolve("@'C(Subsystem).Rev11_0_Spec");
```

***Example 6.*** The following command deletes all printable PostScript files in a given library:

```
Library.Delete ("@'C(Ps)");
```

*Table 9 Arguments for the Class Attribute 'C: Object Classes*

| Argument | Description |
|---|---|
| Ada | Ada program units of any subclass |
| Archived_Code | Objects appearing in a subsystem view for a code-only unit |
| File | Files of any subclass |
| Group | Groups defined for access control |
| Library | Libraries of any subclass |
| Null_Device | Devices that accept output and discard it |
| Pipe | Pipes |
| Session | User session objects |
| Tape | Tape drives in the system |
| Terminal | Terminals in the system |
| User | Users in the system |

*Table 10 Arguments for the Class Attribute 'C: Library Subclasses*

| Short Form | Full Form | Description |
|---|---|---|
| Comb_Ss | Combined_Subsystem | Combined subsystem (can contain only combined views) |
| Comb_View | Combined_View | Combined view of a subsystem |
| Directory | | Directory |
| Load_View | | Load view of a subsystem |
| Spec_View | | Spec view of subsystem |
| Subsystem | | Spec_Load subsystem (can contain spec, load, or combined views) |
| System | System_Subsystem | System (object managed using package Cmvc_Hierarchy) |
| Sys_View | System_View | View in a system |
| World | | World |

*Table 11 Arguments for the Class Attribute 'C: Ada Subclasses*

| Short Form | Full Form | Description |
|---|---|---|
| Alt_List | Alternative_List | Insertion point for alternative list |
| Comp_Unit | Compilation_Unit | Compilation unit that has not been semanticized |
| Context | Context_List | Insertion point for context clause |
| Decl_List | Declaration_List | Insertion point for declaration list |
| Func_Body | Function_Body | Function body |
| Func_Inst | Function_Instantiation | Generic function instantiation |
| Func_Ren | Function_Rename | Function rename |
| Func_Spec | Function_Spec | Function specification |
| Gen_Func | Generic_Function | Generic function |

*Table 11 Arguments for the Class Attribute 'C: Ada Subclasses (continued)*

| Short Form | Full Form | Description |
|---|---|---|
| Gen_Pack | Generic_Package | Generic package |
| Gen_Param | Generic_Parameter_List | Insertion point for generic parameter |
| Gen_Proc | Generic_Procedure | Generic procedure |
| Insertion | Nonterminal | Insertion point |
| Load_Func | Loaded_Function_Spec | Code-only function |
| Load_Proc | Loaded_Procedure_Spec | Code-only procedure |
| Main_Body | Main_Function_Body | Main function body |
| Main_Body | Main_Procedure_Body | Main procedure body |
| Main_Func | Main_Function_Spec | Main function specification |
| Main_Proc | Main_Procedure_Spec | Main procedure specification |
| Pack_Body | Package_Body | Package body |
| Pack_Inst | Package_Instantiation | Generic package instantiation |
| Pack_Ren | Package_Rename | Package rename |
| Pack_Spec | Package_Spec | Package specification |
| Pragma | Pragma_List | Insertion point for pragma |
| Proc_Body | Procedure_Body | Procedure body |
| Proc_Inst | Procedure_Instantiation | Generic procedure instantiation |
| Proc_Ren | Procedure_Rename | Procedure rename |
| Proc_Spec | Procedure_Spec | Procedure specification |
| Statement | Statement_List | Insertion point for statement |
| Subp_Body | Subprogram_Body | Subprogram body |
| Subp_Inst | Subprogram_Instantiation | Generic subprogram instantiation |
| Subp_Ren | Subprogram_Rename | Subprogram rename |
| Subp_Spec | Subprogram_Spec | Subprogram specification |
| Task_Body | | Task body |

*Table 12 Arguments for the Class Attribute 'C: File Subclasses*

| Short Form | Full Form | Description |
|---|---|---|
| Activity | | Activity file (used with subsystem development) |
| Binary | | Binary file |
| Cmvc_Acc | Cmvc_Access | File containing CMVC access-control information for a view or subsystem |
| Cmvc_Db | Cmvc_Database | CMVC database (stores source-control information in subsystems) |
| Code_Db | Code_Database | Code saved for a subsystem load view |
| Compat_Db | Compatibility_Database | Compatibility database for a subsystem |
| Config | Configuration | Configuration pointer for CMVC |
| Dictionry | Dictionary | For future development |

*Table 12 Arguments for the Class Attribute 'C: File Subclasses (continued)*

| Short Form | Full Form | Description |
|---|---|---|
| Documents | Document_Database | Document database (part of Rational Design Facility) |
| Elements | Element_Cache | Element cache for storing permanent collections of Ada program elements (part of Rational Design Facility) |
| Exe_Code | Executable_Code | Executable module generated by the linker of the Cross-Development Facility |
| File_Map | Pure_Element_File_Map | File map |
| Log | | Log file |
| Mail | | Collections of messages (part of Rational Network Mail) |
| Mail_Db | Mail_Database | User's mailbox (part of Rational Network Mail) |
| Markup | | Text file containing markup, generated from abstract document (part of Rational Design Facility) |
| Msg_In | Incoming_Mail_Message | For future development |
| Msg_Out | Outgoing_Mail_Message | For future development |
| Obj_Code | Object_Code | Relocatable object module generated by the compilation system of the Cross-Development Facility |
| Objects | Object_Set | Permanent collection of Directory.Object |
| Ps | Postscript | PostScript file |
| Search | Search_List | Searchlist file |
| Switch | | Switch file |
| Swtch_Def | Switch_Definition | Switch-definition file |
| Text | | Text file |
| Venture | | A collection of work orders for CMVC |
| Work | Work_Order | Work order for CMVC |
| Work_List | Work_Order_List | Work-order list for CMVC |

## Compilation-State Attribute 'S

Ada units can be in one of four compilation states. Ada-unit states can be specified using the 'S attribute with one or more of the arguments listed in Table 13.

Note that:

- The 'S attribute must follow a name component that references an Ada unit.
- At least one argument must be specified.
- Any prefix of these arguments can be used.

***Example 1.*** The name !Users.Anderson?'S(Coded) specifies all units in the coded state in Anderson's home library.

***Table 13 Arguments for the Compilation-State Attribute 'S***

| Shortest Form | Full Form | Description |
|---|---|---|
| A | Archived | Matches units in the archived state. |
| S | Source | Matches units in the source state. |
| I | Installed | Matches units in the installed state. |
| C | Coded | Matches units in the coded state. |

***Example 2.*** The name !Users.Anderson.Tools?'S(A,S) matches all archived or source units in Anderson's tools library.

***Example 3.*** The following command compiles all source subunits of installed parent units in the current library:

```
Compilation.Make("?'S(Installed).?'S(Source)");
```

## Nickname Attribute 'N

Names of subprograms in an Ada unit can be overloaded. A subprogram can be given a unique nickname with the Nickname pragma, which follows the declaration of the subprogram. You can clarify an overloaded Ada-unit name by using the 'N attribute with the unique nickname as its argument. The 'N attribute must follow a name component that references an Ada unit.

***Example.*** Assume the following Ada unit:

```
package Example is
    procedure Overloaded (X : Integer);
    pragma Nickname ("First");
    procedure Overloaded (X : Float);
end Example;
```

The following command displays package Example and highlights the identifier in the first of the two procedures:

```
Def ("Example.Overloaded'N(First)'Spec");
```

## Link Attribute 'L

The attribute 'L causes the next name component to be evaluated using the set of links associated with a given library. A set of links consists of the simple names of Ada units mapped onto their fully qualified pathnames (refer to the Library Management (LM) book).

The 'L attribute must follow a name component that references a library. The Environment resolves *Library_Name*'L.*Simple_Name* by:

- Looking for *Simple_Name* in the links for *Library_Name*
- Mapping that simple name onto the corresponding fully qualified pathname

Although the link attribute 'L is similar to the grave (`), they differ in three important ways (see "Ada Context Character `," above):

- The grave always looks through external and internal links, whereas the 'L attribute has arguments that allow you to restrict the search to just one type of link. These arguments are summarized in Table 14.
- The grave will resolve to Ada units declared in the specified library even if internal links are not created in that library (that is, even if the Create_Internal-_Links library switch is set to False). In contrast, the link attribute 'L can resolve simple names only if the specified library has links for them.
- The grave can occur after Ada-unit names (with more general meaning), whereas 'L can follow only library names.

***Table 14 Arguments for the Link Attribute 'L***

| Argument | Description |
|---|---|
| Any | Resolves the next name component relative to external and internal links (the default if no argument is specified). |
| External | Resolves the next name component relative to external links only. External links reference Ada units outside the closest enclosing world. |
| Internal | Resolves the next name component relative to internal links only. Internal links reference Ada units within the closest enclosing world or any of its subdirectories. |

Note that:

- Any prefix of these arguments can be used.
- If no argument is given, Any is assumed, and a period must separate the link attribute and the subsequent simple name.
- If an argument is specified, no delimiter is required to separate the link attribute and the subsequent simple name.

***Example 1.*** Assume that the current context is !Users.Anderson, which contains a link to the Ada unit !Commands.Text_Io. Then the following command, entered in the current context, displays !Commands.Text_Io:

```
Definition("'L.Text_Io");
```

Note that 'L.Text_Io is equivalent to 'L(Any).Text_Io.

***Example 2.*** Assume that !Users.Anderson.Tools contains a locally defined unit called Text_Io, for which there is an internal link. Then the following command, entered in the context !Users.Anderson, displays !Users.Anderson.Tools.Text_Io:

```
Definition("Tools'L(Int)Text_Io");
```

## Target-Key Attribute 'T

All Environment worlds and views have target keys associated with them. The target key for a world or view determines the compilation system that is used when Ada units are promoted. By default, worlds and views are created with the R1000 target key, which causes units to be compiled for execution on the R1000. In addition, if your site has a Cross-Development Facility product for a specific target processor, you can create worlds or views with the target key for that processor.

You can use the target-key attribute 'T with one or more arguments to reference worlds or views with specific target keys. The arguments to 'T must be valid names of target keys. The Compilation.Set_Target_Key command with default parameters displays a list of valid target keys.

***Example 1.*** The name !??'C(World)'T(~R1000) matches all worlds with target keys other than R1000.

***Example 2.*** Assume that the current context is a project library containing several subsystems and that each subsystem contains a path for the R1000 and for a Motorola® MC68020 microprocessor. The following command makes a release from the MC68020 path of each subsystem:

```
Cmvc.Release ("@.@_Working'T(Mc68020_Bare)");
```

## View Attributes 'Spec_View and 'View

Programs that are composed of subsystems can be executed only if an *activity* is set up. An activity is an "execution table" that specifies one load view for each subsystem from which a spec view has been imported. The activity may optionally list the imported spec views for record-keeping (see the Project Management (PM) book).

The 'Spec_View and 'View attributes resolve names in the context of some activity. That is, when postfixed to the name of a subsystem or an object in a subsystem:

- The 'Spec_View attribute searches the activity and resolves to the spec view that is listed there for the named subsystem.
- The 'View attribute searches the activity and resolves to the load view that is listed there for the named subsystem.

When used without an argument, these attributes resolve names relative to the *current activity.* The current activity is the activity for the job that is currently executing; it may but need not be the default activity. An error results if no current activity is defined. The current activity is returned by the Activity.The_Current_Activity function.

The 'Spec_View and 'View attributes can each take an argument to specify an activity other than the current activity. The argument must name an activity or an error results. The specified activity name is resolved relative to the name preceding the attribute. For example, My_Subsystem'Spec_View(My_Activity) looks for an activity called My_Subsystem.My_Activity. To specify an activity outside that context, use a fully qualified pathname or use the appropriate context characters.

***Example 1.*** Assume that the current context is the project library !Programs.Mail, which contains three subsystems—namely, Command_Interpreter, Mailbox, and Mail_Utilities. Assume further that the current activity contains the following entries for these subsystems:

```
Subsystem            | Spec View   | Load View     | Context

COMMAND_INTERPRETER  | REV1_0_SPEC | REV1_WORKING  | !PROGRAMS.MAIL
MAILBOX              | REV1_3_SPEC | REV1_3_1      | !PROGRAMS.MAIL
MAIL_UTILITIES       | REV1_2_SPEC | REV1_2_2      | !PROGRAMS.MAIL
```

Then you can use the following command to display the load view from the Command_Interpreter subsystem that is used during execution—Rev1_Working:

```
Definition ("Command_Interpreter'View");
```

***Example 2.*** Assume the same subsystems and current activity as in Example 1. Each of the following equivalent commands displays the Units directory of Command_Interpreter.Rev1_Working:

```
Definition ("Command_Interpreter'View.Units");
Definition ("Command_Interpreter.@.Units'View");
```

***Example 3.*** Assume the same subsystems and current activity as in Example 1. Assume further that !Programs.Mail also contains an activity called Alternate-_Activity, which is *not* being used as the current activity. Each of the following commands displays the load view that is listed for the Command_Interpreter subsystem in Alternate_Activity (the second command uses a context character to specify Alternate_Activity relative to the context Command_Interpreter):

```
Definition
  ("Command_Interpreter'View(!Programs.Mail.Alternate_Activity)");
Definition ("Command_Interpreter'View(^Alternate_Activity)");
```

***Example 4.*** Assume that your project uses a tool that is written in subsystems. For you to use this tool, your searchlist must contain a component that references the latest spec view from the tool's topmost subsystem. However, each time a new spec view is created for that tool, your searchlist requires updating. You can avoid this problem by entering a component like the following in your searchlist. This component always points to the current spec view for the tool, provided that the appropriate activity is updated to reflect the new spec view:

```
!Project.Tools.Interface.@.Units'Spec_View
```

## Conditional Attribute 'If

The attribute 'If specifies one or more conditions. An object must satisfy at least one in order to be matched. The conditions are specified as arguments to the 'If attribute. These arguments are listed in Table 15.

***Table 15 Arguments for the Conditional Attribute 'If***

| Short Form | Full Form | Description |
|---|---|---|
| Controlled | | Matches objects if they are controlled. |
| In | Checked_In | Matches objects if they are controlled and checked in. |
| Out | Checked_Out | Matches objects if they are controlled and checked out. |
| Frozen | | Matches objects if they are frozen. |

***Example 1.*** The name @'If(~Controlled) matches all objects in the current library that are not controlled.

***Example 2.*** The name @'If(Out) matches all controlled, checked-out objects in the current library.

***Example 3.*** The name @'If(~In) matches the set of objects that is not controlled or is not checked in. This name is equivalent to @'If(~Controlled, Out).

***Example 4.*** The name @'If(Controlled)'If(Frozen) matches the set of objects that is both controlled and frozen. In contrast, @'If(Controlled, Frozen) matches the set of objects that is either controlled or frozen.

## CONTEXT CHARACTERS FOR USE IN THE DEBUGGER

Various Rational debugger commands operate on stack frames, tasks, and Ada units in the program being debugged. References to these objects are prefixed by special characters, as summarized in Table 16.

***Table 16 Debugger Context Characters***

| Character | Description |
|---|---|
| `.name` | Resolves *name* to an Ada unit that is referenced in the program being debugged. Subsequent name components resolve to objects declared in that Ada unit: `Source (".Initialize.Status")` |
| `_n` | Refers to the stack frame with the specified frame number (1 is the top frame). Subsequent name components resolve to objects in the subprogram whose activation is contained in the frame: `Put ("_3.Status")` |
| `%name, %n` | Refers to the task with the specified task name or number. Subsequent name components resolve to objects declared in the task: `Put ("%Debug_Shell._3.Status")` |

Debugger commands also accept names preceded by context characters such as the exclamation mark (!), dollar sign ($), double dollar sign ($$), and caret (^) (see "Context Characters," page 23). Names that are not preceded either by general or debugger-specific context characters are called *unqualified names*. Unqualified names are interpreted relative to the debugger's control and evaluation contexts (see the Debugging (DEB) book).

### Unit-Name Prefix .

When the period precedes one or more name components, the first (or only) name component resolves to an Ada unit that is referenced by the program being debugged. Subsequent name components denote some object declared in the named Ada unit. The period is used as a prefix only in the debugger; when the period appears elsewhere in a name, it serves as a name-component separator.

***Example.*** Assume that you are debugging a program that *withs* a subprogram called Initialize. In debugger commands such as the following, you can refer to this subprogram as .Initialize:

```
Source (".Initialize");
```

Furthermore, you can refer to an object (such as a variable) in this subprogram by adding the appropriate name component:

```
Source (".Initialize.Status");
```

### Stack-Frame Prefix _

When the underscore character precedes a number, that number refers to a frame on a *call stack* for a given task. The call stack for a task serves as a record of that task's flow of control. When a subprogram is called (or *activated*) in a given task,

a frame for that call is pushed onto the appropriate stack. The top frame in a stack always contains a reference to the most recently called subprogram in the task.

You must reference stack frames whenever you want to display or modify values for objects such as variables. That is, the value of an object in a given subprogram must be referenced relative to a particular frame that contains a particular activation for that subprogram.

Stack frames are numbered starting at the top with 1. Thus, _4 refers to frame number 4 (fourth frame from the top). Frames are alternately numbered from the bottom using negative numbers. Unless it is preceded by a task name or number (see "Task Identifier Prefix %," below), a frame number refers to the call stack for the last task to stop or for the control context task, if such a task has been set.

***Example 1.*** Assume that frame _3 of the current task contains an activation of the subprogram Initialize. The following command displays the value of the variable Status in that instance of the subprogram:

```
Put ("_3.Status");
```

Note that the following command raises an error because it attempts to display a value without referencing a particular stack frame:

```
Put (".Initialize.Status");
```

***Example 2.*** Assume that you want to display the source of the subprogram that is called in frame 5 of the current task. You can use the following command to do this:

```
Source ("_5");
```

## Task-Identifier Prefix %

When the percent character (%) precedes a name or number, that name or number refers to a task. All tasks have numbers that are assigned by the Environment. In addition, task names can be assigned using the !Commands.Debug.Set_Task-_Name or !Tools.Debug_Tools.Set_Task_Name procedure. You can list all tasks with their names and numbers using the !Commands.Debug.Task_Display procedure.

Name components following a task identifier denote some object declared in the specified task. A stack frame number can be included to reference a frame in the call stack for the specified task (see "Stack Frame Prefix _," above). No name components can precede a task identifier.

***Example 1.*** Assume that you want to display the frames in the stack for a task named Debug_Shell. This task also has the number 7C84BE. You can display the stack for this task using either of the following commands:

```
Stack ("%Debug_Shell");
Stack ("%7C84BE");
```

***Example 2.*** Assume that you want to display the value of a variable called Status relative to frame _3 of a task called Debug_Shell. You can do this using the following command:

```
Put ("%Debug_Shell._3.Status");
```

## RESTRICTED NAMING EXPRESSIONS

Some Environment commands have parameters that accept only a restricted subset of naming possibilities. Such parameters typically are used to match names from some kind of list (which involves matching strings) rather than to resolve names to objects (which involves searching the library system).

Examples:

- The Source parameter in the Links.Display command accepts a restricted naming expression to identify the subset of a world's links to be displayed.
- The For_Prefix parameter in the Archive.Copy command accepts a restricted naming expression to identify portions of object names to be replaced.

Restricted naming expressions permit three of the four naming wildcards plus set notation, as shown in Table 17. Note that in a restricted naming expression, the character ? does not distinguish between world and nonworld components.

***Table 17 Restricted Naming Expressions***

| Character | Description |
|---|---|
| # | Matches a single character other than a period: `T####` matches Tools. |
| @ | Matches zero or more characters not containing a period: `!U@.@.Tools` matches !Users.Anderson.Tools. |
| ? | Matches zero or more name components of any kind: `!Users.Anderson?` matches !Users.Anderson and everything in it. |
| [...] | Encloses a set of names: `[!Users.Anderson?,!Users.Miyata?]` matches everything in the home worlds for Anderson and Miyata. |
| ~*name* | Excludes a name from a set: `[@,~Tools]` matches everything except Tools. |

## SPECIFYING OTHER COMMAND INPUTS: OVERVIEW

This section describes parameter values that specify particular modes of command operation. Such values include:

- Ada identifiers
- Special values
- Wildcards used in regular expression searching
- String values accepted by the Options parameter found in many commands
- String values accepted by the Response parameter found in many commands

## ADA IDENTIFIERS

Because Environment commands are Ada subprograms, many of them accept Ada identifiers as parameter values. Such Ada identifiers:

- *Are* used to specify information that is embodied in Ada objects, such as type enumerations, functions, and named constants.

- Are *not* used to specify Environment objects such as files, Ada units, or libraries; instead, Environment objects must be referenced using pathnames, which are strings (see "Referencing Environment Objects: Overview," page 14).

***Example.*** In the following command, Ada identifiers serve as default values for the Scope, Goal, and Effort_Only parameters. The first two of these are type enumerations, whereas the third is a Boolean value.

```
Compilation.Make (Unit        => "<IMAGE>",
                  Scope       => Compilation.All_Parts,
                  Goal        => Compilation.Coded,
                  Limit       => "<WORLDS>",
                  Effort_Only => False,
                  Response    => "<PROFILE>");
```

As shown in this example, Ada identifiers:

- Are not enclosed in quotation marks (unlike string values, which are enclosed).
- Must be qualified if they reference Ada units. For example, the identifiers Compilation.All_Parts and Compilation.Coded, which reference type enumerations, contain a name component for Compilation, the package in which they are defined.

## SPECIAL VALUES

Special values are strings that specify particular command behavior. Special values have the form *"<value>"*, where *value* may be defined on a command-by-command, package-by-package, or systemwide basis. Special values have the same syntax as special names, which differ by resolving to Environment objects.

***Example 1.*** The following command from package Compilation contains one special name (for the Unit parameter) and two special values for the Limit and Response parameters). The "<WORLDS>" special value limits the effect of the command to the worlds that contain the specified units. The "<PROFILE>" special value is described under "Response Parameter," page 55.

```
Compilation.Delete (Unit     => "<SELECTION>",
                    Limit    => "<WORLDS>",
                    Response => "<PROFILE>");
```

***Example 2.*** In the following command from package Cmvc, a special value causes the automatic generation of the new reservation token name:

```
Cmvc.Sever (What_Object                => "<SELECTION>",
            New_Reservation_Token_Name => "<AUTO_GENERATE>");
```

## PATTERN-MATCHING CHARACTERS

By default, the search commands in packages Editor.Search and File_Utilities find literal strings. However, each of these commands also provides an option or parameter that permits *pattern matching*. When pattern matching is used, strings are found if they conform to a pattern (*regular expression*) that is defined using specially interpreted metacharacters called *pattern-matching wildcards*. Such wildcard characters are listed in Table 18.

The pattern-matching wildcards that are used in searches are different from the naming wildcards used in pathnames (see "Wildcards for Matching One or More Pathnames," page 29).

***Table 18 Wildcard Characters for Pattern Matching***

| Character | Description |
|---|---|
| ? | Matches any single character. |
| % | Matches any single character that is legal in an Ada identifier. |
| $ | Matches Ada delimiters: & ' ( ) * + , - . / : ; < = > \| When used outside brackets ([ ]), $ matches beginning and end of line as well. |
| \ | Quotes the next wildcard character, causing it to have a literal (not a wildcard) interpretation. \ must immediately precede the wildcard it quotes. |
| { | Matches the beginning of a line, when used at the beginning of the pattern; otherwise, { has a literal meaning. |
| } | Matches the end of a line, when used at the end of the pattern; otherwise, } has a literal meaning. |
| [ ] | Defines a set of characters, of which any one can be matched. The set can be a list (for example, [ABCDE]) or a range (for example, [A-Z]). |
| ^ | Excludes the next character or set of characters; ^a matches any character other than a, and ^[abc] or [^abc] matches any character other than a, b, or c. |
| * | Matches zero or more occurrences of the previous character(s). |

## OPTIONS PARAMETER

Many Environment commands have an Options parameter through which you can restrict or specify one or more aspects of command operation. The Options parameter for a given command accepts one or more *option specifications*, as defined by that command. By accepting multiple option specifications, a command's Options parameter reduces the need for multiple individual parameters.

Individual option specifications are strings that assign *values* to *options*. The following subsections describe:

- The general format for option specifications
- Shorthand formats for option specifications with certain kinds of values
- The syntax for entering multiple option specifications

### General Option Specifications

Regardless of the command for which they are defined, individual option specifications are strings with the general form:

```
option delimiter value
```

where:

- *option* is a string defined by the command to represent some aspect of command behavior.

- *delimiter* is a string that separates an option from a value; a delimiter can be the => symbol, the colon equals (:=) symbol, or the equals (=) symbol.
- *value* is a string that specifies how the particular aspect of command behavior is to be carried out. Values can be:
  - Strings representing Boolean values (namely, True or False)
  - Predefined literal strings
  - User-specified strings (such as pathnames, time expressions, and so on)

  If a value contains a comma, a semicolon, or any of the delimiter characters, that value must be enclosed in parentheses.

***Example 1.*** In the Options parameter of the !Commands.Archive.Save procedure, the After option can be specified in any of the following three ways:

```
Options => "After=>12/25/90"
Options => "After:=12/25/90"
Options => "After=12/25/90"
```

***Example 2.*** In the Options parameter of the !Commands.Archive.Save procedure, the Label option must be specified with parentheses enclosing the value, which contains commas:

```
Options => "Label=(FRIDAY, JANUARY 25, 1991)"
```

## Options with Boolean Values

Options that accept Boolean values can be specified using the shorthand below:

- To specify the value True, you can omit the delimiter and value after the option.
- To specify the value False, you can precede the option with the tilde and omit the delimiter and value.

***Example 1.*** In the Options parameter of the !Commands.Archive.Restore procedure, the Replace option can be assigned the value True in any of the following ways:

```
Options => "Replace"
Options => "Replace=>True"
Options => "Replace:=True"
Options => "Replace=True"
```

***Example 2.*** The same option can be assigned the value False in any of the following ways:

```
Options => "~Replace"
Options => "Replace=>False"
Options => "Replace:=False"
Options => "Replace=False"
```

## Options with Literal Values

Options that accept enumerated literal values can be specified using just the value, omitting the option and the delimiter.

***Example.*** In the Options parameter of the !Commands.Tape.Write procedure, the Format option can be assigned the literal value Fixed_Length in any of the following ways:

```
Options => "Fixed_Length"
Options => "Format=Fixed_Length"
```

## Multiple Options

Multiple option specifications are separated as follows:

- When the general format is used, option specifications must be separated by either commas (,) or semicolons (;).
- When Boolean options are specified without the delimiter and value, the options can be separated by spaces, omitting the comma or semicolon separators.
- When two or more options are assigned the same value, the options can be separated by vertical bars ( | ), with the delimiter and value following the last option.

***Example 1.*** In the Options parameter of the Archive.Restore procedure, the After and Format options are both specified as follows:

```
Options => "After=12/25/90,Format=R1000"
```

***Example 2.*** In the Options parameter of the Archive.Restore procedure, the Replace and Promote parameters can both be set to True as follows:

```
Options => "Replace Promote"
```

***Example 3.*** In the Options parameter of the Archive.Restore procedure, two access-control options are assigned the same value as follows:

```
Options => "Object_Acl|Default_Acl=>(John=>Rcod)"
```

# RESPONSE PARAMETER

Many Environment commands have a Response parameter that specifies a set of *response characteristics.* A command's response characteristics determine the following aspects of its behavior during execution:

- How to respond to errors
- Where to direct the logs that summarize command behavior
- How to filter and format the messages that are recorded in these logs
- Where to find the correct activity to use (for commands that compile and/or load programs developed in subsystems)
- Where to find the remote-passwords file and remote-session file to use (for commands that must access a remote system on the same network)

In most commands that have a Response parameter, you can specify a set of response characteristics by specifying a string value:

- You can use a special value such as "<PROFILE>", "<WARN>", and "<ERRORS>" to cause the command to take its response characteristics from one of a number of predefined *response profiles.* See "Specifying a Predefined Response Profile," page 56.
- You can combine a special value with one or more option specifications to tailor the response characteristics from a predefined profile. Each option specification overrides an individual response characteristic in the profile for the duration of the command. See "Specifying Individual Response Characteristics," page 59.

In some commands, such as those in package Log, the Response parameter accepts values of type Profile.Response_Profile instead of string values. For more information, see package Profile in the Session and Job Management (SJM) book.

## Specifying a Predefined Response Profile

A response profile is a prepackaged set of response characteristics that are stored by the Environment as an aggregate of component values. These component values are set initially by the Environment and can be changed by users through several interfaces. These interfaces are discussed in detail in the Session and Job Management (SJM) book.

You can request that a command use the response characteristics provided by a particular profile by specifying an appropriate special value to the command's Response parameter. The following subsections discuss the special values for:

- Basic profiles for systemwide, sessionwide, or job-specific use
- Special-purpose profiles for filtering log messages and controlling error response

### Basic Response Profiles

The Environment provides three basic response profiles from which a command can take its response characteristics:

- The *job response profile*, which is available to any command executing in the same job. (A job is one or more procedures executed by promoting a single command window.)
- The *session response profile*, which is available to any command executing in a given session.
- The *system default profile*, which is provided by the Environment for general use.

Through a command's Response parameter, you can specify one of these basic profiles using the special values listed in Table 19.

***Table 19 Special Values for Specifying Basic Profiles***

| Special Value | Description |
|---|---|
| "<PROFILE>" | Causes the command to obtain its response characteristics from the job response profile. |
| "<SESSION>" | Causes the command to obtain its response characteristics from the session response profile, ignoring the job response profile. |
| "<DEFAULT>" | Causes the command to obtain its response characteristics from the system default profile, ignoring the job and session response profiles. |

The response characteristics in each of the basic profiles are stored as an aggregate of component values. The values of each profile are described in the following paragraphs.

***System Default Profile.*** The component values of the system default profile are fixed by the Environment. The values in the system default profile cause a command to:

- Persevere when an error is encountered—that is, continue processing as long as possible and raise no exception

- Log all messages except debug (diagnostic) messages
- Prefix each log message with the date, time, and symbol representing the message type
- Set the width of the log file to be 77 characters
- Send its output to Current_Output

Note that the system default profile provides no values that specify an activity, a remote-passwords file, or a remote-session file.

***Session Response Profile.*** The component values of a session response profile are set using a group of session switches with prefix Profile. The initial values of these session switches are copied from the system default profile, and they embody the response characteristics listed above.

You can change the component values of the session response profile by editing the appropriate switches. Through these switches, you can specify how you want the majority of your commands to execute when you are logged into that session.

Note that the system default profile does not provide an initial value that specifies an activity. However, a default activity is normally set by other mechanisms—for example, by an Activity.Set_Default statement in the Environment's default login procedure—so that you can execute commands from products or applications that are delivered or developed in subsystems. Setting the default activity automatically sets the value of the Profile.Activity_File switch, which, in turn, defines the activity component of the session response profile. Thus, by default, the activity that is used for executing commands that reside in subsystems is also used by commands that compile and/or load programs in subsystems. (See also "Specifying a Nondefault Activity," below.)

A session response profile persists across logins, lasting until you change any of the relevant session-switch values.

***Job Response Profile.*** The component values in a job response profile are set when the corresponding job is initiated. The initial values in the job response profile are copied from the session response profile, and they embody the response characteristics specified by the switch values.

You can change the component values of the job response profile using commands from package Profile entered through the same command window. Changing a job response profile with commands from package Profile:

- Allows you to override one or more response characteristics from the session response profile for the duration of the current job
- Is recommended only if you are executing a number of commands in the job and you want the changed response characteristics to apply to all of them

If you are executing only a single command, it is easier to override individual response characteristics by using option specifications in the Response parameter; see "Specifying Individual Response Characteristics," page 59.

***Summary.*** The Response parameter of most commands has the default special value "<PROFILE>". When you use this special value, the command obtains its response characteristics from the job response profile. This means that, by default, the command behaves as specified by the session switches for the current session, unless you have used commands from package Profile in the same command window to specify different response characteristics.

The special value "<SESSION>" is useful in the following situation:

- You have used commands from package Profile to set a job response profile;

- You want most commands in the job to use values from the job response profile (specify the special value "<PROFILE>" for the Response parameters of these commands); and
- You want some commands to use values from the session response profile (specify the special value "<SESSION>" for the Response parameters of these commands).

If you specify the "<DEFAULT>" special name, be aware that the system default profile does not specify a default activity. When executing commands that compile and/or load programs in subsystems, you must either specify an activity explicitly or else specify a profile that supplies an activity for the command to use.

## Special-Purpose Response Profiles

The Environment provides a number of special-purpose response profiles whose component values are preset to:

- Specify the kinds of log messages a command can send to the log file
- Specify how a command will respond to errors

You can use the special values in Table 20 to cause a command to take its response characteristics from one of these special-purpose profiles. These response characteristics are explained in greater detail following the table.

Note that apart from message filtering and error reaction, these profiles provide the same response characteristics as the current job response profile. That is, you can think of the special values in Table 20 as being variants of the "<PROFILE>" special value.

***Table 20 Special Values for Special-Purpose Response Profiles***

| Special Value | Response Characteristics |
|---|---|
| `"<ERRORS>"` | Logs only negative, error, and exception messages (++*, ***, %%%); perseveres at errors, without raising an exception; otherwise, same as job response profile. |
| `"<IGNORE>"` | Logs no messages; perseveres at errors, without raising an exception; otherwise, same as job response profile. |
| `"<NIL>"` | Logs no messages; quits at the first error, without raising an exception; uses no activity, remote-passwords, or remote-sessions file; ignores job and session response profiles. |
| `"<PROGRESS>"` | Logs only positive, negative, error, and exception messages (+++, ++*, ***, %%%); perseveres at errors, without raising an exception; otherwise, same as job response profile. |
| `"<QUIET>"` | Same as "<IGNORE>". |
| `"<RAISE_EXCEPTION>"` | Raises an exception at the first error and quits immediately; otherwise, same as job response profile. (For package Archive commands, you should use the string `"Raise_Error, <PROFILE>"` instead to permit graceful termination after exception is raised.) |
| `"<VERBOSE>"` | Logs all messages except debug messages (???); otherwise, same as job response profile. |
| `"<WARN>"` | Logs only negative, warning, error, and exception messages (++*, !!!, ***, %%%); perseveres at errors, without raising an exception; otherwise, same as job response profile. |

***Filtering Messages.*** The response characteristics in the special-purpose profiles control the kind and amount of information a command reports about its execution. When an Environment command executes, it may potentially produce a number of different kinds of messages, where each kind of message conveys a characteristic kind of information and is preceded by a characteristic set of symbols:

- Auxiliary messages (:::) and notes messages (---) provide general information about the command.
- Debug messages (???) provide diagnostic messages.
- Positive messages (+++) indicate progress.
- Negative, warning, and error messages (++*, !!!, and ***) indicate various degrees of problems encountered by the command; position messages (>>>) indicate where the problem occurred.
- Exception messages (%%%) indicate where an exception has been raised.
- User-defined messages (@@@, $$$, ###) are generated from user-defined tools.

The special values in Table 20 cause a command to either:

- Filter out certain kinds of messages, so that fewer messages are directed to the log file (for example, "<ERRORS>", "<WARN>", and "<PROGRESS>"). Filtering messages can make the log file easier to read.
- Eliminate all messages (for example, "<IGNORE>", "<QUIET>", and "<NIL>").
- Generate a more complete set of messages than if the response characteristics of the job response profile were used (for example, "<VERBOSE>"). This is useful when your session switches are normally set to filter messages, but you want a more detailed log for a particular command.

***Reacting to Errors.*** Apart from filtering messages, the response characteristics in each special-purpose profile specify how a command should react to the errors it encounters. When a command encounters an error, it may:

- Stop immediately without raising an exception
- Stop immediately and raise an exception
- Continue processing as long as possible, without raising an exception (persevere)
- Continue processing as long as possible and raise an exception

All but three of the special values in Table 20 cause a command to persevere when an error is encountered.

## Specifying Individual Response Characteristics

In addition to accepting special values, the Response parameter serves as a kind of specialized Options parameter through which you can set individual response characteristics using one or more option specifications. By combining option specifications with special values, you can tailor the response characteristics of a predefined profile for a particular command.

The following subsection describes the syntax for combining option specifications with special values. Subsequent subsections describe how to:

- Set error reaction
- Filter log messages

- Change the format of message prefixes
- Adjust output width
- Redirect command output
- Specify a nondefault activity for commands that compile and/or load programs developed in subsystems
- Specify a nondefault remote-passwords file or a remote-sessions file for commands that access a remote system on the same network.

## Syntax for Combining Options and Special Values

Like the option specifications accepted by the Options parameter, those accepted by the Response parameter:

- Have the general form *option=>value*
- Can be abbreviated in different ways, depending on the type of value the option accepts (see "Options Parameter," above)
- Must be separated by commas from other option specifications or special values

Although it is possible to specify an entire set of response characteristics using options, you will normally want to change only a few characteristics, leaving the rest to be taken from a predefined profile such as the job or session response profile. To do this, you enter one or more option specifications, followed by one of the special values listed in Tables 19 and 20. The option specifications and the special value must be separated by a comma.

If you do not include a special value to supply the remaining response characteristics, the Response parameter may cause unwanted (or at least unexpected) command behavior.

***Example.*** Assume that you want the output for a particular command to be 255 characters wide to allow long messages to fit onto single lines. Assume further that this is the only change you want to make to the way the command normally executes. The following Response parameter accomplishes this (note that individual option specifications and the special value are separated by commas):

```
Response => "Width=>255, <PROFILE>"
```

In contrast, the following Response parameter provides the command with only one response characteristic—namely, that of output width. Because no other characteristics are specified, the command generates no messages, has no instructions for responding to errors, and has no activity to use (should it need one):

```
Response => "Width=>255"
```

## Setting Error Reaction

You can specify a command's reaction to errors through the Reaction option. This option accepts one of four literal values that vary along each of two dimensions:

- Whether to quit immediately or to continue processing as long as possible when an error is encountered
- Whether or not to raise an exception after processing is complete

The four literal values are:

`Quit` Stops immediately at the first error; does not raise an exception

| | |
|---|---|
| `Propagate` | Stops immediately at the first error; raises an exception |
| `Persevere` | Continues processing when an error is encountered; does not raise an exception |
| `Raise_Error` | Continues processing when an error is encountered; raises an exception after all processing is complete |

***Example.*** Assume you want a command to continue processing when an error is encountered so that it can terminate gracefully, at which point you want it to raise an exception. Either of the following Response parameters accomplishes this (the second way of specifying the option is an abbreviation of the first—see "Options with Literal Values," above):

```
Response => "Reaction=>Raise_Error, <PROFILE>"
Response => "Raise_Error, <PROFILE>"
```

In contrast, the following Response parameter causes a command to stop processing immediately and raise an exception at the first error:

```
Response => "Propagate, <PROFILE>"
```

Note that the special value "<RAISE_EXCEPTION>" listed in Table 20 above is equivalent to "Propagate, <PROFILE>".

## Filtering Messages

You can specify the kinds of log messages a command generates through a set of twelve Boolean options. A separate option controls each type of message (see "Special-Purpose Response Profiles," above, for a description of the different kinds of messages):

- Setting a given option to True allows messages of the corresponding type to appear in the log.
- Setting a given option to False filters out such messages.

These options are named using the symbols characteristic of each message:

```
:::   ???   ---   +++   >>>   ++*
!!!   ***   %%%   ###   @@@   $$$
```

Lists of options can be abbreviated using range notation. Range notation assumes the left-to-right order in which options are listed above, so that +++ .. *** is equivalent to +++, >>>, ++*, !!!, ***.

These Boolean options are usually specified using the abbreviated notation described in "Options with Boolean Values," above.

***Example 1.*** The following Response parameter permits auxiliary messages to appear, while suppressing warning messages:

```
Response => ":::, ~!!!, <PROFILE>"
```

***Example 2.*** You can use special values such as "<WARN>" instead of specifying certain combinations of Boolean message options. The following list shows equivalent pairs of Response-parameter values:

```
"<ERRORS>"     "++*, ***, %%%, PERSEVERE, <PROFILE>"
"<IGNORE>"     "~:::..$$$, PERSEVERE, <PROFILE>"
"<PROGRESS>"   "+++, ++*, ***, %%%, PERSEVERE, <PROFILE>"
"<WARN>"       "++*..%%%, PERSEVERE, <PROFILE>"
```

## Changing the Format of Message Prefixes

You can use the Prefix option to change the content or format of the information that appears at the beginning of output messages. Such messages can be prefixed with up to three fields of information. For example, under the system default profile, messages are prefixed with the current date, time, and symbols indicating message type, as shown:

```
89/07/23 19:19:27 ::: [Make_Controlled has finished].
```

This Prefix option accepts up to three of the literals listed below, which provide alternative formats for the date and time. The literals can be specified in any combination and in any order; multiple literals must be separated by blanks.

```
TIME        DATE        SYMBOLS
HR_MN_SC    MN_DY_YR
HR_MN       DY_MON_YR
            YR_MN_DY
```

***Example.*** The following Response parameter shows how the system default profile sets the message prefix (as in the sample message above):

```
Response => "Prefix => YR_MN_DY HR_MN_SC SYMBOLS, <PROFILE>"
```

In contrast, the following Response parameter produces shorter messages by suppressing the date and time stamp and leaving only the message symbols:

```
Response => "Prefix => SYMBOLS, <PROFILE>"
```

## Adjusting Output Width

You can change the width of the log using the Width option. The width of the log is the maximum length, in characters, of each message line. This option can be specified with a positive integer. The maximum width is 1,023 characters.

## Redirecting Command Output

The Log_File option specifies where log messages are to be directed. Only log messages are affected by this option; the file or window to which other job output (such as a report or display) is directed is determined on a command-by-command basis.

The Log_File option accepts one of four literal values:

| | |
|---|---|
| `Use_Output` | Directs messages to Current_Output (by default, same as Standard_Output) |
| `Use_Error` | Directs messages to Current_Error (by default, same as Standard_Error) |
| `Use_Standard_Output` | Directs messages to Standard_Output (an Environment output window) |
| `Use_Standard_Error` | Directs messages to Standard_Error (the Environment message window) |

Note that if a command sends log messages to Current_Output or Current_Error, you can cause these messages to be further redirected (for example, to a file) via commands like Log.Set_Output and Io.Set_Output. Such commands do not redirect log messages that are sent to Standard_Output or Standard_Error via the Log_File option. For more information about directing log messages and job output, see the Session and Job Management (SJM) book.

***Example.*** The following Response parameter specifies Use_Error, which causes a command to direct messages to the message window at the top of the screen:

```
Response => "Use_Error, <PROFILE>"
```

### Specifying a Nondefault Activity

You can use the Activity option to specify a nondefault activity for the command to consult during execution. An activity is required during execution by any command that attempts to compile and/or load a program that resides in subsystems. Such a command uses an activity to identify the load view that corresponds to each spec view containing units to be compiled and/or loaded.

Commands may need to look up load views in an activity when they:

- Compile programs that reside in spec and load views (for example, Compilation.Make with the Load_Views scope)
- Create loaded main programs that reference units in spec and load views (for example, Compilation.Load)
- Compile programs to be executed during the same job, where the programs to be compiled reference units in spec and load views (for example, Program.Run, Program.Run_Job, and Program.Create_Job)

Specifying the Activity option is useful when you want a particular command to use a different activity than the one that is set in your session switches.

See package Activity in the Project Management (PM) book for information on activities.

### Specifying Nondefault Remote-Information Files

You can use the Remote_Passwords and Remote_Sessions options to specify a nondefault remote-passwords file and a nondefault remote-sessions file, respectively. These files provide password and session information for commands to use when accessing a remote system on the same network (see package Remote_Passwords in the Session and Job Management (SJM) book).

Specifying one or both of these options is useful when you want a particular command to use a different remote-passwords and/or sessions file than the ones that are set in your session switches.

## Summary of Response-Parameter Options

***Caution:*** *Unspecified options are set to nil; to be safe, use options along with special names. Examples:*

```
Response => ("Width=255, <PROFILE>")
Response => ("Use_Error, <PROFILE>")
Response => ("Symbols, <PROFILE>")
```

- `Width = `*`n`*

  Sets the width of the message output to the specified number of characters.

- `Reaction = `*`literal`*

  Specifies how the command responds to errors:

  | | |
  |---|---|
  | `Quit` | Stops immediately at the first error; does not raise an exception |
  | `Propagate` | Stops immediately at the first error; raises an exception |

| | |
|---|---|
| `Persevere` | Continues processing when an error is encountered; does not raise an exception |
| `Raise_Error` | Continues processing when an error is encountered; raises an exception after all processing is complete |

- *Message-symbols*

  Boolean options that control whether the corresponding types of messages appear in the log. There are twelve such options, one for each type of message:

```
:::   ???   ---   +++   >>>   ++*
!!!   ***   %%%   ###   @@@   $$$
```

- `Prefix =` *literals*

  Specifies up to three fields of information to be prefixed to each message:

```
TIME        DATE        SYMBOLS
HR_MN_SC    MN_DY_YR
HR_MN       DY_MON_YR
            YR_MN_DY
```

- `Log_File =` *literal*

  Specifies where log messages are to be directed:

| | |
|---|---|
| `Use_Output` | Directs messages to Current_Output (by default, same as Standard_Output) |
| `Use_Error` | Directs messages to Current_Error (by default, same as Standard_Error) |
| `Use_Standard_Output` | Directs messages to Standard_Output (an Environment output window) |
| `Use_Standard_Error` | Directs messages to Standard_Error (the Environment message window) |

- `Activity =` *activity name*

  Specifies the name of the activity to use during execution.

- `Remote_Passwords =` *filename*

  Specifies the remote-passwords file to use when accessing remote machines.

- `Remote_Sessions =` *filename*

  Specifies the remote-sessions file to use when accessing remote machines.

## MAP OF THE RATIONAL ENVIRONMENT LIBRARY SYSTEM

The following abbreviations are used to refer to Rational's manuals:

Access: *Rational Access User's Guide*
BOP: *Rational Environment Basic Operations*
DEB: Debugging (Vol. 3 of the ERM)
DIO: Device Input/Output (Vol. 7 of the ERM)
EI: Editing Images (in Vol. 2 of the ERM)
ERM: *Rational Environment Reference Manual*
EST: Editing Specific Types (in Vol. 2 of the ERM)
FTP: File Transfer Protocol (in Vol. 1 of the *Rational Networking—TCP/IP Reference Manual*)
LM: Library Management (Vol. 5 of the ERM)
Mail: *Rational Network Mail User's Guide*
PM: Project Management (Vol. 11 of the ERM)
PT: Programming Tools (Vol. 9 of the ERM)
RN: Release Note (online in !Machine.Release.Release_Notes)
RPC: Remote Procedure Call (in Vol. 2 of the *Rational Networking—TCP/IP Reference Manual*)
RS: This book (Reference Summary, Vol. 1 of the ERM)
RWI: *Rational Windows Interface User's Manual*
RXI: *Rational X Interface User's Guide*
SJM: Session and Job Management (Vol. 4 of the ERM)
SMG: *System Manager's Guide*
SMU: System Management Utilities (Vol. 10 of the ERM)
Spell: *Rational Spelling Checker*
ST: String Tools (Vol. 8 of the ERM)
TEL: Telnet (in Vol. 1 of the *Rational Networking—TCP/IP Reference Manual*)
TIO: Text Input/Output (Vol. 6 of the ERM)
TRL: Transport Layer (in Vol. 2 of the *Rational Networking—TCP/IP Reference Manual*)

A blank book abbreviation indicates that documentation is not meaningful for the item. The abbreviation n/a indicates that documentation currently is not available.

*Note:* This section includes only those objects delivered with the standard Environment. Objects delivered with any of Rational's other products are described in the documentation for those products.





| Name | Kind | Book Abbreviation for Documentation |
|---|---|---|
| ! : Library (World); | | |
| Commands | : Library (World); | see below |
| Compiler_Interface | : Library (World); | n/a |
| Copyright_Rational | : File (Text); | |
| Implementation | : Library (World); | n/a |
| Io | : Library (World); | see below |
| Local | : Library (World); | |
| Lrm | : Library (World); | see below |
| Machine | : Library (World); | see below |
| Model | : Library (World); | see below |
| Tools | : Library (World); | see below |
| Training | : Library (World); | |
| Users | : Library (World); | |
| !Commands : Library (World); | | |
| Abbreviations | : Library (World); | RS |
| Access_List | : Ada (Pack_Spec); | LM |
| Action_Utilities | : Ada (Pack_Spec); | n/a |
| Activity | : Ada (Pack_Spec); | PM |
| Ada | : Ada (Pack_Spec); | EST |
| Archive | : Ada (Pack_Spec); | LM |
| Cmvc | : Ada (Pack_Spec); | PM |
| Cmvc_Access_Control | : Ada (Pack_Spec); | PM |
| Cmvc_Hierarchy | : Ada (Pack_Spec); | PM |
| Cmvc_Maintenance | : Ada (Pack_Spec); | PM |
| Command | : Ada (Pack_Spec); | EST |
| Common | : Ada (Pack_Spec); | EST |
| Compilation | : Ada (Pack_Spec); | LM |
| Convert_Mailboxes | : Ada (Proc_Spec); | n/a |
| Daemon | : Ada (Pack_Spec); | SMU |
| Debug | : Ada (Pack_Spec); | DEB |
| Debug_Maintenance | : Ada (Pack_Spec); | DEB |
| Dependents | : Ada (Pack_Spec); | n/a |
| Diana_Tree | : Ada (Pack_Spec); | n/a |
| Disk_Space | : Ada (Pack_Spec); | SMG |
| Do_Step | : Ada (Load_Proc); | Install Note |
| Editor | : Ada (Pack_Spec); | EI |
| File_Utilities | : Ada (Pack_Spec); | LM |
| Ftp | : Ada (Pack_Spec); | FTP |
| Gateway | : Ada (Pack_Spec); | n/a |
| Gateway_Class | : Ada (Pack_Spec); | n/a |
| Job | : Ada (Pack_Spec); | SJM |
| Library | : Ada (Pack_Spec); | LM |
| Links | : Ada (Pack_Spec); | LM |
| Log | : Ada (Pack_Spec); | SJM |
| Mail | : Ada (Pack_Spec); | Mail |

```
Menu_Operations                          : Ada (Pack_Spec);        n/a
Message                                  : Ada (Pack_Spec);        SMU
Network                                  : Ada (Pack_Spec);        TRL
Operator                                 : Ada (Pack_Spec);        SMU, SJM
Program                                  : Ada (Pack_Spec);        SJM
Queue                                    : Ada (Pack_Spec);        SMU, SJM
Remote                                   : Ada (Pack_Spec);        SJM
Remote_Passwords                         : Ada (Pack_Spec);        SJM
Scheduler                                : Ada (Pack_Spec);        SMU, SMG
Search_List                              : Ada (Pack_Spec);        SJM
Sims'Spec_View.Units                     : Library (Directory);
Speller                                  : Ada (Pack_Spec);        Spell
Switches                                 : Ada (Pack_Spec);        LM
System_Backup                            : Ada (Pack_Spec);        SMU
System_Maintenance'Spec_View.Units       : Library (Directory);
  .Accept_Tokens                         : Ada (Proc_Spec);        SMG
  .Allow_Token_Conversion                : Ada (Proc_Spec);        n/a
  .Analyze_Disks_For_Backup              : Ada (Proc_Spec);        D_12_2_4 RN
  .Change_Boot_Microcode                 : Ada (Proc_Spec);        n/a
  .Check_And_Correct_Users               : Ada (Proc_Spec);        n/a
  .Check_Universe_Acls                   : Ada (Proc_Spec);        SMG
  .Convert_Tokens                        : Ada (Proc_Spec);        SMG
  .Destroy_Library                       : Ada (Proc_Spec);        D_12_2_4 RN
  .Destroy_User                          : Ada (Proc_Spec);        n/a
  .Display_Switches                      : Ada (Proc_Spec);        n/a
  .Donate_Tokens                         : Ada (Proc_Spec);        n/a
  .Find_Block                            : Ada (Proc_Spec);        n/a
  .Find_Null_Acls                        : Ada (Proc_Spec);        SMG
  .Get_Machine_Id                        : Ada (Func_Spec);        n/a
  .Get_Site                              : Ada (Func_Spec);        n/a
  .Image_Tree_Consistency                : Ada (Proc_Spec);        n/a
  .Last_Gasp_Destroy                     : Ada (Proc_Spec);        D_12_6_5 RN
  .Low_Level_Destroy                     : Ada (Proc_Spec);        n/a
  .Monitor_Performance                   : Ada (Proc_Spec);        D_12_2_4 RN
  .Refresh_Terminal_Information          : Ada (Proc_Spec);        D_12_1_1 RN
  .Release_Tape_User                     : Ada (Proc_Spec);        D_12_6_5 RN
  .Repair_Cg_Attrs                       : Ada (Proc_Spec);        D_12_2_4 RN
  .Repair_Directory                      : Ada (Proc_Spec);        n/a
  .Save_Performance_Data                 : Ada (Proc_Spec);        n/a
  .Setup_Machine                         : Ada (Proc_Spec);        D_12_1_1 RN
  .Set_Site                              : Ada (Proc_Spec);        D_12_1_1 RN
  .Set_Universe_Acls                     : Ada (Proc_Spec);        SMG
  .Show_Compiler_Version                 : Ada (Proc_Spec);        n/a
  .Show_Configuration                    : Ada (Proc_Spec);        n/a
  .Show_Directory_Information            : Ada (Proc_Spec);        n/a
  .Show_Elaborated_Configuration         : Ada (Proc_Spec);        D_12_7_3 RN
  .Show_Environment_Vpids                : Ada (Proc_Spec);        n/a
  .Show_Groups                           : Ada (Proc_Spec);        SMG
  .Show_Identity                         : Ada (Proc_Spec);        n/a
  .Show_Image_Options                    : Ada (Proc_Spec);        n/a
```





```
  .Show_Iop_Kernel                       : Ada (Proc_Spec);        n/a
  .Show_Jobs                             : Ada (Proc_Spec);        n/a
  .Show_Job_Names                        : Ada (Proc_Spec);        n/a
  .Show_Load_Proc_Unit_Names             : Ada (Proc_Spec);        D_12_6_5 RN
  .Show_Locks                            : Ada (Proc_Spec);        n/a
  .Show_Log_Throttle_State               : Ada (Proc_Spec);        n/a
  .Show_Machine_Id                       : Ada (Proc_Spec);        n/a
  .Show_Manager_State                    : Ada (Proc_Spec);        n/a
  .Show_Memory_Hogs                      : Ada (Proc_Spec);        n/a
  .Show_Mts_Profiles                     : Ada (Proc_Spec);        n/a
  .Show_Mts_Stats                        : Ada (Proc_Spec);        n/a
  .Show_R1000_Configurations             : Ada (Proc_Spec);        n/a
  .Show_Session_Of_Job                   : Ada (Proc_Spec);        D_12_7_3 RN
  .Show_Site                             : Ada (Proc_Spec);        n/a
  .Show_Snapshot_Times                   : Ada (Proc_Spec);        n/a
  .Show_Stats                            : Ada (Proc_Spec);        n/a
  .Show_Tape_Users                       : Ada (Proc_Spec);        D_12_6_5 RN
  .Show_Tasks                            : Ada (Proc_Spec);        n/a
  .Show_Tokens                           : Ada (Proc_Spec);        n/a
  .Smooth_Snapshots                      : Ada (Proc_Spec);        n/a
  .Token_Information                     : Ada (Proc_Spec);        n/a
  .Training_Authorization                : Ada (Proc_Spec);        n/a
  .Uncode                                : Ada (Proc_Spec);        n/a
  .Uninstall                             : Ada (Proc_Spec);        n/a
  .Verify_Backup                         : Ada (Proc_Spec);        D_12_1_1 RN
Tape                                     : Ada (Pack_Spec);        SMU
Telnet                                   : Ada (Pack_Spec);        TEL
Terminal                                 : Ada (Pack_Spec);        SMU, SMG
Text                                     : Ada (Pack_Spec);        EST
Transfer                                 : Aad (Pack_Spec);        n/a
Transport_Route                          : Ada (Pack_Spec);        TRL
What                                     : Ada (Pack_Spec);        SJM
Work_Order                               : Ada (Pack_Spec);        PM

!Io : Library (World);
Device_Independent_Io                    : Ada (Pack_Spec);        n/a
Direct_Io                                : Ada (Gen_Pack);         DIO
Io                                       : Ada (Pack_Spec);        TIO
Io_Exceptions                            : Ada (Pack_Spec);        DIO, TIO
Object_Set                               : Ada (Pack_Spec);        n/a
Pipe                                     : Ada (Pack_Spec);        n/a
Polymorphic_Io                           : Ada (Pack_Spec);        n/a
Polymorphic_Sequential_Io                : Ada (Pack_Spec);        DIO
Sequential_Io                            : Ada (Gen_Pack);         DIO
Tape_Specific                            : Ada (Pack_Spec);        n/a
Terminal_Specific                        : Ada (Pack_Spec);        n/a
Text_Io                                  : Ada (Pack_Spec);        TIO
Window_Io                                : Ada (Pack_Spec);        DIO
```

```
!Lrm : Library (World):
  Calendar                          : Ada (Pack_Spec);        PT
  System                            : Ada (Pack_Spec);        PT
  Unchecked_Conversion              : Ada (Gen_Func);         PT
  Unchecked_Deallocation            : Ada (Gen_Proc);         PT

!Machine : Library (World):
  Accounting                        : Library (World);
    .Enabled                        : File (Text);
  Archive_Mappings                  : File (Text);
  Attribute_Space_Maps_State        : File;
  Cg_Data                           : Library (World);
  Compiler_Switches_File            : File (Switch);
  Devices                           : Library (World);
  Dfs                               : Library (Subsystem);
    .Analyze_Crashdump              : Ada (Proc_Spec);        D_12_2_4 RN
    .Analyze_Tombstone              : Ada (Proc_Spec);        D_12_2_4 RN
    .Basetype                       : Ada (Pack_Spec);        n/a
    .Bfs                            : Ada (Pack_Spec);        n/a
    .Crashdump_Tools                : Library (Directory);
    .Dfs                            : Ada (Pack_Spec);        n/a
    .Fslabels                       : Ada (Pack_Spec);        n/a
    .Fstypes                        : Ada (Pack_Spec);        n/a
    .Install_Tools                  : Ada (Pack_Spec);        D_12_6_5 RN
    .Log_Previous_Crash             : Ada (Proc_Spec);        n/a
    .Mt                             : Library (Directory);
    .Shared_Tools                   : Library (Directory);
    .Tombstone_Tools                : Library (Directory);
    .Vfslog                         : Ada (Pack_Spec);        n/a
  Editor_Data                       : Library (World);        SMG
    .Daily_Message                  : File (Text);            SMG
    .Enable_Product_Keymaps         : Ada (Load_Proc);        n/a
    .Facit_Commands                 : Ada (Proc_Spec);        BOP (Facit)
    .Facit_Commands                 : Ada (Proc_Body);        BOP (Facit)
    .Facit_Keys                     : File;                   BOP (Facit)
    .Facit_Key_Names                : Ada (Pack_Spec);        BOP (Facit)
    .Facit_User_Commands            : File;                   BOP (Facit)
    .Help_Data                      : Library (Directory);
    .Keymap_Overlays                : Library (Directory);
    .Key_Cache                      : Ada (Pack_Spec);        n/a
    .Masters                        : Library (World);
    .Pc101_Commands                 : Ada (Proc_Spec);        RWI
    .Pc101_Commands                 : Ada (Proc_Body);        RWI
    .Pc101_Keys                     : File (Text);            RWI
    .Pc101_Key_Names                : Ada (Pack_Spec);        RWI
    .Pc101_User_Commands            : File (Text);            RWI
    .Pc86_Commands                  : Ada (Proc_Spec);        RWI
    .Pc86_Commands                  : Ada (Proc_Body);        RWI
    .Pc86_Keys                      : File (Text);            RWI
    .Pc86_Key_Names                 : Ada (Pack_Spec);        RWI
```





```
    .Pc86_User_Commands             : File (Text);            RWI
    .Pc91_Commands                  : Ada (Proc_Spec);        RWI
    .Pc91_Commands                  : Ada (Proc_Body);        RWI
    .Pc91_Keys                      : File (Text);            RWI
    .Pc91_Key_Names                 : Ada (Pack_Spec);        RWI
    .Pc91_User_Commands             : File (Text);            RWI
    .Pcbasic_Commands               : Ada (Proc_Spec);        RWI
    .Pcbasic_Commands               : Ada (Proc_Body);        RWI
    .Pcbasic_Keys                   : File (Text);            RWI
    .Pcbasic_Key_Names              : Ada (Pack_Spec);        RWI
    .Rational_Access_Commands       : Ada (Proc_Spec);        Access
    .Rational_Access_Commands       : Ada (Proc_Body);        Access
    .Rational_Access_Keys           : File (Text);            Access
    .Rational_Access_Key_Names      : Ada (Pack_Spec);        Access
    .Rational_Commands              : Ada (Proc_Spec);        BOP (Rational)
    .Rational_Commands              : Ada (Proc_Body);        BOP (Rational)
    .Rational_Keys                  : File;                   BOP (Rational)
    .Rational_Key_Names             : Ada (Pack_Spec);        BOP (Rational)
    .Rational_User_Commands         : File;                   BOP (Rational)
    .Session_Switch_Help            : File;
    .Terminal_Recognition           : File;                   SMG
    .Terminal_Types                 : File;                   SMG
    .Visible_Key_Names              : Ada (Pack_Spec);        n/a
    .Vt100_Commands                 : Ada (Proc_Spec);        n/a
    .Vt100_Commands                 : Ada (Proc_Body);        n/a
    .Vt100_Keys                     : File;                   n/a
    .Vt100_Key_Names                : Ada (Pack_Spec);        n/a
    .Vt100_User_Commands            : File;                   n/a
    .Xnews4_Commands                : Ada (Proc_Spec);        RXI
    .Xnews4_Commands                : Ada (Proc_Body);        RXI
    .Xnews4_Keys                    : File (Text);            RXI
    .Xnews4_Key_Names               : Ada (Pack_Spec);        RXI
    .Xnews4_User_Commands           : File (Text);            RXI
    .Xr6us_Commands                 : Ada (Proc_Spec);        RXI
    .Xr6us_Commands                 : Ada (Proc_Body);        RXI
    .Xr6us_Keys                     : File (Text);            RXI
    .Xr6us_Key_Names                : Ada (Pack_Spec);        RXI
    .Xr6us_User_Commands            : File (Text);            RXI
  Error_Logs                        : Library (World);
  Gateway_Classes                   : Library (Directory);
  Groups                            : Library (World);        SMG
  Initialization                    : Library (World);        SMG
    .Local                          : Library (World);        SMG
      .Initialize_Site              : Ada (Proc_Spec);        SMG
      .Initialize_Site              : Ada (Proc_Body);        SMG
      .Printer_Configuration        : File (Text);            SMG
      .Terminal_Configuration       : File (Text);            SMG
    .Rational                       : Library (World);        SMG
      .Finish_Install               : Ada (Load_Proc);        n/a
      .Network                      : Ada (Load_Proc);        n/a
```

```
      .Parameters                      : Ada (Load_Proc);          n/a
      .Printers                        : Ada (Load_Proc);          SMG
      .Servers                         : Ada (Load_Proc);          SMG
      .Terminals                       : Ada (Load_Proc);          SMG
    .Site                              : Library (World);          SMG
    .Start                             : Ada (Load_Proc);          SMG
  Link_Host_Name                       : File (Text);
  Login_Policy                         : Library (World);          SMG
    .Login_Policy_Information          : File (Text);
  Machine_Name                         : File;
  Network_Public_Session               : Session;                  SJM
  Operator_Capability                  : File (Text);              SMU
  Public_Session                       : Session;                  SJM
  Queues                               : Library (World);
  Release                              : Library (World);
    .Archive                           : Library (World);
    .Current                           : Library (World);
      .Activity                        : File (Activity);
      .Commands                        : Library (World);
        .Login                         : Ada (Proc_Spec);          SJM
        .Login                         : Ada (Proc_Body);          SJM
      .Products                        : File;
    .Debuggers                         : Library (Subsystem);
    .Environment                       : Library (World);
      .Current                         : File (Text);
    .Release_Notes                     : Library (Directory);
    .Universe_Version                  : File;
    .X_Interface                       : Library (Directory);
  Search_Lists                         : Library (World);          SJM, SMG
    .Default                           : File;                     SJM, SMG
  Shutdown_Help_File                   : File (Text);
  Sims                                 : Library (World);
  Speller_Data                         : Library (World);
  Switch_Definitions                   : File (Swtch_Def);
  Tcp_Ip_Host_Id                       : File (Text);
  Tcp_Ip_Name_Server                   : File (Text);
  Temporary                            : Library (World);
  Transfer                             : Library (World);
  Transport_Name_Map                   : File (Text);
  Transport_Route                      : File;
  Transport_Services                   : File (Text);
  Users                                : Library (World);          SMU, SMG
  User_Acl_Suffix                      : File (Tex
  User_Default_Acl_Suffix              : File (Text);

!Model : Library (World):                                        LM
  Native                               : Library (World);          n/a
  R1000                                : Library (World);          n/a
  R1000_Implementation                 : Library (World);          n/a
  R1000_Portable                       : Library (World);          n/a
```





```
!Tools : Library (World):
  Access_List_Tools                    : Ada (Pack_Spec);          LM
  Ada_Object_Editor                    : Ada (Pack_Spec);          n/a
  Ada_Text                             : Ada (Pack_Spec);          n/a
  Allows_Deallocation                  : Ada (Gen_Func);           PT
  Bit_Operations                       : Ada (Pack_Spec);          n/a
  Bounded_String                       : Ada (Pack_Spec);          ST
  Ci'Spec_View.Units                   : Library (Directory);
    .Ci                                : Ada (Pack_Spec);          SMG
    .Commands                          : Library (Directory);
      .Run                             : Ada (Pack_Spec);          SMG
      .Show                            : Ada (Pack_Spec);          SMG
      .Typ                             : Ada (Proc_Spec);          SMG
  Code_Segment_Object_Editor           : Ada (Pack_Spec);          n/a
  Compatibility'Spec_View.Units        : Library (Directory);
    .Cdb_Maintenance                   : Ada (Pack_Spec);          n/a
    .Check                             : Ada (Pack_Spec);          PM
  Concurrent_Map_Generic               : Ada (Gen_Pack);           PT
  Debug_Tools                          : Ada (Pack_Spec);          DEB
  Diana_Object_Editor                  : Ada (Pack_Spec);          n/a
  Directory_Tools                      : Ada (Pack_Spec);          n/a
  Disk_Daemon                          : Ada (Pack_Spec);          SMG
  Dtia_Rpc_Mechanisms'Spec_View.Units  : Library (Directory);
    .Remote_Shell                      : Library (Directory);      n/a
    .Sun_Rpc                           : Library (Directory);      n/a
    .Target_Interface                  : Library (Directory);      n/a
    .Transfer_Utilities                : Library (Directory);      n/a
    .Utilities                         : Library (Directory);      n/a
  Hash                                 : Ada (Pack_Spec);          PT
  Library_Object_Editor                : Ada (Pack_Spec);          n/a
  Link_Tools                           : Ada (Pack_Spec);          n/a
  List_Generic                         : Ada (Gen_Pack);           PT
  Lrm                                  : Library (Directory);
    .Ada_Program                       : Ada (Pack_Spec);          n/a
    .Associations                      : Ada (Pack_Spec);          n/a
    .Compilation_Units                 : Ada (Pack_Spec);          n/a
    .Declarations                      : Ada (Pack_Spec);          n/a
    .Names_And_Expressions             : Ada (Pack_Spec);          n/a
    .Pragmas                           : Ada (Pack_Spec);          n/a
    .Representation_Clauses            : Ada (Pack_Spec);          n/a
    .Statements                        : Ada (Pack_Spec);          n/a
    .Type_Information                  : Ada (Pack_Spec);          n/a
  Map_Generic                          : Ada (Gen_Pack);           PT
  Math_Support                         : Library (Subsystem);
  Networking                           : Library (Directory);
    .Byte_Defs                         : Ada (Pack_Spec);          TRL
    .Byte_String_Io                    : Ada (Pack_Spec);          n/a
    .File_Transfer                     : Ada (Pack_Spec);          FTP
    .Ftp_Defs                          : Ada (Pack_Spec);          FTP
    .Ftp_Name_Map                      : Ada (Pack_Spec);          FTP
```

```
    .Ftp_Product                      : Ada (Pack_Spec);        FTP
    .Ftp_Profile                      : Ada (Pack_Spec);        FTP
    .Ftp_Server                       : Ada (Pack_Spec);        n/a
    .Host_Id_Io                       : Ada (Pack_Spec);        TRL
    .Interchange                      : Ada (Pack_Spec);        RPC
    .Interchange_Defs                 : Ada (Pack_Spec);        RPC
    .Network_Product                  : Ada (Pack_Spec);        TRL
    .Rpc                              : Ada (Pack_Spec);        RPC
    .Rpc_Access_Utilities             : Ada (Pack_Spec);        RPC
    .Rpc_Client                       : Ada (Pack_Spec);        RPC
    .Rpc_Product                      : Ada (Pack_Spec);        RPC
    .Rpc_Server                       : Ada (Pack_Spec);        RPC
    .Show_Trace                       : Ada (Load_Proc);
    .Tcp_Ip_Boot                      : Ada (Proc_Spec);        TRL
    .Tcp_Ip_Dump                      : Ada (Proc_Spec);        n/a
    .Telnet_Product                   : Ada (Pack_Spec);        n/a
    .Telnet_Profile                   : Ada (Pack_Spec);        TEL
    .Telnet_Tools'Spec_View.Units     : Library (Directory);
      .Telnet_Port                    : Ada (Pack_Spec);        n/a
      .Telnet_Protocol                : Ada (Pack_Spec);        n/a
      .Telnet_Tools                   : Ada (Pack_Spec);        n/a
    .Transfer_Generic                 : Ada (Gen_Pack);         FTP
    .Transport                        : Ada (Pack_Spec);        TRL
    .Transport_Defs                   : Ada (Pack_Spec);        TRL
    .Transport_Interchange            : Ada (Pack_Inst);        RPC
    .Transport_Name                   : Ada (Pack_Spec);        TRL
    .Transport_Server                 : Ada (Gen_Pack);         RPC
    .Transport_Server_Job             : Ada (Pack_Spec);        RPC
    .Transport_Stream                 : Ada (Pack_Spec);        RPC
  Object_Editor                       : Ada (Pack_Spec);        n/a
  Parameter_Parser                    : Ada (Gen_Pack);         n/a
  Profile                             : Ada (Pack_Spec);        SJM
  Program_Library'Spec_View.Units     : Library (Directory);
    .Program_Library_Maintenance      : Ada (Pack_Spec);        n/a
  Program_Library_Object_Editor       : Ada (Pack_Spec);        n/a
  Queue_Generic                       : Ada (Gen_Pack);         PT
  Random                              : Ada (Pack_Spec);        n/a
  Rpc'Spec_View.Units                 : Library (Directory);
    .Interchange                      : Ada (Pack_Spec);        n/a
    .Interchange_Defs                 : Ada (Pack_Spec);        n/a
    .Rpc                              : Ada (Pack_Spec);        n/a
    .Rpc_Client                       : Ada (Pack_Spec);        n/a
    .Rpc_Product                      : Ada (Pack_Spec);        n/a
    .Rpc_Server                       : Ada (Pack_Spec);        n/a
    .Transport_Interchange            : Ada (Pack_Inst);        n/a
    .Transport_Server                 : Ada (Gen_Pack);         n/a
    .Transport_Server_Job             : Ada (Pack_Spec);        n/a
    .Transport_Server_Proc            : Ada (Gen_Proc);         n/a
    .Transport_Stream                 : Ada (Pack_Spec);        n/a
  Rpc_Servers'Spec_View.Units         : Library (Directory);
```





```
    .Queue_Service                       : Ada (Pack_Spec);        n/a
  Script                                 : Ada (Pack_Spec);        n/a
  Set_Generic                            : Ada (Gen_Pack);         PT
  Simple_Status                          : Ada (Pack_Spec);        PT
  Stack_Generic                          : Ada (Gen_Pack);         PT
  String_Map_Generic                     : Ada (Gen_Pack);         ST
  String_Table                           : Ada (Pack_Spec);        ST
  String_Utilities                       : Ada (Pack_Spec);        ST
  System_Availability'Spec_View.Units    : Library (Directory);
    .Accounting_Information              : Ada (Pack_Spec);        n/a
    .Accounting_Report                   : Ada (Proc_Spec);        n/a
    .Data                                : Library (Directory);
    .Log_Reader                          : Ada (Pack_Spec);        n/a
    .Outage_Information                  : Ada (Pack_Spec);        n/a
    .Reports                             : Library (Directory);
      .Availability_Report               : Ada (Proc_Spec);        n/a
      .Display_Large_Error_Logs          : Ada (Proc_Spec);        n/a
      .Error_Summary                     : Ada (Proc_Spec);        n/a
      .Error_Summary_Server              : Ada (Proc_Spec);        n/a
      .Pm_Report                         : Ada (Proc_Spec);        n/a
    .Show_Error_Log                      : Ada (Proc_Spec);        n/a
    .System_Information                  : Ada (Pack_Spec);        n/a
    .System_Report                       : Ada (Pack_Spec);        n/a
    .Utilities                           : Library (Directory);
  System_Utilities                       : Ada (Pack_Spec);        SMU
  Table_Formatter                        : Ada (Gen_Pack);         ST
  Table_Sort_Generic                     : Ada (Gen_Proc);         PT
  Tape_Tools                             : Ada (Pack_Spec);        SMG
  Time_Utilities                         : Ada (Pack_Spec);        PT
  Unbounded_String                       : Ada (Gen_Pack);         ST
  Unchecked_Conversions                  : Ada (Pack_Spec);        PT
  Xref_Utility'Spec_View.Units           : Library (Directory);
    .Commands.Xref                       : Ada (Pack_Spec);        LM
```

## !COMMANDS

This tabbed section contains copies of the specifications for the Ada packages and subprograms defined in the world !Commands and in its sublibraries. With the exception of minor formatting differences, these specifications are exact copies of those online.

The specifications in this section are organized alphabetically, as you would find them listed online. Units within a package or sublibrary are grouped together under the name of their enclosing package or sublibrary.

The standard contents of world !Commands are listed below. Those objects that are *not* documented in this section are noted.

To find other locations where a package or subprogram is documented, see the Map of the Rational Environment Library System (behind the Environment Specifications tab) or see the Master Index.

```
!Commands : Library (World);
  Abbreviations                       : Library (World);  -- separate section
  Access_List                         : Ada (Pack_Spec);
  Action_Utilities                    : Ada (Pack_Spec);
  Activity                            : Ada (Pack_Spec);
  Ada                                 : Ada (Pack_Spec);
  Archive                             : Ada (Pack_Spec);
  Cmvc                                : Ada (Pack_Spec);
  Cmvc_Access_Control                 : Ada (Pack_Spec);
  Cmvc_Hierarchy                      : Ada (Pack_Spec);
  Cmvc_Maintenance                    : Ada (Pack_Spec);
  Command                             : Ada (Pack_Spec);
  Common                              : Ada (Pack_Spec);
  Compilation                         : Ada (Pack_Spec);
  Convert_Mailboxes                   : Ada (Proc_Spec);
  Daemon                              : Ada (Pack_Spec);
  Debug                               : Ada (Pack_Spec);
  Debug_Maintenance                   : Ada (Pack_Spec);
  Dependents                          : Ada (Pack_Spec);
  Diana_Tree                          : Ada (Pack_Spec);
  Disk_Space                          : Ada (Pack_Spec);
  Do_Step                             : Ada (Load_Proc);
  Editor                              : Ada (Pack_Spec);
  File_Utilities                      : Ada (Pack_Spec);
  Ftp                                 : Ada (Pack_Spec);
  Gateway                             : Ada (Pack_Spec);
  Gateway_Class                       : Ada (Pack_Spec);
  Job                                 : Ada (Pack_Spec);
  Library                             : Ada (Pack_Spec);
  Links                               : Ada (Pack_Spec);
  Log                                 : Ada (Pack_Spec);
```





```
  Mail                                : Ada (Pack_Spec);
  Menu_Operations                     : Ada (Pack_Spec);
  Message                             : Ada (Pack_Spec);
  Network                             : Ada (Pack_Spec);
  Operator                            : Ada (Pack_Spec);
  Program                             : Ada (Pack_Spec);
  Queue                               : Ada (Pack_Spec);
  Remote                              : Ada (Pack_Spec);
  Remote_Passwords                    : Ada (Pack_Spec);
  Scheduler                           : Ada (Pack_Spec);
  Search_List                         : Ada (Pack_Spec);
  Sims'Spec_View.Units                : Library (Directory); -- not documented
  Speller                             : Ada (Pack_Spec);
  Switches                            : Ada (Pack_Spec);
  System_Backup                       : Ada (Pack_Spec);
  System_Maintenance'Spec_View.Units  : Library (Directory);
    .Accept_Tokens                    : Ada (Proc_Spec);
    .Allow_Token_Conversion           : Ada (Proc_Spec);
    .Analyze_Disks_For_Backup         : Ada (Proc_Spec);
    .Change_Boot_Microcode            : Ada (Proc_Spec);
    .Check_And_Correct_Users          : Ada (Proc_Spec);
    .Check_Universe_Acls              : Ada (Proc_Spec);
    .Convert_Tokens                   : Ada (Proc_Spec);
    .Destroy_Library                  : Ada (Proc_Spec);
    .Destroy_User                     : Ada (Proc_Spec);
    .Display_Switches                 : Ada (Proc_Spec);
    .Donate_Tokens                    : Ada (Proc_Spec);
    .Find_Block                       : Ada (Proc_Spec);
    .Find_Null_Acls                   : Ada (Proc_Spec);
    .Get_Machine_Id                   : Ada (Func_Spec);
    .Get_Site                         : Ada (Func_Spec);
    .Image_Tree_Consistency           : Ada (Proc_Spec);
    .Last_Gasp_Destroy                : Ada (Proc_Spec);
    .Low_Level_Destroy                : Ada (Proc_Spec);
    .Monitor_Performance              : Ada (Proc_Spec);
    .Refresh_Terminal_Information     : Ada (Proc_Spec);
    .Release_Tape_User                : Ada (Proc_Spec);
    .Repair_Cg_Attrs                  : Ada (Proc_Spec);
    .Repair_Directory                 : Ada (Proc_Spec);
    .Save_Performance_Data            : Ada (Proc_Spec);
    .Setup_Machine                    : Ada (Proc_Spec);
    .Set_Site                         : Ada (Proc_Spec);
    .Set_Universe_Acls                : Ada (Proc_Spec);
    .Show_Compiler_Version            : Ada (Proc_Spec);
    .Show_Configuration               : Ada (Proc_Spec);
    .Show_Directory_Information       : Ada (Proc_Spec);
    .Show_Elaborated_Configuration    : Ada (Proc_Spec);
    .Show_Environment_Vpids           : Ada (Proc_Spec);
    .Show_Groups                      : Ada (Proc_Spec);
    .Show_Identity                    : Ada (Proc_Spec);
```

```
    .Show_Image_Options                : Ada (Proc_Spec);
    .Show_Iop_Kernel                   : Ada (Proc_Spec);
    .Show_Jobs                         : Ada (Proc_Spec);
    .Show_Job_Names                    : Ada (Proc_Spec);
    .Show_Load_Proc_Unit_Names         : Ada (Proc_Spec);
    .Show_Locks                        : Ada (Proc_Spec);
    .Show_Log_Throttle_State           : Ada (Proc_Spec);
    .Show_Machine_Id                   : Ada (Proc_Spec);
    .Show_Manager_State                : Ada (Proc_Spec);
    .Show_Memory_Hogs                  : Ada (Proc_Spec);
    .Show_Mts_Profiles                 : Ada (Proc_Spec);
    .Show_Mts_Stats                    : Ada (Proc_Spec);
    .Show_R1000_Configurations         : Ada (Proc_Spec);
    .Show_Session_Of_Job               : Ada (Proc_Spec);
    .Show_Site                         : Ada (Proc_Spec);
    .Show_Snapshot_Times               : Ada (Proc_Spec);
    .Show_Stats                        : Ada (Proc_Spec);
    .Show_Tape_Users                   : Ada (Proc_Spec);
    .Show_Tasks                        : Ada (Proc_Spec);
    .Show_Tokens                       : Ada (Proc_Spec);
    .Smooth_Snapshots                  : Ada (Proc_Spec);
    .Token_Information                 : Ada (Proc_Spec);
    .Training_Authorization            : Ada (Proc_Spec);
    .Uncode                            : Ada (Proc_Spec);
    .Uninstall                         : Ada (Proc_Spec);
    .Verify_Backup                     : Ada (Proc_Spec);
Tape                                   : Ada (Pack_Spec);
Telnet                                 : Ada (Pack_Spec);
Terminal                               : Ada (Pack_Spec);
Text                                   : Ada (Pack_Spec);
Transfer                               : Aad (Pack_Spec);
Transport_Route                        : Ada (Pack_Spec);
What                                   : Ada (Pack_Spec);
Work_Order                             : Ada (Pack_Spec);
```





```
package Access_List is

    subtype Name is String;  -- an object name

    Read   : constant Character := 'R';  -- objects and worlds
    Write  : constant Character := 'W';  -- objects only
    Delete : constant Character := 'D';  -- worlds only; same bit as W
    Create : constant Character := 'C';  -- worlds only
    Owner  : constant Character := 'O';  -- worlds only

    subtype Acl is String;
    -- String representations of access lists have the following syntax:
    --  Acl       ::= Acl_Entry [',' Acl_Entry]*
    --  Acl_Entry ::= Group '=>' Access
    --  Group     ::= Identifier
    --  Access    ::= Acc_Type+
    --  Acc_Type  ::= 'R' | 'W' | 'D' | 'C' | 'O' |
    --                'r' | 'w' | 'd' | 'c' | 'o'
    --  Examples:  "Phil => R , TRW => rw",  "Public=>RCOD"


    procedure Display (For_Object : Name := "<CURSOR>");

    -- Display the access list of the specified object(s).
    -- Output and error messages are send to current output.

    procedure Set (To_List    : Acl    := "Network_Public => RWCOD";
                   For_Object : Name   := "<SELECTION>";
                   Response   : String := "<PROFILE>");

    -- Set the access list for the specified object(s).
    -- Setting the access list requires "Owner" access to the containing world.
    -- Sends messages to a log that is under control of the Response parameter.

    procedure Default_Display (For_World : Name := "<CURSOR>");

    -- Display the default acl of the specified world(s) in an output window.
    -- Error messages are sent to the window in case of any error.
    -- Wildcards in the name are allowed.
    -- Non-world objects are filtered out of the display.
    -- A null display is produced if no worlds are referenced.

    procedure Set_Default (To_List   : Acl    := "Network_Public => RW";
                           For_World : Name   := "<SELECTION>";
                           Response  : String := "<PROFILE>");

    -- Set the default ACL for the specified world(s).
```

```
    -- Owner access to each world is required.
    -- Sends messages to a log that is under control of the Response parameter.
    -- A log is written indicating success or errors.
    -- Wildcards are allowed in the name.
    -- Any non-world objects referenced are ignored.
    -- A summary of the number of objects affected is included in the log.

    procedure Add (To_List    : Acl    := "Network_Public => RWCOD";
                   For_Object : Name   := "<SELECTION>";
                   Response   : String := "<PROFILE>");

    -- Add the access list to the existing value for the specified object(s).
    -- Changing the access list requires "Owner" access to the containing world.
    -- Sends messages to a log that is under control of the
    -- Response parameter.

    procedure Add_Default (To_List   : Acl    := "Network_Public => RW";
                           For_World : Name   := "<SELECTION>";
                           Response  : String := "<PROFILE>");

    -- Add the default ACL to the existing value for the specified world(s).
    -- Owner access to each world is required.
    -- Sends messages to a log that is under control of the Response parameter.
    -- A log is written indicating success or errors.
    -- Wildcards are allowed in the name.
    -- Any non-world objects referenced are ignored.
    -- A summary of the number of objects affected is included in the log.

    procedure Remove (Group      : String := ">>SIMPLE NAME<<";
                      For_Object : Name   := "<SELECTION>";
                      Response   : String := "<PROFILE>");
    --
    -- Remove the group from the specified object(s)' access list(s).
    -- Changing the access list requires "Owner" access to the containing
    -- world.  Sends messages to a log that is under control of the
    -- Response parameter.

    procedure Remove_Default (Group     : String := ">>SIMPLE NAME<<";
                              For_World : Name   := "<SELECTION>";
                              Response  : String := "<PROFILE>");
    --
    -- Remove the group from the specified world(s)' default access list(s).
    -- Owner access to each world is required.  Sends messages to the log that
    -- is under control of the Response parameter.  Wildcards are allowed
    -- in the name.  Any non-world objects referenced are ignored.
    -- A summary of the number of objects affected is included in the log.

end Access_List;
```





```
with Action;
with Directory;
with Machine;

package Action_Utilities is

    procedure Display_Action (Id : Action.Id);
    procedure Display_Action (Id : Integer);
    -- displays either not in progress or put_task_info (creating task_id)
    -- the second form converts the integer to an action.id and
    -- invokes the first form

    procedure Lock_Information (Version : Directory.Version);
    procedure Lock_Information
                 (Name    : Directory.Naming.Name  := "<Image>";
                  Version : Directory.Version_Name := Directory.All_Versions);
    -- displays the following information
    --   actions (if any) that have a read lock on the version
    --   action  (if any) that has an update lock on the version
    --  action  (if any) that has an overwrite lock on the version
    --  request queue of [task,action,mode] triples waiting for the object
    -- the second form does name resolution and then calls the first form

    procedure Display_Task (Task_Id : Machine.Task_Id);
    -- shows the user, session and job for the specified task

    procedure Display_Object (Version : Directory.Version);
    procedure Display_Object
                 (Class : Natural; Instance : Natural; Host : Machine.Id);
    -- displays the name and version of the specified object
    -- the second form construct a directory.version and calls the first form

    procedure Lock_Information
                 (Class : Natural; Instance : Natural; Host : Machine.Id);

end Action_Utilities;
```

```
package Activity is

    subtype Activity_Name is String;

    -- An Activity is a managed object that maintains a map between
    -- subsystems and pairs of views. The pair consists of a spec view and
    -- a load (non-spec) view of the subsystem. An activity name is a
    -- string name for the managed object. The view pair can be specified
    -- indirectly by associating a subsystem in one activity with another
    -- activity, which then maps the subsystem to a pair of views.

    -- In these Activity commands, the default Activity is the object
    -- selected in the accompanying window, the object associated with the
    -- accompanying window, or, as a last resort, The_Current_Activity.


    type Creation_Mode is (Differential, Exact_Copy, Value_Copy);

    -- When a subsystem is copied from one Activity to another, the entry
    -- in the destination activity can be created in three ways:

    -- Differential : In the destination activity, the subsystem is mapped
    --                to the source Activity.

    --  Exact_Copy   : In the destination activity, the subsystem is mapped
    --                 to the same object it mapped to in the source
    --                 activity;  this may be either a view or an activity.

    --  Value_Copy   : In the destination activity, the subsystem is mapped
    --                 to the view currently associated with the subsystem
    --                 in the source activity.


    subtype Subsystem_Name is String;
    -- String name of a World directory.

    subtype View_Simple_Name is String;
    subtype View_Name        is String;
    -- View_Name = Subsystem_Name & '.' & View_Simple_Name

    -- A View is a world whose enclosing world is a Subsystem world.
    -- Any number of directories may come between a view and its subsystem.
    -- Hence, the view's subsystem is implicit in the full name of the
    -- view. The simple name of the view is used where the name of the
    -- subsystem is easily derived from other parameters.

    subtype View_Or_Activity_Name is String;
```





```
    -- An activity can be used to indirectly specify a view.

    subtype Unit_Name is String;
    -- The string name for an Ada library unit nested within a view of a
    -- subsystem.


    function Nil return Activity_Name;

    -- The name of the canonical activity with no subsystems;
    -- the empty activity.


    procedure Current (Response : String := "<PROFILE>");

    -- Prints the name of the activity currently associated with the
    -- running job; if no Activity has been associated with the job, it
    -- then returns the Activity currently associated with the running
    -- sesssion.


    function The_Current_Activity return Activity_Name;

    -- returns the name of the current activity; as defined above.


    procedure Set (The_Activity : Activity_Name := "<ACTIVITY>";
                   Response     : String        := "<PROFILE>");

    -- Makes The_Activity the current activity for the running job only.


    procedure Set_Default (The_Activity : Activity_Name := "<ACTIVITY>";
                           Response     : String        := "<PROFILE>");

    -- Makes Activity the current activity for the session.  If the job's
    -- activity is nil, set that as well.


    procedure Enclosing_View (Unit     : Unit_Name := "<IMAGE>";
                              Response : String    := "<PROFILE>");

    -- Prints the name of the enclosing view (either a load or spec view);


    function The_Enclosing_View
               (Unit : Unit_Name := "<IMAGE>") return View_Name;

    -- The name of the enclosing view (either a load or spec view);
```

```
procedure Enclosing_Subsystem (View     : View_Name := "<IMAGE>";
                               Response : String    := "<PROFILE>");

-- Prints the name of the subsystem that encloses the View, which may
-- be either a Spec or Load view.


function The_Enclosing_Subsystem
            (View : View_Name := "<IMAGE>") return Subsystem_Name;

-- The name of the subsystem that encloses the View, which may
-- be either a Spec or Load view.


procedure Create (The_Activity : Activity_Name := ">>ACTIVITY NAME<<";
                  Source       : Activity_Name := Activity.Nil;
                  Mode         : Creation_Mode := Activity.Exact_Copy;
                  Response     : String        := "<PROFILE>");

-- Create a new Activity object.  If the Source activity is not Nil,
-- its contents are copied to the new activity according to the
-- specified Mode.


procedure Add (Subsystem    : Subsystem_Name        := "<CURSOR>";
               Load_Value   : View_Or_Activity_Name := Activity.Nil;
               Spec_Value   : View_Or_Activity_Name := Activity.Nil;
               The_Activity : Activity_Name         :=
                  Activity.The_Current_Activity;
               Mode         : Creation_Mode         := Activity.Exact_Copy;
               Response     : String                := "<PROFILE>");

-- Add a subsystem to an existing Activity. If the load or spec values
-- are activities, the mapping is created according to the specified
-- mode.  The Load_Value and Spec_Value names are resolved in the
-- context of the given Subsystem, so that View_Simple_Names may be
-- used.


procedure Remove (Subsystem    : Subsystem_Name := "<SELECTION>";
                  The_Activity : Activity_Name  :=
                     Activity.The_Current_Activity;
                  Response     : String         := "<PROFILE>");

-- Remove a subsystem from an Activity.


procedure Set_Spec_View (Spec_View : View_Or_Activity_Name := "<CURSOR>";
                         Subsystem : Subsystem_Name := "";
                         Mode : Creation_Mode := Activity.Differential;
                         The_Activity : Activity_Name :=
                            Activity.The_Current_Activity;
                         Response : String := "<PROFILE>");
```





```
-- If Spec_View designates a view, associates the given view as the spec
-- view for the subsystem that contains the view.

-- If Spec_View designates an activity, associates the spec view defined
-- in the given source activity as the new spec view of the given
-- subsytem in the destination Activity. The mapping is created
-- according the given Mode.

-- The Spec_View parameter is resolved in the context established by the
-- Subsystem parameter.  The subsystem is derived from the Spec_View
-- parameter if it denotes a view, otherwise the Subsystem parameter
-- must be given.


procedure Set_Load_View (Load_View : View_Or_Activity_Name := "<CURSOR>";
                         Subsystem : Subsystem_Name := "";
                         Mode : Creation_Mode := Activity.Differential;
                         The_Activity : Activity_Name :=
                            Activity.The_Current_Activity;
                         Response : String := "<PROFILE>");

-- If Load_View designates an activity, associates the given View as the
-- load view for the subsystem that contains the view.

-- If Load_View designates an activity, associates the load view defined
-- in the given Source activity as the new load view of the given
-- subsystem in the named Activity. The mapping is created according the
-- given Mode.

-- The Load_View parameter is resolved in the context established by the
-- Subsystem parameter.  The subsystem is derived from the Load_View
-- parameter if it denotes a view, otherwise the Subsystem parameter
-- must be given.


procedure Display (Subsystem    : Subsystem_Name := "?";
                   Spec_View    : View_Name      := "?";
                   Load_View    : View_Name      := "?";
                   Mode         : Creation_Mode  := Activity.Value_Copy;
                   The_Activity : Activity_Name  :=
                      Activity.The_Current_Activity;
                   Response     : String         := "<PROFILE>");

-- Display the mappings between subsystems and views defined by the
```

```
-- given activity.  Only the mappings that match the patterns given in
-- the Subsystem, Spec_View, and Load_View parameters are listed. (The
-- default is to list all mappings in the activity.)  In the Value_Copy
-- mode, all indirect references are resolved and only the resolution
-- is displayed.  In the Exact_Copy mode, Indirect mappings are not
-- resolved and the name of the source activity is displayed. In the
-- Differential mode, the indirect mappings are resolved and both the
-- resolution and the original indirect activity are displayed.


procedure Edit (The_Activity : Activity_Name := "<ACTIVITY>");

-- Invoke the Activity object editor on the given Activity.


procedure Insert (Subsystem : Subsystem_Name        := ">>SUBSYSTEM NAME<<";
                  Spec_View : View_Or_Activity_Name := "";
                  Load_View : View_Or_Activity_Name := "");

-- Inserts the specified subsystem mapping into the activity associated
-- with the command window. (The current activity is brought up in an
-- Activity window and modified if the command is not associated with
-- an Activity window). The given names may specify a view or another
-- activity. If the subsystem name is omitted, it is inferred from the
-- view names.


procedure Change (Spec_View : View_Or_Activity_Name := "";
                  Load_View : View_Or_Activity_Name := "");

-- The selected subsystem mapping is changed to the new values given in
-- the Views specification.  Valid only in an Activity window.


procedure Write (File : Activity_Name := ">>ACTIVITY NAME<<");

-- Copies the current content of the Activity window to the designated
-- File. Valid only in an Activity window.


procedure Visit (The_Activity : Activity_Name := "<ACTIVITY>");

-- Same as Edit, except that if the command is given on an activity
-- window, the new activity is displayed in that window rather than in
-- a new one.
```





```
procedure Merge (Source    : Activity_Name  := ">>ACTIVITY NAME<<";
                 Subsystem : Subsystem_Name := "?";
                 Spec_View : View_Name      := "?";
                 Load_View : View_Name      := "?";
                 Mode      : Creation_Mode  := Activity.Exact_Copy;
                 Target    : Activity_Name  := "<ACTIVITY>";
                 Response  : String         := "<PROFILE>");

-- The subsystem mappings defined in the Source Activity that match the
-- given subsystem and view patterns are copied to the Target activity
-- according to the specified Creation mode. New subsystems are added
-- to the Target activity if necessary; Existing subsystem mappings are
-- replaced. The default Target activity is the current selection/image.

end Activity;
```

```
package Ada is

    procedure Code_Unit;
    -- Bring the unit corresponding to current image to the coded state.
    -- May involve coding subunits, parent unit, or corresponding visible
    -- part, but no closure operation is performed.  If the operation
    -- succeeds, the unit will be read-only.

    procedure Install_Unit;
    -- Bring the unit corresponding to current image to the installed
    -- state.  Will install no other units; may reduce subunits or parent
    -- unit to installed, but no closure operation is performed.  If the
    -- operation succeeds, the unit will be read-only.

    procedure Source_Unit;
    -- Bring the unit to source state such that its library declaration has
    -- the appropriate name and the image is read-only.

    procedure Withdraw (Name : String := "<IMAGE>");
    -- Edit the indicated unit, removing its declaration from the library.

    procedure Diana_Edit (Name : String := "<CURSOR>");
    -- Show a read-only image of the internal form of the Diana tree
    -- corresponding to the image given.

    procedure Install_Stub;
    -- Make the stub for the current compilation unit have the real name
    -- of the unit rather than its _Ada_nn_ name.

    procedure Make_Inline;
    -- Make a separate subunit body into an inline unit body

    procedure Make_Separate;
    -- Make an inline subunit body be a separate subunit body

    procedure Other_Part (Name     : String  := "<IMAGE>";
                          In_Place : Boolean := False);
    -- If a new window is required, In_Place indicates that the current
    -- frame should be used.

    procedure Replace_Id (Old_Id : String := ">>OLD NAME<<";
                          New_Id : String := ">>NEW NAME<<");
    -- For the current selection, change all occurrences of Old_ID into
    -- occurrences of New_ID.  Only changes Ada identifier references that
    -- match exactly.
```





```
    procedure Show_Usage (Name    : String  := "<CURSOR>";
                          Global  : Boolean := True;
                          Limit   : String  := "<ALL_WORLDS>";
                          Closure : Boolean := False);
    -- Show uses of the indicated item.
    -- Global => mark units other than the one indicated.
    -- Limit specifies the range of units if Global is true.
    -- Closure causes Show_Usage to find indirect references, e.g. renames.

    procedure Show_Unused (In_Unit           : String  := "<IMAGE>";
                           Check_Other_Units : Boolean := True);
    -- Show the declarations in a unit that are not referenced

    procedure Create_Body (Name : String := "<IMAGE>");
    -- Create a body declaration corresponding to the indicated
    -- declaration or visible part.

    procedure Create_Private (Name : String := "<IMAGE>");
    -- Create a private part declaration for each private type that still
    -- requires one.

    procedure Get_Errors;
    -- Restore the error underlining from the last compile, semanticize,
    -- etc.

    procedure Insert_Blank_Line (Repeat : Positive := 1);
    -- Insert repeat blank lines before the current line

    procedure Delete_Blank_Line (Repeat : Positive := 1);
    -- Delete repeat blank lines at the current cursor

    procedure Expand_Names (Name             : String  := "<SELECTION>";
                            Prefix_Standard  : Boolean := False;
                            Prefix_Unit      : Boolean := False;
                            Expand_Operators : Boolean := False);
    -- Expands names in the named Ada fragment. Prefix_Standard causes
    -- names from Standard to get prefixed.  Prefix_Unit causes names
    -- from the current unit to get prefixed.  Expand_Operators causes
    -- operators (such as "=" and "+") to get prefixed.

end Ada;
```

```
with Machine;

package Archive is

    procedure Save (Objects  : String := "<IMAGE>";
                    Options  : String := "R1000";
                    Device   : String := "MACHINE.DEVICES.TAPE_0";
                    Response : String := "<PROFILE>");

    -- Save a set of objects (files, Ada units, etc.) to a tape or directory
    -- such that they may be restored to their original form at a later time
    -- or on another system.

    -- The Objects parameter specifies the primary objects to be saved.  It
    -- can be any naming expression.  By default, the current image is saved
    -- unless there is a selection on that image, in which case the selected
    -- object is saved.  Normally, the specified object(s) and all contained
    -- objects are archived; this feature can be disabled.

    -- The Options parameter specifies the type of tape to be written and
    -- options to control what is saved.  The Options parameter for each of
    -- the Archive operations is written as a sequence of option
    -- names separated by spaces or commas.  Options with arguments are
    -- given as an option name followed by an equal sign followed by a
    -- value.

    ---------------------------------------------------------------------------

    -- The save options are:
    --
    --   FORMAT = R1000 | R1000_LONG | ANSI
    --      R1000
    --        Writes an ANSI tape with the data file followed by the index
    --        file.  The images of the objects being saved are written
    --        directly to the tape. This is the default.

    --      R1000_LONG
    --        like R1000 format but the data file is written to one ANSI tape
    --        and the index file to a second ANSI tape.

    --      ANSI
    --        Writes the data to a temporary file and then writes both index
    --        and data file to a tape using ANSI tape facilities.

    --   LABEL=(<any balanced string>)
    --      An identifying string written at the head of the archived data.
    --      The label parameter allows the user to specify a string that
```





```
    --      will be put at the front of the index file.  When a restore is
    --      done the label specified to the restore procedure will be
    --      checked against the one on the save tape.

    --   NONRECURSIVE
    --      Save only the objects resolved to by the Objects parameter.  Do
    --      not recursively save objects that are inside of other objects.
    --      The default is to save the objects mentioned in the Objects
    --      parameter and all objects contained in them.
    --      To save a world and a subset of its contents one can say:
    --      Save (Objects => "[!FOO?,~!FOO.ABC?,~!FOO.DEF?]", ...,
    --            Options => "R1000 NONRECURSIVE");

    --   AFTER=<time_expression>
    --      Only objects changed after the time represented by
    --      <time_expression> will be archived.  The <time_expression>
    --      should be acceptable to the time_utilities.value function.

    --   STARTING_AT=<time_expression>
    --      the archive will delay until the given time
    --      (after the mount request has been processed).

    --   EFFORT_ONLY
    --      The EFFORT_ONLY option causes a list to be printed of the names
    --      of the units that would be saved if the archive command
    --      was given with this set of parameters.

    --   CODE [=load_proc_list]
    --      The CODE option specifies that code is to be archived for the
    --      named load procs. If no specification follows the
    --      CODE option, the Objects parameter specification is used
    --      instead.

    --   COMPATIBILITY_DATABASE (CDB) [=<Subsystems>]
    --      Causes the full compatibility database for each subsystem
    --      specified to be archived. If no subsystems are specified with
    --      the option, the Objects parameter specification is used instead.

    --      When Ada units in a subsystem are archived, the relevant
    --      portions of the subsystem Compatibility Database is
    --      automatically archived with them.  Therefore, this option is
    --      required only in special situations, primarily when one needs to
    --      "sync up" a primary and a secondary subsystem.

    --      To archive just Compatibility Databases, use
    --      Save ("Subsystems", "CDB");

    --      To archive compatibility databases with other objects, use
    --      Save ("Other Stuff", "CDB=Subsystems");
```

```
--      The "Subsystems" and "Other Stuff" specifications will usually
--      describe disjoint sets of objects.

--   IGNORE_CDB
--      Specifies that no compatibility database information is to be
--      saved.

--   LINKS [=<worlds>]
--      Causes only the link pack for each world specified to be archived.
--      If no worlds are specified with the option, the Objects
--      parameter specification is used instead.

--   PREFIX=<naming pattern>
--      A naming pattern that is saved with the archived objects, which
--      can be recalled as the For_Prefix when the data is Restored.
--      When set to an appropriate value, the restorer need not know
--      exactly the names of the archived objects to be able to restore
--      them to a new place.  If this option is not given, the value
--      used is derived from the Objects parameter.

--   UNLOAD
--      A boolean option.  When True (the default), causes the tape to
--      be rewound and unloaded after the operation is complete.  When
--      False, this option causes the tape to be rewound to the beginning
--      and to remain online and available for subsequent requests.
--      When the tape is left online, subsequent requests send a tape-
--      mount request to the operator's console, which must be answered
--      before the tape can be accessed.

-- For downward compatibility the following options are provided.
--
--   DELTA0
--      write a tape which can be read on a delta0 system.
--
--   DELTA1
--      write a tape which can be read on a delta1 system.
--
--   VERSION=<archive_version_number>
--      write a tape that can be read by a version of source
--      earlier than the current one. The argument is a three digit
--      integer. For example, version=440.


-- The Device parameter can be set to the name of a directory.  In this
-- case the index and data files are written to that directory.  The
-- tape format option is irrelevant in this case.

----------------------------------------------------------------------------
```





```
procedure Restore (Objects    : String := "?";
                   Use_Prefix : String := "*";
                   For_Prefix : String := "*";
                   Options    : String := "R1000";
                   Device     : String := "MACHINE.DEVICES.TAPE_0";
                   Response   : String := "<PROFILE>");

-- Restore an object or a set of objects from an Archive Tape.

-- If the archive is on a tape then the tape format option given to
-- Restore should be the same as that given during the save.  If the
-- archive is in a directory then the device parameter on the restore
-- should be set to that directory.

-- The Objects parameter may be any wildcard pattern specifying the
-- objects to be restored.
--
-- For example:
--     !USERS.FOO.CLI.TEST
--     [!USERS.FOO.TESTS.@, !USERS.FOO.LOGS.ABC]

-- The pattern in the Objects parameter is compared against the full
-- names of the saved objects. The objects whose names match the Objects
-- parameter specification are restored. If the name denotes an Ada
-- unit all of its parts are restored from the tape. If the name denotes
-- a world or directory all of its subcomponents are restored.

-- The Use_Prefix and For_Prefix parameters provide a simple means for
-- changing the names of the archived objects when they are restored.

-- If the Use_Prefix is the special default value, "*", the For_Prefix
-- is ignored and the objects are restored using the names they had when
-- they were saved.

-- If the Use_Prefix is not "*", it must specify the name of an object
-- into which the archived objects can be restored.  The name for a
-- restored object is derived from the name of the archived object by
-- replacing the shortest portion of the name matched by the For_Prefix
-- with the value of the Use_Prefix.  If the For_Prefix is "*" the
-- archived objects are restored using the Default_Prefix stored with
-- the archived data.

-- For example:
--
--   Restore (Objects    => "!A.B.C.D.E",
--            Use_Prefix => "!X.Y",
--            For_Prefix => "!A.B.C");
--
-- will restore to !X.Y.D.E.
```

```
-- The For_Prefix may contain wildcard characters (#, @, ?) and the
-- Use_Prefix parameter may contain substitution characters (@ or # only).

-- For example:
--
--   Restore (Objects    => "[!A.B.TEST1, !D.E.F.TEST2]"
--            For_Prefix => "?.@"
--            Use_Prefix => "!C.D.@");
--
--   will restore to !C.D.TEST1 and !C.D.TEST2

-- If the object named by the prefix of the target name of an object
-- being restored doesn't exist, that object will be created as a set of
-- nested libraries. So, for example, if the For_Prefix is !A.B and the
-- unit being restored is then !A.B.X.Y.Z and ...X.Y hasn't been saved on
-- the tape then !A, !A.B, !A.B.X, !A.B.X.Y will be created.

---------------------------------------------------------------------------

-- The following options are allowed in the Options parameter:
--
--   FORMAT and LABEL
--      as in the save option.

--   NONRECURSIVE
--      prevents subcomponents of libraries and Ada units from being
--      implicitly restored. for example:
--       Archive.Restore
--          (Objects => "[!USERS.FOO, !USERS.FOO.CLI, !USERS.FOO.CLI.@]",
--           Options => "R1000 NONRECURSIVE");
--       will restore only the named objects and not their substructure.

--   ALL_OBJECTS
--      All specified objects are restored.  This is the default.

--   NEW_OBJECTS
--      Only specified objects that don't already exist on the target
--      machine are restored.

--   UPDATED_OBJECTS
--      Only specified objects that already exists on the target are
--      restored, but only if the update time of the archived object
--      is greater than the update time on the target object.

--   CHANGED_OBJECTS
--      Restore both new and updated Objects.

--   EXISTING_OBJECTS
--      Only specified objects that already exists on the target
```





```
--      are restored.

--   DIFFERENT_OBJECTS
--      Only specified objects that already exists on the target
--      and have a different update time from the archived object are
--      restored.

--   REPLACE
--       Given an object that is being restored that already exists
--       on the target, this option will cause the restore operation
--        (1) to unfreeze the target object if it is frozen.
--        (2) If the target object is an installed or coded Ada unit
--            with clients, it is demoted to source using Compilation.
--            Demote with the "<ALL_WORLDS>" parameter.
--            Any frozen dependents will be unfrozen.
--        (3) if the parent library into which an object is being
--            restored is frozen, the parent will be unfrozen to restore
--            the object then refrozen.

--   PROMOTE
--      After they are restored, any Ada units will be promoted to the
--      state they were in when they were archived.

--   REMAKE [=NO_MAINS]
--      Like the promote option with the further effect that
--      any objects outside the restore set which were demoted
--      because the replace option was given will be repromoted.
--      If no_mains is specified dependent main programs will not
--      be recoded.

--   GOAL_STATE = ARCHIVED | SOURCE | INSTALLED | CODED
--      Specify that all ada objects restored should (if possible)
--      end up in the given state.

--   EFFORT_ONLY
--      Show what would be restored if restore is run with this
--      set of parameters.

--   CODE [=load_proc_list>]
--      Specifies that just the Code Archive Object for the named
--      load_procs are to be restored.  This option is needed only when
--      it is desired to restore a Code Archive Object from an archive
--      that also contains the spec for that load_proc, which is not
--      to be restored.

--   COMPATIBILITY_DATABASE, (CDB) [=<Subsystems>]
--      Specifies that the Compatibility Databases for just the named
--      subsystems are to be restored.
```

```
--   IGNORE_CDB
--      Specifies that no compatibility information is to be restored.

--   LINKS [=<worlds>]
--      specifies that just the link packs for the named worlds are to
--      be restored. if no argument is given all link packs of all worlds
--      on the tape are restored.

--   PRIMARY
--      restore the compatibility database as a primary, rather than as a
--      secondary (which is the default).

--   REVERT_CDB
--      allow the compatibility database to be overwritten by the values
--      in the restore.

--   OBJECT_ACL=<acl_value>
--   WORLD_ACL=<acl_value>
--   DEFAULT_ACL=<acl_value>
--      Specifies the Access Control List for restored objects
--      (OBJECT_ACL) and worlds (WORLD_ACL) and the default ACL for
--      restored worlds (DEFAULT_ACL).
--      The value is either an ACL specification or one of the special
--      values RETAIN, ARCHIVED, INHERIT.
--      - RETAIN means to set the final acl of an already existing object
--        to the value it had before the restore. If the object doesn't
--        exist the archived acl value will be used. This is the default.
--      - ARCHIVED means to use the ACL archived with the object.
--      - INHERIT means to use the standard inheritence rules for new
--        versions of objects.

--   BECOME_OWNER
--      Modify the ACL of all restored objects such that the restorer
--      becomes the owner of the restored object.

--   TRAILING_BLANKS=integer
--      Specifies the number of trailing blanks which are to be
--      considered significant when parsing ada units during the
--      restore. If a line ends in more than this number of blanks,
--      the line break will be preserved in the image of the restored
--      ada unit. The default is 2.

--   UNCONTROL
--      specifies that controlled objects which are checked in
--      will be made uncontrolled before being overwritten.
--      objects will be recontrolled at the end of the archive.
--      only valid if the replace option is also given.

--   REQUIRE_PARENTS
```





```
--      require that the parent context for a unit to be restored
--      already exists. default is false.

--   VOLUME
--      specifies which volume archive is to create worlds on.

--   VERBOSE
--      Specifies that extra log messages are to be generated describing
--      more fully the steps of the restore process.

--   UNLOAD
--      A boolean option.  When True (the default), causes the tape to
--      be rewound and unloaded after the operation is complete.  When
--      False, this option causes the tape to be rewound to the beginning
--      and to remain online and available for subsequent requests.
--      When the tape is left online, subsequent requests send a tape-
--      mount request to the operator's console, which must be answered
--      before the tape can be accessed.

---------------------------------------------------------------------------

procedure List (Objects  : String := "?";
                Options  : String := "R1000";
                Device   : String := "MACHINE.DEVICES.TAPE_0";
                Response : String := "<PROFILE>");

-- Produce a listing of the names of the objects on an Archive tape.

-- The Objects parameter specifies the objects to be listed.  Wildcards
-- are permitted, so if Objects = "?", the default, then all Objects are
-- listed.

-- The Options parameters are:

--   FORMAT and  LABEL
--      as in the Save options.

--   NONRECURSIVE
--      don't list items on tape which are subcomponents of objects
--      selected by first paramater.

---------------------------------------------------------------------------

procedure Copy (Objects    : String := "<IMAGE>";
                Use_Prefix : String := "*";
                For_Prefix : String := "*";
                Options    : String := "";
                Response   : String := "<PROFILE>");
```

```
-- Copy objects from one location to another, including between
-- machines on the same network.

-- The Objects parameter specifies where the objects are to be gotten
-- from as in an Archive.Save.

-- The Objects and Use_prefix parameters consist of an (optional)
-- machine name followed directly by an objects name.
-- A machine name has the form !!name.
--   Example:
--      !!machine1!users.foo
-- Another acceptable way to say the same thing is !!machine1.users.foo
--
-- The machine name on the objects parameter specifies the source machine.
-- The machine name on the use prefix specifies the destination machine.
-- If either machine is the one on which the command is being given
-- it is not necessary to give the machine name.

-- The non-machine name part of the Objects parameter is a name like that
-- given to the save operation.
--
-- The Use_Prefix/For_Prefix parameters specify where the objects
-- are to go as in Archive.Restore.

-- If the Use_Prefix parameter is "*" or just a machine name, then the
-- source Objects are moved to the same place on the destination machine
-- as specified by the source.  The For_Prefix parameter is ignored.

-- If neither Objects nor Use_Prefix have a machine name then the
-- objects are copied from the source to the Use_Prefix on the
-- current machine.
--
-- If it is desired to transfer the same set of objects to multiple
-- targets in the same command a set of target names can be
-- specified as the use_prefix in one of the following two ways.
--   The use prefix can be of the form:
--     [use_prefix1, ..., use_prefixn]<optional_naming_expression>
--     examples:
--       archive.copy (..., use_prefix => "[m1,m2]", ...);
--       archive.copy (..., use_prefix => "[m1,m2,m3]!users.foo", ...);
--
--   The use prefix can be of the form:
--      _filename<optional_naming_expression_beginning_with_!>
--   The filename should contain a list of use_prefix's, one per line.
--     examples:
--       archive.copy (... use_prefix => "_targets", ...);
--       archive.copy (... use_prefix => "_targets!users.foo", ...);
--     where targets is a text file containing (e.g)
--       m1
```





```
--       m2
--
-- In both of the above cases the leading !! in machine names is optional.

----------------------------------------------------------------------------

-- The Options parameter has the following options.
--
--   AFTER, CODE, CDB, LINKS, IGNORE_CDB
--   NONRECURSIVE, EFFORT_ONLY
--      as in the save operation.
--
--   ALL_OBJECTS, NEW_OBJECTS, UPDATED_OBJECTS, CHANGED_OBJECTS,
--   EXISTING_OBJECTS, DIFFERENT_OBJECTS
--   PROMOTE, REPLACE, UNCONTROL, REMAKE, GOAL_STATE,
--   PRIMARY, REVERT_CDB, REQUIRE_PARENTS, VOLUME
--   BECOME_OWNER, OBJECT_ACL, WORLD_ACL, DEFAULT_ACL
--   VERBOSE
--      as in the restore operation.

--  ENABLE_PRIVILEGES
--      cause the archive server (and the copy job) to attempt to
--      enable_privileges.

----------------------------------------------------------------------------

-- Examples of calls:
--
--  Copy (Objects    => "!USERS.JMK.CLI",
--        Use_Prefix => "!!M1");
--
--   will copy the CLI directory in !USERS.JMK on the
--   current machine to machine M1 !USERS.JMK.CLI.
--
--  Copy (Objects => "!!M2!USERS.OLLIE.CLI");
--
--   will copy !USERS.OLLIE.CLI on M2 to !USERS.OLLIE.CLI on the
--   current machine.
--
--  Copy (Objects    => "!!M3!USERS.JMK.CLI.CMD",
--        Use_Prefix => "!USERS.OLLIE",
--        For_Prefix => "!USERS.JMK.CLI");
--
--   will copy the file !USERS.JMK.CLI.CMD on M3 to
--   !USERS.OLLIE.CMD on the current machine.
--   note when repositioning Objects it is necessary to give a
--   for_prefix which is a prefix of the Objects part of the
--   source parameter.
--
```

```
    --  Copy (Objects    => "!!M1!USERS.JMK.ILFORD",
    --        Use_Prefix => "!!M2!AGFA",
    --        For_Prefix => "!USERS.JMK");
    --
    --    will copy !USERS.JMK.ILFORD from machine M1 to
    --    machine M2 !AGFA!ILFORD
    --
    --  Copy (Objects    => "!USERS.JMK.CLI",
    --        Use_Prefix => "!!M1",
    --        Options    => "REPLACE AFTER=12/25/86");
    --
    --    will copy those files which have changed since 12/25/86 in
    --    !USERS.JMK.CLI on the current machine to machine M1 !USERS.JMK.CLI
    --    Any existing files with the same names will be overwritten.

    ---------------------------------------------------------------------------

    procedure Server;

    -- start the archive server;


    procedure Status (For_Job : Machine.Job_Id);

    -- Prints information about the status of the Archive job specified.
    -- Can be the job number of an Archive Server or of a job running
    -- Archive.Copy, Archive.Restore, or Archive.Save.

end Archive;
```





```
with Compilation;
with System_Utilities;

package Cmvc is

    -- All CMVC commands raise Profile.Error if any error is detected
    -- and Profile.Propagate or Profile.Raise_Error is true

    ---------------------------------------------------------------------------

    -- Some of the following reservation commands take the name of an object
    -- that appears in more than one view.  The naming expression
    --      !mumble.subsystem.[view1, view2, view3].units.object
    -- is useful for such times.

    procedure Check_Out
                 (What_Object                   : String := "<CURSOR>";
                  Comments                      : String := "";
                  Allow_Implicit_Accept_Changes : Boolean := True;
                  Allow_Demotion                : Boolean := False;
                  Remake_Demoted_Units          : Boolean := True;
                  Goal                          : Compilation.Unit_State :=
                     Compilation.Coded;
                  Expected_Check_In_Time        : String := "<TOMORROW>";
                  Work_Order                    : String := "<DEFAULT>";
                  Response                      : String := "<PROFILE>");

    -- Check out reserves one or more objects (specified by What_Object) so
    -- that they may be modified in only one view.  All of the
    -- objects specified must belong to the same working view.
    -- An object must be 'controlled' to be reserved (see Make_Controlled),
    -- a warning is issued for objects that are not controlled.

    -- The reservation spans all of the views that share the
    -- same reservation token for the element.

    -- This command implicitly accepts changes in the checked out object,
    -- updating the value of the object to correspond to the most
    -- recent generation of that element/reservation token pair.
    -- Demotions caused by the implicit accept_changes may be remade to the
    -- goal specified.

    -- The Comments field is stored with the notes for the object.
    -- If What_Object is a set, the comment is stored with all of them.

    -- Expected_Check_In accepts any string that Time_Utilities.Value
    -- will accept.
```

```
procedure Check_In (What_Object : String := "<CURSOR>";
                    Comments    : String := "";
                    Work_Order  : String := "<DEFAULT>";
                    Response    : String := "<PROFILE>");

-- Release the reservation on the object.  What_Object may
-- specify a set of objects.  This command only applies to
-- the controlled objects in the set and will note any
-- objects that are not controlled.

-- Comments are treated as in Check_Out

procedure Accept_Changes
             (Destination          : String                     := "<CURSOR>";
              Source               : String                     := "<LATEST>";
              Allow_Demotion       : Boolean                    := False;
              Remake_Demoted_Units : Boolean                    := True;
              Goal                 : Compilation.Unit_State :=
                 Compilation.Coded;
              Comments             : String                     := "";
              Work_Order           : String                     := "<DEFAULT>";
              Response             : String                     := "<PROFILE>");

-- This operation updates the Destination to reflect changes
-- (objects that have been checked in) specified by Source.
-- Demoted units may be repromoted to the specified goal.

-- The Destination is either a view or a set of objects (all in
-- one view).   Specifying the view is equivalent to specifying
-- all the objects in the view.  Uncontrolled objects in the
-- destination are ignored except that a note is issued.

-- The Source is either "<LATEST>", a view, a configuration,
-- or a set of objects all in one view.

-- If the Source is "<LATEST>", the destination objects
-- will be updated to the most recently checked in version.
-- If the most recent generation of a source object is currently
-- checked out, the previous generation is used and a warning
-- is issued.

-- If the Source is a view and the Destination is a view, this command
-- is basically "Make the Destination view look exactly like the
-- Source view".  Every controlled object in the source is copied
-- to the destination and the configuration in the destination
-- is updated.  This includes new objects which did not previously
-- exist in the destination.  If the destination has a more recent
-- version than the source, the destination will not be updated and
-- a warning is issued.  In particular, if objects are checked out in
```





```
-- the destination, they will not be changed.
-- If objects are checked out in the source this operation
-- will use the previously checked in version of the object and
-- a warning will be issued.

-- If the Source is a view and the Destination is a set of objects,
-- the destination objects are updated to the corresponding objects
-- in the source view, as above.

-- If the source is a configuration it is identical to having the
-- source be a view except that the configuration specifies the
-- versions to use and they may be older (less up to date) than
-- the ones in the destination.  Thus if the source is a configuration
-- then destination objects may "go backwards", while this will not
-- happen if the source is a view.

-- If the source is a set of objects and the destination is a view,
-- the corresponding objects in the destination view are updated
-- to the source objects.

-- A common way of using  Accept_Changes is to use the default parameters
-- during normal development to accept changes made in other subpaths.
-- Then periodically an integration view (in the path) is updated by
-- first accepting all relevant subpaths into the integration view
-- (accept_changes (destination => integration_view, source =>
-- active_subpath_working_view)).
-- Then this integration view is compiled (and tested).  The subpaths are
-- then re-synchronized by accepting the integration view (source =>
-- integration_view, destination => destination_subpath_working_view).

-- In addition to synchronizing the source, this protocol updates
-- the libraries in such a way the relocation operates most effectively,
-- preventing compilation in many cases when changes move between views.

procedure Abandon_Reservation
             (What_Object          : String := "<SELECTION>";
              Allow_Demotion       : Boolean := False;
              Remake_Demoted_Units : Boolean := True;
              Goal                 : Compilation.Unit_State :=
                 Compilation.Coded;
              Comments             : String := "";
              Work_Order           : String := "<DEFAULT>";
              Response             : String := "<PROFILE>");

-- Forget about a check_out of some object, or set of objects.
-- This reverts the objects back to last checked in version.
-- This operation is an "undo" for Check_Out, except that it
-- does not undo the implict Accept_Changes that goes with
-- a Check_Out.  Demoted units may be repromoted to the specified goal.
```

```
procedure Revert
             (What_Object             : String := "<SELECTION>";
              To_Generation           : Integer := -1;
              Make_Latest_Generation  : Boolean := False;
              Allow_Demotion          : Boolean := False;
              Remake_Demoted_Units    : Boolean := True;
              Goal                    : Compilation.Unit_State :=
                 Compilation.Coded;
              Comments                : String := "";
              Work_Order              : String := "<DEFAULT>";
              Response                : String := "<PROFILE>");

-- Replace the contents of the specified object with the contents
-- of the specified generation.  The operation is equivalent to an
-- Accept_Changes from a configuration containing the specified
-- generation.

-- If Make_Latest_Generation is true, then the operation is equivalent to
-- a Check_Out, a copy of the specified generation into the object, and
-- a Check_In.

-- Generation of -n means n generations back; thus -1 => the previous
-- generation.

----------------------------------------------------------------------

-- The following commands allow the creation and interogation of
-- a note scratchpad for each element.  Descriptive information
-- regarding what is being changed, why, or whatever, can be put
-- into the scratchpad.

-- The notes for each element can also be manipulated through the
-- cmvc object editor (see cmvc.notes below).  In most instances
-- this interface will prove easier to use (for example, the
-- object need not be checked out to manipulate the notes).

procedure Get_Notes (To_File     : String := "<WINDOW>";
                     What_Object : String := "<CURSOR>";
                     Response    : String := "<PROFILE>");

-- Copy the notes from the object.  If To_File is the default, then
-- a new I/O window is created and the notes are copied into this window.
-- The first line of this window is the name of the object, which is
-- used by Put_ and Append_Notes to put the notes back.  The notes
-- displayed are those that go with the generation of the object pointed
-- at.  See Cmvc_History for ways of getting notes and other information
-- on a range of generations
```





```
-- The next three commands require the object in question to be
-- checked out.

procedure Put_Notes (From_File   : String := "<WINDOW>";
                     What_Object : String := "<CURSOR>";
                     Response    : String := "<PROFILE>");

-- Replace the notes for the specified object.  If the I/O window
-- was created by Get_Notes, the window (first line) contains the name
-- of the object to write back into, and What_Object is ignored.

procedure Append_Notes (Note        : String := "<WINDOW>";
                        What_Object : String := "<CURSOR>";
                        Response    : String := "<PROFILE>");

-- Append the specified text to the notes.  If Note is <WINDOW>,
-- the associated window must have been created by Get_Notes or
-- Create_Empty_Note_Window; in this case What_Object is ignored.
-- If note is a string, then that string is appended to the object
-- selected by What_Object.  If the content of Note is prepended with a
-- '_', Note is interpreted as a text file name, and the content of
-- that file is appended to the selected object.

procedure Create_Empty_Note_Window (What_Object : String := "<CURSOR>";
                                    Response    : String := "<PROFILE>");

-- Create an empty window (with no underlying directory object)
-- to be used for constructing notes for the specified object.
-- Typically, Append_Notes is used to actually add the text
-- to the object's notes.

----------------------------------------------------------------------

procedure Make_Controlled
             (What_Object            : String  := "<CURSOR>";
              Reservation_Token_Name : String  := "<AUTO_GENERATE>";
              Join_With_View         : String  := "<NONE>";
              Save_Source            : Boolean := True;
              Comments               : String  := "";
              Work_Order             : String  := "<DEFAULT>";
              Response               : String  := "<PROFILE>");
-- Make the object or objects specified by What_Object be subject to
-- reservation.  The objects must be in a working view and not
-- already controlled.  All objects must be in the same subsystem.
-- If Join_With_View is specified, the objects are joined with the
-- object in that view, using the reservation token specified by that view.
-- If no view is specified, the reservation token name is used if provided,
-- else the development path name of the view containing the object is
-- used to compute a new reservation token name.
```

```
-- The value of save_source is meaningful only the first time an
-- object with a particular name is controlled.  When the first object
-- with the name is controlled, save_source specifies whether or
-- not source will be saved in the CMVC database for all objects
-- with the same name.  When an object with the name has already
-- been controlled, the value of save_source must agree with the
-- value that was set for the first such object.
-- Note that setting save_source to false is the only way to control
-- files that do not have textual representation.
-- Also note that when save_source is false the following operations
-- behave differently:
--  1. Abandon_Reservation will not revert the object to its previous
--     state.
--  2. Check_Out will not cause the object to be updated to the latest
--     checked in value, unless that value exists in the directory system.
--  3. Accept_Changes will update the object only if the last checked in
--     object exists in the directory system.
--  4. Make_Controlled will not check that the new object being controlled
--     is equivalent to the last checked in value.

procedure Make_Uncontrolled (What_Object : String := "<CURSOR>";
                             Comments    : String := "";
                             Work_Order  : String := "<DEFAULT>";
                             Response    : String := "<PROFILE>");

-- Make an object or objects uncontrolled.
-- This means the objects are no longer subject to reservation
-- (in the enclosing view).

procedure Sever (What_Object                  : String := "<SELECTION>";
                 New_Reservation_Token_Name : String := "<AUTO_GENERATE>";
                 Comments                     : String := "";
                 Work_Order                   : String := "<DEFAULT>";
                 Response                     : String := "<PROFILE>");

-- Make the object or objects in the given working view have a separate
-- reservation.  If multiple objects are specified, all objects must be
-- in the same view.
-- A specific reservation token name can be provided or a new token
-- name will be generated.  Providing a token name is not allowed
-- to cause implicit joining to other views.

procedure Join (What_Object              : String := "<SELECTION>";
                To_Which_View            : String := ">>VIEW NAME<<";
                Reservation_Token_Name : String := "";
                Comments                 : String := "";
                Work_Order               : String := "<DEFAULT>";
                Response                 : String := "<PROFILE>");
```





```
-- Make object in two or more working views share a reservation.  The
-- objects may either be joined to a specific view or token (but only
-- one may be specified).  The objects being joined must be identical
-- to the last checked_in version in each joined set.

------------------------------------------------------------------------

procedure Merge_Changes
            (Destination_Object      : String  := "<SELECTION>";
             Source_View             : String  := ">>VIEW_NAME<<";
             Report_File             : String  := "";
             Fail_If_Conflicts_Found : Boolean := False;
             Comments                : String  := "";
             Work_Order              : String  := "<DEFAULT>";
             Response                : String  := "<PROFILE>");
-- Merge two versions of the same object together, leaving the result
-- in destination object.  In order for this command to succeed, the
-- Source_View and the view containing the Destination_Object must
-- have been copied from some common view sometime in the past, and
-- the configuration for that view must still exist.

-- Destination_Object must refer to the last generation; all changes must
-- have been accepted.

-- The command writes a report showing what it did, as well as changing
-- the destination object.  If the report_file name is "", the report
-- is written to Get_Simple_Name (Destination_Object) & "_Merging_Report".

-- Conflicts are defined to be regions of change in the source and
-- destination that directly overlap, ie the same line(s) have been
-- changed in both objects.  If Fail_If_Conflicts_Found is true,
-- no updating is done, but the report file is left.

-- If it is desired to rejoin the two objects after the merge, then
-- check out the Merge source object, copy the Merge Destination_Object
-- into the source, then Join the objects.

------------------------------------------------------------------------

function Imported_Views
           (Of_View                : String  := "<CURSOR>";
            Include_Import_Closure : Boolean := False;
            Include_Importer       : Boolean := False;
            Response               : String  := "<WARN>") return String;

-- return a string suitable for name resolution that names the union of
-- all of the imports specified by the view(s) Of_View.  These views
-- are in no particular order.
```

```
   ---------------------------------------------------------------------
   --                              IMPORTS                            --
   ---------------------------------------------------------------------

   -- CMVC supports selective importing of units when views are imported.
   -- This is accomplished using Imports_Restrictions and
   -- Exports_Restrictions.
   --
   -- Exports_Restrictions are subsets of exported Ada units controlled
   -- by the exporting view (spec view).  The subset is determined by the
   -- contents of a text file in the Exports directory of the view.  This
   -- file contains Naming expressions which, when resolved against the
   -- Units directory, produce a list of objects that are exported by
   -- that subset.

   -- Imports_Restrictions are further restrictions on what Ada units are
   -- to be imported.  The restriction specifies which export restriction
   -- to use (if any), a list of Ada units (using simple names) to
   -- exclude, and a list of units to rename.  A restriction is a text
   -- file, in the Imports directory, with the same name as the subsystem
   -- containing the view being imported.  The name of restriction
   -- may either be the simple name of the imported subsystem or the
   -- full name of the subsystem.  If the full name is used, the file
   -- name is formed by taking the path name, removing the leading "!" and
   -- changing all periods to underscores.  If both forms are found
   -- in the same directory, then the full name form takes precedence.
   -- Each line of the file specifies one thing.  The form of the lines are:
   --
   --   EXPORT_RESTRICTION=>restriction_name
   --       Specify the name of the export restriction.  No blanks are
   --       allowed.  If more than one restriction is specified, the
   --       union of all of the restictions is used.
   --   Object_Name     Link_Name
   --       Import Object_Name but make a link with Link_Name (a rename)
   --   ~Object_Name
   --       Dont import Object_Name
   --   Object_name
   --       Import Object_Name and use Object_Name for the link name
   --   @
   --      Import all Objects, except those removed above
   -- In all cases, the names provided above are simple names, ie no '.'s
   -- in them.

   ---------------------------------------------------------------------
   --                          SELECTING VIEWS                        --
   ---------------------------------------------------------------------

   -- In the following commands, wherever a view is called for, a naming set
   -- can be used.  A text file containing the names of configurations
```





```
   -- or views can also be used.  However, you must use the leading '_'
   -- convention supported by Naming.  Also, configuration names can be
   -- used in place of views anywhere, assuming that the view represented
   -- by the configuration still exists.


   ---------------------------------------------------------------------
   --                            SPEC VIEWS                           --
   ---------------------------------------------------------------------

   -- Spec views in CMVC are by default uncontrolled.  The reason for this
   -- is to allow free changing of specs in the load views, accepting the
   -- changes back and forth, then incrementally making the changes in the
   -- spec views.

   -- It controlling of spec views is desired, use Make_Controlled after
   -- creating the views.  But be forewarned that checking out a spec
   -- where an implicit accept is required will probably obsolesce all
   -- of the spec's clients.

   ---------------------------------------------------------------------

   procedure Release
                (From_Working_View          : String := "<CURSOR>";
                 Release_Name               : String := "<AUTO_GENERATE>";
                 Level                      : Natural := 0;
                 Views_To_Import            : String := "<INHERIT_IMPORTS>";
                 Create_Configuration_Only  : Boolean := False;
                 Compile_The_View           : Boolean := True;
                 Goal                       : Compilation.Unit_State :=
                    Compilation.Coded;
                 Comments                   : String := "";
                 Work_Order                 : String := "<DEFAULT>";
                 Volume                     : Natural := 0;
                 Response                   : String := "<PROFILE>");

   -- Create a new release view in the subsystem.  If Release_Name is
   -- "<AUTO_GENERATE>", the view will have the same name prefix as the
   -- working view, with _n_m appended as appropriate given the level.
   -- Otherwise Release_Name must be the simeple name of the new release.

   -- Since the new view is a release, it is frozen.  If From_Working_View
   -- names multiple views, each named working view is released as
   -- above, and the imports are adjusted so that the new releases
   -- reference each other as appropriate instead of the working views.
```

```
-- Views_To_Import specifies, perhaps by indirection through an activity,
-- a set of views to be used as imports by the new view(s).  This allows
-- changing imports during a release.  Imports already adjusted during
-- the releasing of working views will be left alone, otherwise
-- subsystems currently imported will be reimported.  In other words,
-- if this were an import command, Only_Change_Imports would be true.

-- If Compile_The_View is true, the compiler is run before the views
-- are frozen, trying to promote the units to the indicated Goal.
-- The views are frozen even if compilation fails.

-- This command creates a configuration object named release_name
-- and a state directory named release_name_state.  Both
-- objects are created in SUBSYSTEM.state.configurations.
-- The objects can be used by the build command to reconstruct
-- a view from the released configuration.

-- A controlled text object (state.release_history) is used by this
-- command.  Release enters the comments supplied with the command
-- into the notes for this object.  Feel free to check out and modify
-- this object to further describe what is going on.  This object is joined
-- across all of the releases and the working view of a subpath.
-- Furthermore, the object is checked out and in by the release command
-- in order to mark the time of the release.

------------------------------------------------------------------------

procedure Copy
            (From_View               : String := "<CURSOR>";
             New_Working_View        : String := ">>SUB/PATH NAME<<";
             View_To_Modify          : String := "";
             View_To_Import          : String := "<INHERIT_IMPORTS>";
             Only_Change_Imports     : Boolean := True;
             Join_Views              : Boolean := True;
             Reservation_Token_Name  : String := "<AUTO_GENERATE>";
             Construct_Subpath_Name  : Boolean := False;
             Create_Spec_View        : Boolean := False;
             Create_Load_View        : Boolean := False;
             Create_Combined_View    : Boolean := False;
             Level_For_Spec_View     : Natural := 0;
             Model                   : String := "<INHERIT_MODEL>";
             Remake_Demoted_Units    : Boolean := True;
             Goal                    : Compilation.Unit_State :=
                Compilation.Coded;
             Comments                : String := "";
             Work_Order              : String := "<DEFAULT>";
             Volume                  : Natural := 0;
             Response                : String := "<PROFILE>");
```





```
-- Create a new working view.   Working views are named Mumble_Working,
-- where mumble is supplied as New_Working_View.  If Join_Views is
-- true, the two views share reservations of the all of the controlled
-- objects in the two views.  If false, reservations aren't shared
-- across the views for any objects.  If From_View names multiple views, a
-- copy is made for each of those views and, if the originals
-- import each other (computed using the subsystem, not the view),
-- the copies will (try) to import the new views of those subsystems.

-- If Join_Views is false, new reservation tokens are created for all
-- of the controlled objects.  The default is to use the name supplied
-- as the >>SUBPATH_NAME<<.

-- View_To_Import supplies a set of views to be processed according to
-- the value of Only_Change_Imports.  If Only_Change_Imports is true,
-- a copied view always inherits the source view's imports.  After the
-- copy, the imports specified by View_To_Import are applied against the
-- new view, replacing any inherited import if needed.
-- If Only_Change_Imports is false, then either the imports are inherited
-- from the source, or the complete set of imports specified by
-- by View_To_Import is imported into the copy.

-- View_To_Modify specifies the set of working views that are to have
-- their imports changed to refer to the new copy(s).  The
-- View_To_Modify views are also changed to refer to the views specified
-- by View_To_Import.  For this import operation, Only_Change_Imports
-- is forced to true.

-- Construct_Subpath_Name cause Copy to contruct the target view name
-- by appending New_Working_View to the prefix of the source view name
-- up to the first '_' (See paths and subpaths below).

-- Remake demoted units, if true, indicates that ada units that were
-- demoted during the copy process are to be recompiled.  They are
-- compiled to the level indicated by Goal.

-- Goal further indicates the desired state of all of the units after
-- copy.  No unit will be in a state higher than specified by goal, but
-- might be in a lower state.  For example, a source unit that is copied
-- will remain source, regardless of Goal, but a Coded unit will be
-- demoted if Goal is installed or less.

-- The order of the copy and import operations is:
--
--      1.  Create the new view.
--      2.  If Inherit_Imports, bring along the old imports
--      3.  Import the new views into the new views, forcing
--          Only_Change_Imports => True
--      4.  If not Inherit_Imports, import the specified views
```

```
--          into the new views.
--       5. Import the new views + View_To_Import into Views_To_Modify,
--          forcing Only_Change_Imports => true

-- Spec views are created by copying the units if the source is a load
-- view, otherwise using Relocation.  Spec views are created with all
-- objects uncontrolled.  If level_for_spec_view = natural'last, the
-- spec view is given the name supplied as new_working_view, otherwise
-- a name is generated as 'New_Working_View & Release_Numbers & "_spec"'

-- In a spec_load subsystem, combined views can be created by setting
-- the create_combined_view parameter.  Combined views are useful in
-- spec_load subsystems when spec and load views are compiled for the
-- R1000 target and combined views must be compiled for a different
-- target that does not support subsystem look-through.

-- Note that if create_spec_view, create_load_view, and create_combined_view
-- are all false, then the new view has the same type as from_view.
-- It is an error to set more that one of these parameters to be true.

-- It is recognized that this is a complicated command.  Using the
-- procedures below (which are effectively renames) might make more
-- sense if the methodolody in use permits it (Path, Subpath, etc).


------------------------------------------------------------------------
--                       PATHS AND SUBPATHS                           --
------------------------------------------------------------------------

-- The following procedures support the notion of paths and subpaths.
-- A Path is a logically connected series of releases in which all
-- controlled objects are joined together.  In other words, there is
-- no branching within a path.  A Subpath is an extension of the
-- path, allowing multiple developers to make changes and test
-- without getting in each others way.  However, controlled objects
-- in the subpaths are joined with the path; people in two subpaths
-- cannot independently change the same object. In addition, a path
-- and its subpaths share the same model, which means they share
-- the same Target_Key and initial links.

-- In Delta, paths and subpaths are identified by string name conventions.
-- The name of the path is the view name up to the first '_'.  The
-- subpath extension is the name from this '_' to the '_Working'. Thus
-- Rev9_Cbh_Working has a path name of Rev9 and subpath extension of
-- Cbh.

-- Multiple paths are used when multiple targets are involved, or when
-- objects are to be changed independently.  For example, assume that
-- a version of a product has been shipped, and is in maintenance, and
```





```
-- that development is progessing on a new version.  It is likely that
-- the old and new versions would be separate paths, since the objects
-- would have to be independently changed (these paths would not be
-- 'joined').

-- In the multiple target case, the paths might be created joined.
-- Using the above scenario, assume that the release that has been shipped
-- works on two targets, but most or all of the code is target
-- independent.  Then the two paths, one for each target, would be
-- created joined together, then have the objects that are not common
-- 'Sever'ed.

procedure Make_Path
            (From_Path             : String := "<CURSOR>";
             New_Path_Name         : String := ">>PATH NAME<<";
             View_To_Modify        : String := "";
             View_To_Import        : String := "<INHERIT_IMPORTS>";
             Only_Change_Imports   : Boolean := True;
             Create_Load_View      : Boolean := False;
             Create_Combined_View  : Boolean := False;
             Model                 : String := "<INHERIT_MODEL>";
             Join_Paths            : Boolean := True;
             Remake_Demoted_Units  : Boolean := True;
             Goal                  : Compilation.Unit_State :=
                Compilation.Coded;
             Comments              : String := "";
             Work_Order            : String := "<DEFAULT>";
             Volume                : Natural := 0;
             Response              : String := "<PROFILE>");

procedure Make_Subpath
            (From_Path              : String := "<CURSOR>";
             New_Subpath_Extension  : String := ">>SUBPATH<<";
             View_To_Modify         : String := "";
             View_To_Import         : String := "<INHERIT_IMPORTS>";
             Only_Change_Imports    : Boolean := True;
             Remake_Demoted_Units   : Boolean := True;
             Goal                   : Compilation.Unit_State :=
                Compilation.Coded;
             Comments               : String := "";
             Work_Order             : String := "<DEFAULT>";
             Volume                 : Natural := 0;
             Response               : String := "<PROFILE>");

-- The Subpath_Extension is appended to the path name of the source
-- view (From_Path).  From_Path can actually name the path or any
-- subpath of the path.  The '_' between the path and subpath extension
-- is automatically provided.
```

```
procedure Make_Spec_View
            (From_Path             : String := "<CURSOR>";
             Spec_View_Prefix      : String := ">>PREFIX<<";
             Level                 : Natural := 1;
             View_To_Modify        : String := "";
             View_To_Import        : String := "<INHERIT_IMPORTS>";
             Only_Change_Imports   : Boolean := True;
             Remake_Demoted_Units  : Boolean := True;
             Goal                  : Compilation.Unit_State :=
                Compilation.Coded;
             Comments              : String := "";
             Work_Order            : String := "<DEFAULT>";
             Volume                : Natural := 0;
             Response              : String := "<PROFILE>");

-- Make a spec view for a path.  Spec_View_Prefix is the string that
-- replaces the path and subpath name.  For example, if creating a
-- spec view from a subpath named rev9_cbh_working, with
-- Spec_View_Prefix => Env9, the result will be Env9_n_Spec, assuming
-- level => 0 and two levels are specified by the model.  N is a
-- number automatically generated from the current release number for
-- the path/subpath.  If level = natural'last, the name supplied as
-- Spec_View_Prefix is used for the name of the view, with no suffixes

----------------------------------------------------------------------

procedure Import
            (View_To_Import       : String                  := "<REGION>";
             Into_View            : String                  := "<CURSOR>";
             Only_Change_Imports  : Boolean                 := False;
             Import_Closure       : Boolean                 := False;
             Remake_Demoted_Units : Boolean                 := True;
             Goal                 : Compilation.Unit_State :=
                Compilation.Coded;
             Comments             : String                  := "";
             Work_Order           : String                  := "<DEFAULT>";
             Response             : String                  := "<PROFILE>");

-- Imports spec or combined views as appropriate into the specified
-- view(s).  The import specification can be a set of view names,
-- in which case all views are imported, unless only_change_imports is
-- true.  In this case only subsystems that were imported sometime in
-- the past are reimported.  All others are ignored.

-- If View_To_Import is "", then the imports of Into_View are refreshed.
-- This means the various imported views are examined, and any new
-- Ada specs are imported in to the current view.
```





```
-- It is useful to invoke Import with Views_To_Import = Into_View and
-- Only_Change_Imports is true.  This will cause a set of views to be
-- changed to import each other.

procedure Remove_Import (View        : String := ">>VIEW NAME<<";
                         From_View   : String := "<CURSOR>";
                         Comments    : String := "";
                         Work_Order  : String := "<DEFAULT>";
                         Response    : String := "<PROFILE>");

-- remove references to a previously imported view.

procedure Remove_Unused_Imports (From_View  : String := "<CURSOR>";
                                 Comments   : String := "";
                                 Work_Order : String := "<DEFAULT>";
                                 Response   : String := "<PROFILE>");

-- Search through all of the Ada units in the view and examine the
-- withs.  If no units in some imported view are referenced, remove
-- that import.

-- This command generates warnings if units in spec or combined
-- views are referenced, but the view isn't imported.  Errors are
-- generated if units in load views are referenced.

procedure Replace_Model (New_Model  : String := ">>NEW MODEL NAME<<";
                         In_View    : String := "<CURSOR>";
                         Comments   : String := "";
                         Work_Order : String := "<DEFAULT>";
                         Response   : String := "<PROFILE>");

-- Replace the model with the new one.  All units must be source.
-- This command gets the switch file from the new model (if one
-- was provided), readjusts the maximum levels (which affects future
-- releases), and rebuilds the links.

------------------------------------------------------------------------
--                          SYSTEM OBJECTS                            --
------------------------------------------------------------------------

type System_Object_Enum is (Spec_Load_Subsystem,
                            Combined_Subsystem, System);

-- System objects may be either subsystems or systems.  A subsystem
-- may be either a spec_load_subsystem or a combined_subsystem.
```

```
-- The type of subsystem controls the kinds of views which
-- the system object may contain.  The subsystem type also controls
-- whether importing into the subsystem must be hierarchical or
-- may be non-hierarchical.

-- Spec_Load subsystems may contain spec views, load views, or
-- combined views.  All views in spec_load subsystems are
-- restricted to have only hierarchical imports.  A views imports
-- are hierarchical if the import closure of the view does not
-- contain itself.

-- Combined subsystems may only contain combined views.  The views
-- in a combined subsystem need not have hierarchical imports.
-- A view in a combined subsystem may include itself in its
-- import closure.

-- Systems contain system views.  Systems are used by Cmvc_Hierarchy
-- to coordinate the construction of multi-subsystem systems.

procedure Initial
             (System_Object          : String := ">>SYSTEM OBJECT NAME<<";
              Working_View_Base_Name : String := "Rev1";
              System_Object_Type     : System_Object_Enum :=
                 Cmvc.Spec_Load_Subsystem;
              View_To_Import         : String := "";
              Create_Load_View       : Boolean := True;
              Model                  : String := "R1000";
              Comments               : String := "";
              Work_Order             : String := "<DEFAULT>";
              Volume*                : Natural := 0;
              Response               : String := "<PROFILE>");

-- Build a new system object of the specified type.  Also create a working
-- view and import as specified.  This command can be used to create
-- an empty view in an existing system object.  If the system object type
-- is spec_load_subsystem the new view will be either a load view or a
-- combined view depending on the value of create_load_view.
-- While creating the new view directory structure in the model world
-- will be duplicated in the new view.

------------------------------------------------------------------------

procedure Information (For_View                  : String  := "<CURSOR>";
                       Show_Model                : Boolean := True;
                       Show_Whether_Frozen       : Boolean := True;
                       Show_View_Kind            : Boolean := True;
                       Show_Creation_Time        : Boolean := True;
                       Show_Imports              : Boolean := True;
```





```
                       Show_Referencers          : Boolean := True;
                       Show_Unit_Summary         : Boolean := True;
                       Show_Controlled_Objects   : Boolean := False;
                       Show_Last_Release_Numbers : Boolean := False;
                       Show_Path_Name            : Boolean := False;
                       Show_Subpath_Name         : Boolean := False;
                       Show_Switches             : Boolean := False;
                       Show_Exported_Units       : Boolean := False;
                       Response                  : String  := "<PROFILE>");

-- Show various things about a view.  Please see Cmvc_History for
-- ways of extracting other information about the controlled objects
-- in the view.

------------------------------------------------------------------------

procedure Destroy_View
             (What_View                    : String  := "<SELECTION>";
              Demote_Clients               : Boolean := False;
              Destroy_Configuration_Also   : Boolean := False;
              Comments                     : String  := "";
              Work_Order                   : String  := "<DEFAULT>";
              Response                     : String  := "<PROFILE>");

-- Destroy a view.  If Demote_Clients is false, the view can have no
-- referencing views (clients); if it does, the destroy fails.  If
-- Demote_Clients is true, the view is "remove_import"ed from those
-- clients (which might cause lots of obsolescence), then the view is
-- destroyed.  The configuration object for the view is left behind
-- in its normal place (see Release, above) so the view can be
-- reconstructed using "Build"

procedure Destroy_Subsystem (What_Subsystem : String := "<SELECTION>";
                             Comments       : String := "";
                             Work_Order     : String := "<DEFAULT>";
                             Response       : String := "<PROFILE>");

-- Destroy a subsystem.  There must be no views in the subsystem.

procedure Destroy_System (What_System : String := "<SELECTION>";
                          Comments    : String := "";
                          Work_Order  : String := "<DEFAULT>";
                          Response    : String := "<PROFILE>");

-- Destroy a system.  There must be no views in the system.

------------------------------------------------------------------------
```

```
procedure Build
             (Configuration  : String := ">>CONFIGURATION NAME<<";
              View_To_Import : String := "<INHERIT_IMPORTS>";
              Model          : String := "<INHERIT_MODEL>";
              Goal           : Compilation.Unit_State :=
                 Compilation.Installed;
              Limit          : String := "<WORLDS>";
              Comments       : String := "";
              Work_Order     : String := "<DEFAULT>";
              Volume         : Natural := 0;
              Response       : String := "<PROFILE>");

-- Rebuild a view from history.  If Configuration_Object_Name refers to
-- a text file, that file is assumed to contain a list of configuration
-- object names to be built.

-- If View_To_Import = "<INHERIT_IMPORTS>", and if a directory with
-- the name "same as configuration_object" & "_STATE" exists, that
-- directory contains state that is used to rebuild the imports.
-- Note, that the state directory is created by the release command
-- when a configuration-only release is created.

---------------------------------------------------------------------
--                     HISTORY COMMANDS                            --
---------------------------------------------------------------------

-- The following commands display history information, in various
-- formats, of Cmvc controlled objects

procedure Show_History (For_Objects            : String  := "<CURSOR>";
                        Display_Change_Regions : Boolean := True;
                        Starting_Generation    : String  := "<CURSOR>";
                        Ending_Generation      : String  := "";
                        Response               : String  := "<PROFILE>");
-- Display the history for the specified objects.  If a view is
-- specified, all of the controlled objects in that view are displayed.
-- This history includes notes, checked_out and _in information, and
-- optionally the actual changes

-- If display_change_regions is true, the differences between a
-- generation and the previous one (n-1, n) are displayed.  The display
-- is in the form of regions where changes occurred similar to that
-- produced by File_Utilities.Difference(Compressed_Output=>True)

-- The first generation to display is determined by looking up
-- the object in the view(s) specified by Starting_Generation.  If
-- Starting_Generation = "", the display starts at generation 1.
```





```
-- The last generation to display is determined by Ending_Generation.
-- If E.._G.. is "", the last displayed is the latest one.  If E.._G..
-- is the name of a view, the generation specified by that view is
-- used as the last.

procedure Show_History_By_Generation
             (For_Objects            : String  := "<CURSOR>";
              Display_Change_Regions : Boolean := True;
              Starting_Generation    : Natural := 1;
              Ending_Generation      : Natural := Natural'Last;
              Response               : String  := "<PROFILE>");

-- A form of show_history_by_generation that takes explicit
-- generation numbers.

procedure Show_Image_Of_Generation (Object         : String := "<CURSOR>";
                                    Generation     : Integer := -1;
                                    Output_Goes_To : String := "<WINDOW>";
                                    Response       : String := "<PROFILE>");
-- Reconstruct an image of some generation of the specified object.
-- The default (-1) indicates back up one generation from that of
-- Object.  Negative numbers are relative to the generation of Object,
-- positive numbers are actual generation numbers.
-- The result is written to current output unless a file name is
-- supplied in Output_Goes_To.

---------------------------------------------------------------------

-- The following commands produce a report showing objects that
-- meet some criteria.  This report shows the following information
-- about each object.

-- Obj Name  Gen    Where Chkd Out Who Expect Check In  Source Saved
-- ========= ====== ===== ======== === ===============  ============
-- UNITS.FOO 5 of 8 VIEW  Yes      MTD Apr 7, 1987      Yes

-- Object name is the element name (the name from the view down)

-- Generation is a pair.  The first number is the generation of
-- the object used to lookup the element.  The second number is
-- the highest generation produced.

-- Where is either the view containing a copy of the last generation
-- if the object is not checked out, or the view in which the object
-- is checked out.  In the case where the object is not checked out,
-- it is possible that there is no representative object, in which
-- case this field is blank.

-- Chkd Out is 'Checked Out'.  If this is yes, 'By Who' and
```

```
-- 'Expected Check In' provide more information.

-- "Source Saved" tells whether or not source is being saved in the
-- cmvc database for this object.

---------------------------------------------------------------------

procedure Show (Objects  : String := "<CURSOR>";
                Response : String := "<PROFILE>");

-- Produce the information desribed above for the listed objects.
-- Also produces a report for each object showing which views
-- contain elements sharing a reservation token with the object.

procedure Show_All_Checked_Out (In_View  : String := "<CURSOR>";
                                Response : String := "<PROFILE>");

-- Look through all of the controlled objects in the supplied view, and
-- display information about them if they are checked out anywhere

procedure Show_Checked_Out_In_View (In_View  : String := "<CURSOR>";
                                    Response : String := "<PROFILE>");

-- Display information about all of the objects checked out in the
-- view pointed at (or in)

procedure Show_Checked_Out_By_User
             (In_View  : String := "<CURSOR>";
              Who      : String := System_Utilities.User_Name;
              Response : String := "<PROFILE>");

-- Display information about any object in the view that is checked out
-- be the user given.  This command will find the object even if it is
-- checked out in some other view, as long as it is controlled in the
-- view referred to.

procedure Show_Out_Of_Date_Objects (In_View  : String := "<CURSOR>";
                                    Response : String := "<PROFILE>");

-- Display information about all objects in the view that are not
-- at the latest revision.

procedure Show_All_Uncontrolled (In_View  : String := "<CURSOR>";
                                 Response : String := "<PROFILE>");

-- Show all uncontrolled objects in the designated view.

procedure Show_All_Controlled (In_View  : String := "<CURSOR>";
                               Response : String := "<PROFILE>");
```





```
-- Display information about all controlled objects in this view


--------------------------------------------------------------------
--                        ARCHIVE COMMANDS                        --
--------------------------------------------------------------------

procedure Make_Code_View (From_View      : String  := "<CURSOR>";
                          Code_View_Name : String  := "";
                          Comments       : String  := "";
                          Work_Order     : String  := "<DEFAULT>";
                          Volume         : Natural := 0;
                          Response       : String  := "<PROFILE>");

-- Make a code view with the given name.  From_View must only
-- name load views.  The result is a load view containing an
-- object called "Code_Database" which contains the executable code for
-- the units in the from_view.  There are no ada units in
-- the units directory of the resulting view.
-- This operation fails if any unit isn't coded, or any spec exists
-- for which a body is required and one doesn't exist.


--------------------------------------------------------------------
--                     OBJECT EDITOR COMMANDS                     --
--------------------------------------------------------------------


procedure Edit (View_Or_Config        : String  := "<CURSOR>";
                In_Place              : Boolean := False;
                Allow_Check_Out       : Boolean := True;
                Allow_Check_In        : Boolean := True;
                Allow_Accept_Changes  : Boolean := True);

-- Brings up the cmvc object editor on the configuration or the
-- configuration associated with the view.  The view parameter may be a
-- view itself or any object within a view.

-- The cmvc object editor display reservation state, object history,
-- and the notes associated with objects.

-- The parameters allow_check_out, allow_check_in, and allow_accept_changes
-- control whether or not these operations can be performed as common
-- commands in the object editor.

procedure Notes (What_Object : String  := "<CURSOR>";
                 In_Place    : Boolean := False);

-- Brings up the notes for the generation associated with the
```

```
-- specified controlled object.  New notes may be appended.

procedure Def (What_Object : String  := "<CURSOR>";
               In_Place    : Boolean := False);

-- Used to go between images in the object editor and
-- objects in the directory system.
-- When applied to an image in the cmvc object editor, tries
-- to find the associated object in the directory system.
-- When applied to a directory object, the cmvc image for that
-- object is produced.


------------------------------------------------------------------------
--                       VIEW COMPARISON COMMANDS                     --
------------------------------------------------------------------------

procedure Compare (Destination          : String  := "<CURSOR>";
                   Source               : String  := "<REGION>";
                   Compare_Both         : Boolean := True;
                   Show_New_Uncontrolled : Boolean := True;
                   Show_New_Controlled  : Boolean := True;
                   Show_Uncontrolled    : Boolean := True;
                   Show_Severed         : Boolean := True;
                   Show_Modified        : Boolean := True;
                   Show_Equal           : Boolean := False;
                   Ada_Units            : Boolean := True;
                   Files                : Boolean := True;
                   Response             : String  := "<PROFILE>");

-- Compare the views.  Destination and Source may be each be
-- a view or a configuration.  The Ada_Units and Files
-- parameters determine which type of objects are considered in the
-- comparison.  The other parameters determine the differences that
-- are displayed and are dependent on the comparison mode.
-- Compare_Both determines whether the views are compared symetrically
-- or whether just the Source is compared against the Destination.
-- When Compare_Both is false the information is useful in determining
-- the effect of Cmvc.Accept_Changes from the Source to the Destination.
--
-- Compare_Both = True
-- Show_New_Uncontrolled - Show any objects that are uncontrolled in one
--                         view but do not exist in the other.
-- Show_New_Controlled - Show any objects that are controlled in one
--                       view but do not exist in the other.
-- Show_Uncontrolled - Show any objects that are uncontrolled in one view
--                     but which do exist in the other view (either
--                     controlled or uncontrolled).
-- Show_Severed - Show any objects that exist in both views,
```





```
--                but are not joined.  This includes objects that
--                are controlled in one and uncontrolled in the other.
-- Show_Modified - Show objects that exist and are joined in both views
--                 but are not the same generation.
-- Show_Equal - Show the objects that exist in both views are controlled
--              and joined and have equal generations.
--
-- Compare_Both = False
-- Show_New_Uncontrolled - Show objects in the Source that are uncontrolled
--                         and do not exist in the destination.
-- Show_New_Controlled - Show objects in the Source that are controlled
--                       and do not exist in the Destination.
-- Show_Uncontrolled - Show objects in the Source that are uncontrolled
--                     but do exist in the Destination.
-- Show_Severed - Show objects in the Source view or configuration which
--                also exist in the Destination but are not joined or
--                possibly not controlled.
-- Show_Modified - Show objects in the Source (view or configuration)
--                 and in the Destination, which are joined but later
--                 in the source in the Destination.  This may also
--                 include objects checked out in the source.
-- Show_Equal - Show objects in the source view or configuration
--              that also exist with the same generation in the
--              Destination.
--
-- The default parameter settings will display all objects in either
-- view that have different characteristics than the corresponding
-- object in the other view.

procedure Accept_Changes_Effort
             (Destination           : String  := "<CURSOR>";
              Source                : String  := "<REGION>";
              Compare_Both          : Boolean := False;
              Show_New_Uncontrolled : Boolean := False;
              Show_New_Controlled   : Boolean := True;
              Show_Uncontrolled     : Boolean := False;
              Show_Severed          : Boolean := False;
              Show_Modified         : Boolean := True;
              Show_Equal            : Boolean := False;
              Ada_Units             : Boolean := True;
              Files                 : Boolean := True;
              Response              : String  := "<PROFILE>")
    renames Compare;

-- The default parameter settings will display the effect of
-- Cmvc.Accept_Changes from the Source into the destination.

end Cmvc;
```

```
package Cmvc_Access_Control is

    -- Control over access to objects in views and subsystems,
    -- and control over the execution of Cmvc and related commands
    -- when they reference specific views or subsystems.
    --
    -- "Access classes" define the kind of access that a group
    -- may have to the objects in a view or subsystem.
    --
    -- "Execution rights" determine the Cmvc and related commands
    -- that a group may execute within a view or subsystem.
    -- Each execution right requires some mimimum access to all
    -- referenced views and subsystems.


    ------------------------------------------------------------------------
    --                          Access Classes                            --
    ------------------------------------------------------------------------


    type Access_Class is (Reader, Client, Developer, Owner);

    -- The kind of access that groups may have to subsystems or views.
    -- The access class of a group determines the settings of the
    -- ACLs for the objects within the subsystem or view.
    -- A group may have only one kind of access at a given time.
    -- Higher access classes imply all the rights of lower access classes.


    procedure Add_Group (The_Group             : String := "NETWORK_PUBLIC";
                         In_Class              : Access_Class :=
                            Cmvc_Access_Control.Reader;
                         View_Or_Subsystem     : String := "<SELECTION>";
                         Add_Execution_Rights  : Boolean := True;
                         Response              : String := "<PROFILE>");

    -- Add the group to the list of groups that has access to the
    -- designated view or subsystem.  If the group is already on
    -- the list then change the access to the designated value.
    -- The ACLs for objects in the view or subsystem are
    -- adjusted appropriately.  If Add_Execution_Rights is set then
    -- the group is also granted all execution rights that are
    -- appropriate to the specified access class.  Whether or not
    -- Add_Execution_Rights is set, all inappropriate execution rights
    -- are removed when the access class for a group is changed.
    -- There is a limit of 7 groups that may have access to a subsystem
    -- or view at any one time.
```





```
    procedure Remove_Group (The_Group         : String := "<ALL>";
                            View_Or_Subsystem : String := "<SELECTION>";
                            Response          : String := "<PROFILE>");

    -- Clear the access for the group from the specified view or subsystem.
    -- If the group is <ALL> then access is restricted for all groups
    -- with current access.  All execution rights are also cleared.


    procedure Display (For_Group         : String  := "<ALL>";
                       View_Or_Subsystem : String  := "<CURSOR>";
                       Execution_Rights  : Boolean := False;
                       Response          : String  := "<PROFILE>");

    -- Display the current access control information for the specified
    -- group.  The symbol <ALL> specifies that all groups with access
    -- are to be displayed.  Execution rights are displayed if requested.


    function Has_Access (User_Or_Group     : String := "<USER>";
                         In_Class          : Access_Class :=
                            Cmvc_Access_Control.Reader;
                         View_Or_Subsystem : String := "<CURSOR>";
                         Group_Only        : Boolean := False;
                         Response          : String := "<WARN>") return Boolean;

    -- Determine if the user or group has the specified access to the
    -- view or subsystem.  If a user is named then all groups which
    -- include the user are checked for the access.  If a group is specified
    -- then the group is checked for the specified access.  If Group_Only
    -- is specified then, group access is checked even if there exists
    -- a user with the same name.  The special symbol <USER> denotes
    -- the current user.


    ------------------------------------------------------------------------
    --                         Execution Rights                           --
    ------------------------------------------------------------------------


    type Execution_Right is new Natural range 0 .. 127;

    -- Each command in Cmvc, Cmvc_Maintenance, or Cmvc_Hierarchy is
    -- composed of "primitive operations" that correspond to the part
    -- of the command that is applied to the different parameters.
    -- An execution right is the capability to execute a primitive
    -- operation on a view or a subsystem.  There is an execution right
    -- for each primitive operation.  When a command is executed
    -- the execution rights are checked for each primitive operation
```

```
-- involved in the command.
--
-- For example, when Cmvc.Accept_Changes is executed to move
-- changes from one view to another, the execution right
-- "Accept_Changes_Source" is checked on the source view and
-- the execution right "Accept_Changes_Destination" is checked on
-- the destination view.
--
-- Below are listed the various execution rights.  The name of
-- each execution right is formed from  the command it is primarily
-- associated with, as the first part of the name, and the
-- parameter (if more than one) as the second part of the name.


-- All appropriate rights, depending on the context
All_Rights : constant Execution_Right := 0;

-- Cmvc operations for views.
Check_Out                  : constant Execution_Right := 1;
Check_In                   : constant Execution_Right := 2;
Accept_Changes_Destination : constant Execution_Right := 3;
Accept_Changes_Source      : constant Execution_Right := 4;
Abandon_Reservation        : constant Execution_Right := 5;
Revert                     : constant Execution_Right := 6;
Modify_Notes               : constant Execution_Right := 7;
Make_Controlled            : constant Execution_Right := 8;
Make_Uncontrolled          : constant Execution_Right := 9;
Sever                      : constant Execution_Right := 10;
Join_What                  : constant Execution_Right := 11;
Join_To                    : constant Execution_Right := 12;
Merge_Changes_Destination  : constant Execution_Right := 13;
Merge_Changes_Source       : constant Execution_Right := 14;
Release                    : constant Execution_Right := 15;
Copy                       : constant Execution_Right := 16;
Make_Path                  : constant Execution_Right := 17;
Make_Subpath               : constant Execution_Right := 18;
Make_Spec_View             : constant Execution_Right := 19;
Import_From                : constant Execution_Right := 20;
Import_Into                : constant Execution_Right := 21;
Remove_Import              : constant Execution_Right := 22;
Replace_Model              : constant Execution_Right := 23;
Destroy_View               : constant Execution_Right := 24;
Make_Code_View             : constant Execution_Right := 25;
Query_View                 : constant Execution_Right := 26;

-- Cmvc_Maintenance operations for views.
Check_Consistency : constant Execution_Right := 27;

-- Cmvc_Hierarchy operations for views
```





```
Build_Activity_In   : constant Execution_Right := 28;
Build_Activity_From : constant Execution_Right := 29;
Expand_Activity     : constant Execution_Right := 30;

-- Cmvc operations for subsystem.
Initial           : constant Execution_Right := 31;
Destroy_Config    : constant Execution_Right := 32;
Destroy_Subsystem : constant Execution_Right := 33;
Build             : constant Execution_Right := 34;
Query_Subsystem   : constant Execution_Right := 35;
Edit_Notes        : constant Execution_Right := 36;

-- Cmvc_Maintenance operations for subsystems.
Expunge_Database            : constant Execution_Right := 37;
Subsystem_Check_Consistency : constant Execution_Right := 38;
Update_Cdb                  : constant Execution_Right := 39;
Make_Primary                : constant Execution_Right := 40;
Make_Secondary              : constant Execution_Right := 41;
Destroy_Cdb                 : constant Execution_Right := 42;

-- Cmvc_Hierarchy operations for systems and subsystems
Add_Child_Parent : constant Execution_Right := 43;
Add_Child_Child  : constant Execution_Right := 44;
Remove_Child     : constant Execution_Right := 45;


procedure Add_Right (For_Group         : String := "NETWORK_PUBLIC";
                     The_Right         : Execution_Right :=
                        Cmvc_Access_Control.All_Rights;
                     View_Or_Subsystem : String := "<SELECTION>";
                     Response          : String := "<PROFILE>");

-- Add the execution right for the group in the designated subsystem.
-- An execution right can only be added if it is appropriate to
-- the current access class for that group in the view or subsystem.
-- If the right is All_Rights then all rights which appropriate to
-- the group's access class are added.


procedure Remove_Right (For_Group         : String := "<ALL>";
                        The_Right         : Execution_Right :=
                           Cmvc_Access_Control.All_Rights;
                        View_Or_Subsystem : String := "<SELECTION>";
                        Response          : String := "<PROFILE>");

-- Remove the designated execution right.  If <ALL> is specified
-- for the group then the execution right is removed for all
-- groups with current rights.
```

```
function Has_Right (User_Or_Group     : String := "<USER>";
                    The_Right         : Execution_Right :=
                       Cmvc_Access_Control.All_Rights;
                    View_Or_Subsystem : String := "<CURSOR>";
                    Group_Only        : Boolean := False;
                    Response          : String := "<WARN>") return Boolean;

-- Determine if the user or group has the specifed execution right.
-- If The_Right is All_Rights or if Group_Only is set then the
-- name is interpreted as a group name.  Also, if The_Right is All_Rights
-- then true is returned only if all rights appropriate to the current
-- access class are set.


------------------------------------------------------------------------
--                    Miscellaneous Operations                        --
------------------------------------------------------------------------

procedure Check (View_Or_Subsystem      : String  := "<SELECTION>";
                 Repair_Inconsistencies : Boolean := False;
                 Response               : String  := "<PROFILE>");

-- Check that the current access classes for the view or subsystem
-- are compatible with ACLs on the objects in the views and/or subsystems.
-- If the ACLs on subobjects are not compatible with the access classes
-- and the current user has owner access to the view or subsystem and
-- Repair_Inconsistencies is set then the ACLs on the subobjects
-- will be reset.
--
-- Incompatible ACLs are either 1) ACL entries lacking access indicated
-- by the access of the view or subsystem, or 2) ACL entries for groups
-- without access to the view or subsystem.
-- Repair_Inconsistencies will repair both types.  If Repair_Inconsistencies
-- is false then an error occurs if groups having access to the
-- view or subsystem are not correctly represented on an ACL lists.
-- A warning is issued if groups not having access to the view
-- or subsystem are found to be on the ACL of a subobject.
--
-- This procedure will also check that there are no entries for
-- groups that have been deleted.  If there is such an entry and
-- Repair_Inconsistencies is false then an error occurs.  If there
-- is an entry for a delete group and Repair_Inconsistencies is true
-- then the deleted entry will be removed from all ACL lists as
-- specified.  This is the only way in which obsolete entries can
-- be deleted from access control state.


function Is_Consistent (View_Or_Subsystem : String := "<CURSOR>";
                        Response : String := "<WARN>") return Boolean;
```





```
-- Perform same consistency check as procedure above, does not
-- put out any warnings nor will it make any changes.  Will also return
-- false if access control information does not exist or if the
-- current user does not have sufficient access to determine if
-- the information is consistent.

procedure Initialize (View_Or_Subsystem : String := "<SELECTION>";
                      Response          : String := "<PROFILE>");

-- Initialize the access control information for a view or subsystem.
-- All groups that have owner access in the ACL of the view or
-- subsystem world are put into the owner access class.  All groups
-- that have read access in the ACL, but not owner access, are
-- put into the reader access class.  All appropriate execution rights
-- are set.
--
-- This operation is primarily useful for setting up access control
-- in subsystems or views that were created by previous environment
-- releases.  It can also be applied to views or subsystems with
-- access control information in which case the view or subsystem
-- is reset to its just initialized state with only owners and
-- readers having access.
--
-- All commands in this package will fail (unless otherwise noted)
-- if access control information is not initialized or if the current
-- user does not have read access to the view or subsystem the
-- command is being applied to.


No_Access : exception;
-- Raised by Get_Access and Get_Rights if the group has no access.

function Get_Access (The_Group         : String := "NETWORK_PUBLIC";
                     View_Or_Subsystem : String := "<CURSOR>";
                     Response          : String := "<WARN>")
                    return Access_Class;

-- Return the access class of a group with access to the view or
-- subsystem.  If the group has no access then No_Access is raised.

type Group_Index is range 0 .. 6;
-- Index of a group that has access to a view or subsystem.  Since,
-- there can be at most 7 groups with access to a view or subsystem
-- the groups are indexed from 0 to 6.


function Group_Name (The_Index         : Group_Index := 0;
                     View_Or_Subsystem : String := "<CURSOR>";
                     Response          : String := "<WARN>") return String;
```

```
-- Get the name of the group at the specified index.  If no group
-- is at the index then returns "".  Note that if index N is empty
-- then index N+1 is also empty.


-----------------------------------------------------------------------
--                       Execution Right Tables                     --
-----------------------------------------------------------------------

-- In many instances, especially in building additional tools, it will
-- be useful to treat execution rights as a composite object.  An
-- "execution table" is a table of all execution rights.  The execution
-- rights for a particular group can set by providing a complete
-- execution table.  Constant execution tables, corresponding to
-- rights appropriate for various access classes, are also provided.


type Execution_Table is array (Execution_Right) of Boolean;
-- Table of all possible execution rights.


Nil_Rights : constant Execution_Table := (others => False);
-- Setting execution rights to this prevents execution of any commands.


procedure Set_Rights (For_Group          : String := "NETWORK_PUBLIC";
                      The_Rights         : Execution_Table :=
                         Cmvc_Access_Control.Nil_Rights;
                      View_Or_Subsystem : String := "<SELECTION>";
                      Response           : String := "<PROFILE>");

-- Set all the execution rights for the specified group in the view
-- or subsystem.  Only rights appropriate to the groups access class
-- are actually set.  Warnings are produced for inappropriate rights.


function Get_Rights (For_Group : String := "NETWORK_PUBLIC";
                     View_Or_Subsystem : String := "<CURSOR>";
                     Response : String := "<WARN>") return Execution_Table;

-- Return the execution rights for a group.  If the group does not
-- have access to the view or subsystem then No_Access is raised.


-----------------------------------------------------------------------
--                  Constant Execution Right Tables                 --
-----------------------------------------------------------------------
```





```
-- Below are constants describing the execution rights that appropriate
-- to various access classes.
-- Rights for a view that only require READER access to the view
View_Reader_Rights : constant Execution_Table :=
   Execution_Table'(Accept_Changes_Source => True,
                    Join_To               => True,
                    Merge_Changes_Source  => True,
                    Query_View            => True,
                    Expand_Activity       => True,
                    others                => False);

-- Rights for a view that only require CLIENT access to the view
View_Client_Rights : constant Execution_Table :=
   Execution_Table'
      (Import_From => True, Build_Activity_From => True, others => False)
    or View_Reader_Rights;

-- Rights for a view that only require DEVELOPER access to the view
View_Developer_Rights : constant Execution_Table :=
   Execution_Table'(Check_Out                  => True,
                    Check_In                   => True,
                    Accept_Changes_Destination => True,
                    Abandon_Reservation        => True,
                    Revert                     => True,
                    Modify_Notes               => True,
                    Make_Controlled            => True,
                    Make_Uncontrolled          => True,
                    Sever                      => True,
                    Join_What                  => True,
                    Release                    => True,
                    Make_Path                  => True,
                    Make_Subpath               => True,
                    Make_Code_View             => True,
                    Copy                       => True,
                    Make_Spec_View             => True,
                    Merge_Changes_Destination  => True,
                    Build_Activity_In          => True,
                    others                     => False)
    or View_Client_Rights;

-- Rights for a view that require OWNER access to the view
View_Owner_Rights : constant Execution_Table :=
   Execution_Table'(Import_Into       => True,
                    Remove_Import     => True,
                    Replace_Model     => True,
                    Destroy_View      => True,
                    Check_Consistency => True,
                    others            => False)
    or View_Developer_Rights;
```

```
-- Rights for a subsystem that only require READER access to the subsystem
Subsystem_Reader_Rights : constant Execution_Table :=
   Execution_Table'(Query_Subsystem => True, others => False);

-- Rights for a subsystem that only require CLIENT access to the subsystem
Subsystem_Client_Rights : constant Execution_Table :=
   Execution_Table'(Add_Child_Child => True, others => False) or
      Subsystem_Reader_Rights;

-- Rights for a subsystem that only require DEVELOPER access to the subsystem
Subsystem_Developer_Rights : constant Execution_Table :=
   Execution_Table'(Edit_Notes       => True,
                    Add_Child_Parent => True,
                    Remove_Child     => True,
                    Update_Cdb       => True,
                    others           => False)
    or Subsystem_Client_Rights;

-- Rights for a subsystem that require OWNER access to the subsystem
Subsystem_Owner_Rights : constant Execution_Table :=
   Execution_Table'(Initial                     => True,
                    Destroy_Config              => True,
                    Destroy_Subsystem           => True,
                    Build                       => True,
                    Subsystem_Check_Consistency => True,
                    Expunge_Database            => True,
                    Make_Primary                => True,
                    Make_Secondary              => True,
                    Destroy_Cdb                 => True,
                    others                      => False)
    or Subsystem_Developer_Rights;

-- Rights for a view that only require READER access
-- to the enclosing subsystem

Subsystem_Reader_View_Rights : constant Execution_Table :=
   Execution_Table'(Import_From         => True,
                    Import_Into         => True,
                    Remove_Import       => True,
                    Replace_Model       => True,
                    Query_View          => True,
                    Build_Activity_In   => True,
                    Build_Activity_From => True,
                    Expand_Activity     => True,
                    others              => False);


-- Rights for a view that only require CLIENT access
```





```
-- to the enclosing subsystem
Subsystem_Client_View_Rights : constant Execution_Table :=
   Subsystem_Reader_View_Rights;

-- Rights for a view that only require DEVELOPER access
-- to the enclosing subsystem
Subsystem_Developer_View_Rights : constant Execution_Table :=
   Execution_Table'(Check_Out                  => True,
                    Check_In                   => True,
                    Accept_Changes_Source      => True,
                    Accept_Changes_Destination => True,
                    Abandon_Reservation        => True,
                    Revert                     => True,
                    Modify_Notes               => True,
                    Make_Controlled            => True,
                    Make_Uncontrolled          => True,
                    Sever                      => True,
                    Join_What                  => True,
                    Join_To                    => True,
                    Merge_Changes_Destination  => True,
                    Merge_Changes_Source       => True,
                    others                     => False)
    or Subsystem_Client_View_Rights;

-- Rights for a view that require OWNER access
-- to the enclosing subsystem
Subsystem_Owner_View_Rights : constant Execution_Table :=
   Execution_Table'(Release           => True,
                    Copy              => True,
                    Make_Path         => True,
                    Make_Subpath      => True,
                    Make_Spec_View    => True,
                    Make_Code_View    => True,
                    Destroy_View      => True,
                    Check_Consistency => True,
                    others            => False)
    or Subsystem_Developer_View_Rights;

end Cmvc_Access_Control;
```

```
package Cmvc_Hierarchy is

    -- Operations to manipulate Cmvc "Systems".  Each system has some
    -- number of children which may be either subsystems or other
    -- systems.

    -- A system contains "system views".  Each system view contains
    -- an activity called the "release activity" which selects a
    -- released view for each child subsystem of the system.
    -- Thus, a system view describes a complete implementation of the
    -- entire system in much the same way that a load view contains a
    -- complete implementation of a subsystem.

    -- Systems and system views are created by commands in the Cmvc.
    -- Releasing a system view creates a new released system view.


    procedure Add_Child (Child      : String := ">>SYSTEM/SUBSYSTEM NAME<<";
                         To_System  : String := "<CURSOR>";
                         Comments   : String := "";
                         Work_Order : String := "<DEFAULT>";
                         Response   : String := "<PROFILE>");

    -- This operation creates a structural link between the designated
    -- system and child.
    -- These links may not form cycles through child systems.


    procedure Remove_Child (Child       : String := ">>SYSTEM/SUBSYSTEM NAME<<";
                            From_System : String := "<CURSOR>";
                            Comments    : String := "";
                            Work_Order  : String := "<DEFAULT>";
                            Response    : String := "<PROFILE>");

    -- The opposite of the operation above, this call will sever the
    -- connection between a child and a parent.


    procedure Build_Activity (Working_System_View : String  := "<CURSOR>";
                              Views_To_Include    : String  := "<LATEST>";
                              Update_Imports      : Boolean := True;
                              Allow_Code_Views    : Boolean := False;
                              Comments            : String  := "";
                              Work_Order          : String  := "<DEFAULT>";
                              Response            : String  := "<PROFILE>");

    -- Builds or updates the "release activity" in the working system view
```





```
    -- to include the specified views.  If <LATEST> is specified then
    -- the latest releases of all the children of the system are included.
    -- Spec views or releases are included in the release activity if
    -- the views have been created after the last time that build_activity
    -- was run on the specified working system view.
    -- Path restrictions may be used to control which releases are included.

    -- If Update_Imports is set then the system view is made to import
    -- all of the subsystem views referenced by the release activity.
    -- Note that this importing is subject to the normal compatibility
    -- requirements.

    -- If Allow_Code_Views is true then code-only views are treated like
    -- release in that they can be included in the release activity.
    -- Otherwise, only regular released views will be included in the
    -- release activity.


    procedure Expand_Activity (New_Activity : String := ">>NEW ACTIVITY NAME<<";
                               System_View  : String := "<CURSOR>";
                               Response     : String := "<PROFILE>");

    -- Make a copy of a release activity which does not contain any
    -- release views - replace all references to releases with the
    -- contents of that release.


    function Contents (Of_System_View : String  := "<CURSOR>";
                       Recursive      : Boolean := True;
                       Response       : String  := "<WARN>") return String;

    -- Returns the contents of the release_activity of the system view.


    function Children (Of_System : String  := "<CURSOR>";
                       Recursive : Boolean := True;
                       Response  : String  := "<WARN>") return String;

    -- This subprogram returns a list of the children of a system.


    function Parents (Of_Subsystem : String  := "<CURSOR>";
                      Recursive    : Boolean := False;
                      Response     : String  := "<WARN>") return String;

    -- This returns a list of parent Systems.

end Cmvc_Hierarchy;
```

```
package Cmvc_Maintenance is

    procedure Expunge_Database (In_Subsystem : String := "<CURSOR>";
                                Response     : String := "<PROFILE>");

    -- Free up space in the Database by first finding all configurations
    -- in the database that no longer have objects and destroying them,
    -- then destroying all elements and join sets (with all of their
    -- generations) that are no longer referenced.

    procedure Delete_Unreferenced_Leading_Generations
                 (In_Subsystem : String := "<CURSOR>";
                  Response     : String := "<PROFILE>");

    -- Not yet implemented

    procedure Convert_Old_Subsystem (Which    : String := "<SELECTION>";
                                     Response : String := "<PROFILE>");

    -- Convert all of the views in a subsystem to CMVC subsystems.  This
    -- command can convert more than one subsystem per call.

    procedure Check_Consistency (Views    : String := "<CURSOR>";
                                 Response : String := "<PROFILE>");

    -- Verify that all of the views are consistent with the CMVC invariants.
    -- Checks that:
    --      The configurations all exist and are correct.
    --      There are no dangling controlled objects.
    --      The imports are ok, and that all of the imported subsystems
    --          record the reference.
    --      Various other things.

    ------------------------------------------------------------------------
    -- User level commands for manipulating the compatibility database (CDB)
    -- associated with subsystems.
    ------------------------------------------------------------------------

    procedure Display_Cdb (Subsystem  : String  := "<CURSOR>";
                           Show_Units : Boolean := False;
                           Response   : String  := "<PROFILE>");

    -- Displays a summary of the information in the CDB. If "show_units"
    -- is true, then a summary of information for the units currently
    -- known in the subsystem is also displayed.
```





```
    procedure Make_Primary (Subsystem      : String  := "<SELECTION>";
                            Moving_Primary : Boolean := False;
                            Response       : String  := "<PROFILE>");

    -- Makes the subsystem into a primary subsystem with its own read/write
    -- CDB.  If the subsystem was a primary this operation is a no-op.  If
    -- the subsystem is a secondary then a new subsystem_id is assigned.
    -- If "moving_primary" is set to true, then the location of the
    -- primary for this subsystem is being moved and the current subsystem_id
    -- will be used.  When moving a primary the user must make sure
    -- that the original primary is either destroyed or converted into
    -- a secondary to prevent corruption of the CDB.

    procedure Make_Secondary (Subsystem : String := "<SELECTION>";
                              Response  : String := "<PROFILE>");

    -- Makes the subsystem into a secondary with the same subsystem_id.

    procedure Destroy_Cdb (Subsystem   : String  := "<SELECTION>";
                           Limit       : String  := "<WORLDS>";
                           Effort_Only : Boolean := True;
                           Response    : String  := "<PROFILE>");

    -- Destroys the CDB  and all remnants of it in compiled units.
    -- This includes demoting ALL units in the subsystem to source
    -- and deleting all code-only views.  If "effort-only" is set
    -- to true, then the effects of the operation are computed
    -- and displayed.

    procedure Update_Cdb (From_Subsystem : String := "<ASSOCIATED_PRIMARY>";
                          To_Subsystem   : String := "<SELECTION>";
                          Response       : String := "<PROFILE>");

    -- Moves the CDB from one subsystem to another using the network
    -- if necessary.  Both subsystems must have the same subsystem_id.

    procedure Repair_Cdb (Subsystem      : String  := "<SELECTION>";
                          Verify_Only    : Boolean := True;
                          Delete_Current : Boolean := False;
                          Response       : String  := "<PROFILE>");

    -- Will rebuild the CDB to be consistent with the currently compiled
    -- units in the subsystem.  If "verify_only" is true then the CDB
    -- will not be changed, but will be checked for consistency with
    -- the currently compiled units.  If "verify_only" is false and
    -- "delete_current" is true then the current CDB will be deleted
    -- and then rebuilt.  If the "verify_only" is false and
    -- "delete_current" is false then existing entries in the CDB
    -- will be verified and missing entries will be added.
```

```
    procedure Display_Code_View (View              : String  := "<CURSOR>";
                                 Verbose_Unit_Info : Boolean := False;
                                 Show_Map_Info     : Boolean := False;
                                 Response          : String  := "<PROFILE>");

end Cmvc_Maintenance;
```





```
package Command is

    procedure Diana_Edit (Name : String := "<IMAGE>");
    -- View the Diana tree for the current Command window.  Read only.

    procedure Spawn;
    -- Execute the contents of the current Command window as a background job.

    procedure Debug;
    -- Execute the contents of the current Command window under the control of
    -- the debugger.

    procedure Make_Procedure (Name    : String := ">>Simple Procedure Name<<";
                              Context : String := "$");

    -- Creates a procedure of the given Name in the given Context whose body is
    -- the contents of the command window that contains the cursor. With
    -- clauses are added to the procedure definition as needed so that
    -- unqualified names will semanticize correctly. Also, if needed, a
    -- Links.Add is attempted. This is an interactive command only. The command
    -- creates an Ada Editor window, builds the procedure, and leaves the
    -- cursor in that window when it is complete. Error messages will appear in
    -- the message window.

end Command;
```

```
package Common is

    procedure Abandon (Window : String := "<IMAGE>");
    -- Release all locks, and delete the associated window.
    -- This causes the loss of any editing changes.

    procedure Clear_Underlining;
    -- Remove underlining marks left on the image by previous commands.

    procedure Commit;
    -- Make changes to the image permanent

    procedure Complete (Menu : Boolean := True);
    -- Make the current image complete.  Provides syntactic and semantic
    -- completion, as possible.
    -- Menu => bring up a menu window for ambiguous references

    procedure Create_Command;
    -- Go to the command window for the current image, creating one if
    -- necessary.

    procedure Definition (Name     : String  := "<CURSOR>";
                          In_Place : Boolean := False;
                          Visible  : Boolean := True);
    -- Bring up the appropriate image to show the designated object.
    -- Do not make the image modifiable.  If a new window is required
    -- In_Place indicates that the current frame should be used.  Visible
    -- controls how names that resolve to both a visible part and a body
    -- should be resolved.  Visible causes the visible part to be pre-
    -- ferred; not Visible brings up the body if that is possible

    procedure Edit (Name     : String  := "<IMAGE>";
                    In_Place : Boolean := False;
                    Visible  : Boolean := True);
    -- Bring up the appropriate image to show the designated object.
    -- Attempt to make the image modifiable.
    -- In_Place and Visible are as in Definition.

    procedure Enclosing (In_Place : Boolean := False;
                         Library  : Boolean := False);
    -- Bring up the image for the object enclosing this one.
    -- In_Place is as in Definition.
    -- Library => the resulting image should be a Library; e.g. for Ada
    -- subunits, go to the enclosing directory rather than parent body.

    procedure Elide (Repeat : Positive := 1);
    -- Reduce the level of detail presented by the number of levels
```





```
    -- specified.  Attempts to expand beyond maximum level have no effect.
    -- It is not expected that Elide will reorder the presentation.

    procedure Expand (Repeat : Positive := 1);
    -- Increase the level of detail presented by the number of levels
    -- specified.  Attempts to expand beyond maximum level have no effect.
    -- It is not expected that Expand will reorder the presentation.

    procedure Explain;
    -- Provide additional information about the indicated object.
    -- The additional information may take the form of more detailed
    -- display or error message explanation.  If more detailed infor-
    -- mation is supplied, repeated applications cause the display to
    -- cycle through the available presentations.  For Ada, provides
    -- text of messages associated with underlinings.

    procedure Format;
    -- Format the current image appropriately for its image type.

    procedure Revert;
    -- Restore the image to the reflect the state of the underlying object.
    -- This causes the loss of any editing changes.

    procedure Release (Window : String := "<IMAGE>");
    -- Make changes to the designated image permanent (if applicable),
    -- release all locks, and delete the associated window

    procedure Semanticize;
    -- Perform semantic checking on the image.

    procedure Sort_Image (Format : Integer := 1);
    -- Sort the display according to the given format.  Format numbering is
    -- specific to the object type.  It is assume that if format 1 sorts by
    -- increasing values that format -1 will sort by decreasing values of
    -- the same key.  Clearly not relevant to all object types.

    procedure Demote;
    -- Bring the image to the next lower state.

    procedure Promote;
    -- Bring image to the next higher state.

    procedure Redo (Repeat : Positive := 1);
    -- Inverse of Undo

    procedure Undo (Repeat : Positive := 1);
    -- restore the contents of the image to the previous consistent state
```

```
    procedure Insert_File (Name : String := "<REGION>");
    -- Insert the contents of the indicated file into the current image

    procedure Write_File (Name : String := ">>FILE NAME<<");
    -- Write the contents to the named text file

    package Object is
        procedure Insert;
        procedure Copy;
        procedure Delete;
        procedure Move;
        procedure Previous    (Repeat : Positive := 1);
        procedure Next        (Repeat : Positive := 1);
        procedure Parent      (Repeat : Positive := 1);
        procedure Child       (Repeat : Positive := 1);
        procedure First_Child (Repeat : Positive := 1);
        procedure Last_Child  (Repeat : Positive := 1);
    end Object;

end Common;
```





```
with Action;
package Compilation is

    subtype Name      is String;
    subtype Unit_Name is String;

    -- All names are resolved in the established naming context for the job.

    -- A parameter of type Unit_Name may designate a set of Ada units,
    -- Worlds, Directories, or Activities. If a world or directory is
    -- designated, all Ada units contained by that world or directory are
    -- operated on.  If an activity is given, all Ada units in the views
    -- specified by the Activity are operated on.

    type Unit_State is (Archived, Source, Installed, Coded);
    subtype Change_Limit is String;

    -- Parameters of type Change_Limit control which units an operation is
    -- allowed to change in order to perform its task.  Three special values
    -- are predefined:

    Same_Directories  : constant Change_Limit := "<DIRECTORIES>";
    Current_Directory : constant Change_Limit := Same_Directories;

    -- Only units in the same directories as the units specified to the
    -- operation are allowed to change.

    Same_Worlds : constant Change_Limit := "<WORLDS>";
    Same_World  : constant Change_Limit := Same_Worlds;

    -- Only units in the same worlds as the units specified to the operation
    -- are allowed to change.

    All_Worlds : constant Change_Limit := "<ALL_WORLDS>";

    -- A unit in any world may be changed.
    -- A Change_Limit parameter may also be a string name that designates a
    -- set of worlds, directories or activities.  Only units in the
    -- designated worlds or directories are allowed to change.  The set of
    -- worlds designated by an activity is the set of views referenced by
    -- that activity.

    procedure Demote (Unit        : Unit_Name    := "<SELECTION>";
                      Goal        : Unit_State   := Compilation.Source;
                      Limit       : Change_Limit := "<WORLDS>";
                      Effort_Only : Boolean      := False;
                      Response    : String       := "<PROFILE>");
```

```
-- All units that must be demoted in order to demote the specified
-- unit will be demoted if possible. Any messages are appended to the
-- log file.

procedure Parse (File_Name      : Name    := "<REGION>";
                 Directory      : Name    := "$";
                 List           : Boolean := False;
                 Source_Options : String  := "";
                 Response       : String  := "<PROFILE>");

-- The named file must contain Ada source for a compilation.  After it
-- is parsed, the library compilation units are placed in the designated
-- Directory. LIST => true generates a listing of the input file into
-- the log file.  Wildcards in the File_Name are supported.


type Promote_Scope is (Single_Unit, Unit_Only, Subunits_Too,
                       All_Parts, Load_Views);

procedure Promote (Unit        : Unit_Name     := "<IMAGE>";
                   Scope       : Promote_Scope := Compilation.Subunits_Too;
                   Goal        : Unit_State    := Compilation.Installed;
                   Limit       : Change_Limit  := "<WORLDS>";
                   Effort_Only : Boolean       := False;
                   Response    : String        := "<PROFILE>");

-- Attempts to promote the units designated by the Unit parameter to the
-- designated Goal. The operation is a no-op if the units are already at
-- or beyond the goal state.

-- Unless the Scope is Single_Unit, Promote will attempt to promote the
-- ancestor units of, the visible part of, and any units with'ed by the
-- designated units before promoting the designated units. The with'ed
-- units must exist in the libraries specified by the Limit parameter.

-- Promotion of other units is NOT attempted; specifically: promotion of
-- siblings is NOT attempted.  If a designated unit is a visible part,
-- promotion of the body is NOT attempted.

-- Scope => Subunits_Too will cause subunits to be promoted.
-- Scope => All_Parts is equivalent to the Make procedure described below.
-- Scope => Load_Views is an even wider scope than All_Parts, in that it
-- will look through the current activity and try to make units
-- in referenced load views.

-- Semantic messages are attached to the tree. Semantic and other
-- messages are appended to the end of the Log_File.
```





```
procedure Make (Unit        : Unit_Name     := "<IMAGE>";
                Scope       : Promote_Scope := Compilation.All_Parts;
                Goal        : Unit_State    := Compilation.Coded;
                Limit       : Change_Limit  := "<WORLDS>";
                Effort_Only : Boolean       := False;
                Response    : String        := "<PROFILE>") renames Promote;

-- Same as Promote except that an attempt is made to promote the
-- secondary units of each visible part promoted.

procedure Delete (Unit     : Unit_Name    := "<SELECTION>";
                  Limit    : Change_Limit := "<WORLDS>";
                  Response : String       := "<PROFILE>");

-- Demotes and deletes the default version of the named unit and its
-- subunits.

procedure Destroy (Unit      : Unit_Name    := "<SELECTION>";
                   Threshold : Natural      := 1;
                   Limit     : Change_Limit := "<WORLDS>";
                   Response  : String       := "<PROFILE>");

-- Deletes and expunges all versions of the named unit and its subunits.
-- Wildcard notation may be used to specify more than one unit to be
-- destroyed. The Threshold is the number of objects to be destroyed per
-- unit specified.

procedure Compile (File_Name      : Name         := "<REGION>";
                   Library        : Name         := "$";
                   Goal           : Unit_State   := Compilation.Installed;
                   List           : Boolean      := False;
                   Source_Options : String       := "";
                   Limit          : Change_Limit := "<WORLDS>";
                   Response       : String       := "<PROFILE>");

-- Parses and promotes the units in the given file_name(s) (wildcards
-- allowed) to the given Goal state in the given Library according to
-- the Chapter 10 LRM rules for libraries.  If List is true a source
-- listing with interleaved error messages will be generated to the log
-- file.

procedure Dependents (Unit       : Unit_Name := "<IMAGE>";
                      Transitive : Boolean   := False;
                      Response   : String    := "<PROFILE>");

-- Displays the installed units that depend on (with) the given unit(s);
```

```
    procedure Atomic_Destroy (Unit      :     Unit_Name;
                              Success   : out Boolean;
                              Action_Id :     Action.Id    := Action.Null_Id;
                              Limit     :     Change_Limit := "<WORLDS>";
                              Response  :     String       := "<PROFILE>");

    -- Deletes and expunges all versions of the named unit and its subunits.
    -- Wildcard notation may be used to specify more than one unit to be
    -- destroyed.  The operation succeeds only if all designated units can
    -- be destroyed.

    procedure Load (From     : String := ">>MAIN_PROGRAM NAME<<";
                    To       : String := ">>LOADED_MAIN NAME<<";
                    Response : String := "<PROFILE>");

    -- Produce a Loaded_Main program from the main program specified by From.
    -- Put the result at To.

    procedure Set_Target_Key (The_Key  : String := "?";
                              To_World : String := "<IMAGE>";
                              Response : String := "<PROFILE>");

    -- Assign the target key to the specified world.  Once a key has
    -- been assigned to a world, the assignment can be changed only if
    -- the new key and the old key differ only in the front end/back end
    -- policy sub-components.  The default Key string, "?", causes a
    -- list of all available keys to be displayed.

    procedure Show_Target_Key (For_World : String := "<IMAGE>";
                               Response  : String := "<PROFILE>");

    -- Displays in the log the target key currently assigned to the
    -- indicated world.

    function Get_Target_Key (For_World : String := "<IMAGE>") return String;

    -- returns the image of the target key assigned to the indicated
    -- world.

end Compilation;
```





```
procedure Convert_Mailboxes (User, Mailbox : String := "@");

-- Convert the specified Mailbox owned by the specified User
-- from Mail 1 format to Mail 2 format.
-- Mailbox => "@" means "all mailboxes owned by User".
-- User => "@" means "all users".
```

```
with Calendar;
package Daemon is

    -- There are five types of Daemon tasks controlled by this package, their
    -- characteristics and default scheduling:
    --
    --      Snapshot.   Frequent.  ~1 minute slowdown.  Hourly.
    --
    --      Action.     Frequent, unobtrusive.  Every two hours.
    --
    --      Weekly.     Unobtrusive.  Weekly at 2:30 AM.
    --                  Code_Segment Group Session Tape Terminal User
    --
    --      Daily.      Variable, possibly significant interruption.
    --                  Nightly at 3:00 AM.
    --                  Ada DDB Directory Error_Log File Disk
    --
    --      Disk.       Daily or as needed.    Prolonged slowdown.
    --                  Last portion of the Daily run
    --
    -- If no other action is taken, all clients will be scheduled at a
    -- frequency and time normally appropriate.  These schedules can be
    -- changed to suit specific needs.  Note that Disk is included in the
    -- Daily category and will be run with the other Daily Daemons.
    --
    -- Clients that interfere with normal operations warn all users.
    --
    -- There is a group of clients referred to as Major_Clients that are
    -- expected to be of interest in monitoring the state of the machine:
    --     Snapshot, Action, Disk, Ada, DDB, Directory, and File.

    Major_Clients : constant String := "*";

    procedure Run (Client   : String := "Snapshot";
                   Response : String := "<PROFILE>");
    -- Cause the named Client to run the specified operation immediately;
    -- Has no effect on the next scheduled run of Client.

    procedure Schedule (Client    : String   := ">>CLIENT NAME<<";
                        Interval  : Duration;
                        First_Run : Duration := 0.0;
                        Response  : String   := "<PROFILE>");
    -- Sets the interval at which the Client operation will take place.

    procedure Quiesce (Client           : String   := ">>CLIENT NAME<<";
                       Additional_Delay : Duration := 86_400.0;
                       Response         : String   := "<PROFILE>");
```





```
    -- Reschedule the Client not to run at the next scheduled time.
    -- Equivalent to Schedule with a new First_Run, but the same Interval.
    -- Defaults to a 1-day delay; use Duration'Last for indefinite delay.

    procedure Status (Client : String := "*");
    -- print a formatted display of current status for given Client
    -- Matches on prefix of Client name, "" is prefix of all clients
    -- Major Clients (*): Actions, Ada, DDB, Directory, Disk, File, Snapshot
    -- The Disk Client provides additional information when run separately.

    procedure Warning_Interval (Interval : Duration := 120.0);
    function  Get_Warning_Interval return Duration;
    -- Warning given before starting Daily clients to allow time to Quiesce.

    function  In_Progress    (Client : String) return Boolean;
    function  Next_Scheduled (Client : String) return Calendar.Time;
    function  Last_Run       (Client : String) return Calendar.Time;
    function  Interval       (Client : String) return Duration;
    procedure Get_Size       (Client                :     String;
                              Size                  : out Long_Integer;
                              Size_After_Last_Run   : out Long_Integer;
                              Size_Before_Last_Run  : out Long_Integer);
    -- Sizes are set to -1 if invalid

    -- Control of the Disk Daemon
    --
    -- The Disk Daemon runs in response to a number of stimuli:
    --
    --   Daemon.Schedule   Runs at priority 6; intended for machine idle.
    --   Daemon.Run        Runs at priority -1; background collection.
    --   Daemon.Collect    Runs at specified priority
    --   over threshold    Starts at priority 0 with escalation
    --
    -- Messages to all users are issued for each of the three explicitly
    -- called collections.  In addition, a message is sent when a Set_Priority
    -- is called and it causes a change in priority.
    --
    -- A background task monitors over threshold situations and sends messages
    -- of interesting events.  Threshold_Warnings (False) allows an
    -- installation-provided job to tailor policy.
    --
    -- Additional control over Disk operations is available in the
    -- Disk_Daemon tools package.

    subtype Volume              is Integer range 0 .. 31;
    subtype Collection_Priority is Integer range -1 .. 6;
    -- -1 is the default and implies very low-level background activity
    --  0 guarantees progress in collection but has some effect on response
    --  6 causes collection to take over the machine
```

```
procedure Collect (Vol : Volume; Priority : Collection_Priority := 0);
-- If this call initiates a collection, it waits for its completion.

procedure Set_Priority (Priority : Collection_Priority := -1);
-- Set the priority of a currently running collection to Priority

procedure Threshold_Warnings (On : Boolean := True);
-- Cause messages to be sent when collection thresholds are passed.

--
-- Control of snapshot messages
--

procedure Snapshot_Warning_Message (Interval : Duration := 120.0);
procedure Snapshot_Start_Message   (On : Boolean := True);
procedure Snapshot_Finish_Message  (On : Boolean := True);
procedure Show_Snapshot_Settings;
procedure Get_Snapshot_Settings    (Warning        : out Duration;
                                    Start_Message  : out Boolean;
                                    Finish_Message : out Boolean);

------------------------------------------------------------------------
--
-- Control of the contents and permanence of the operations error log
--
------------------------------------------------------------------------

type Condition_Class is (Normal, Warning, Problem, Fatal);
type Log_Threshold   is (Console_Print, Log_To_Disk, Commit_Disk);

procedure Show_Log_Thresholds;
procedure Set_Log_Threshold (Kind : Log_Threshold; Level : Condition_Class);
function  Get_Log_Threshold (Kind : Log_Threshold) return Condition_Class;

-- Options on client compactions.
--
-- Consistency checking does additional work to assure that the internal
-- state of the system is as it seems.  This is normally only run when
-- there are suspected problems.  Consistency checking slows operations
-- for which it is meaningful by between one and three orders of magnitude.
--
-- Access_List_Compaction is the process of removing non-existent groups
-- from the access lists of objects.  This condition occurs when groups
-- are removed from the machine.  Access_List_Compaction is only done
-- for Ada, Directory and File clients.  All other clients reqested will
-- be silently ignored.  All three must be compacted for any old group
-- numbers to be freed.
--
-- The default is disabled.  The default is restored after
```





```
    -- the next appropriate daemon run has completed.

    procedure Set_Consistency_Checking (Client   : String  := "";
                                        On       : Boolean := True;
                                        Response : String  := "<PROFILE>");
    function  Get_Consistency_Checking (Client : String := "") return Boolean;

    procedure Set_Access_List_Compaction (Client   : String  := "";
                                          On       : Boolean := True;
                                          Response : String  := "<PROFILE>");
    function  Get_Access_List_Compaction (Client : String := "") return Boolean;

end Daemon;
```

```
package Debug is

    subtype Path_Name      is String;
    subtype Task_Name      is String;
    subtype Exception_Name is String;
    subtype Hex_Number     is String;

    -- A Path_Name is used to reference declarations, objects, statements,
    -- stack frames, tasks or types within progam units.

    -- Many commands take both a Path_Name and a Stack_Frame.  Though
    -- the Path_Name type allows the specification of a stack frame, the
    -- addition of the Stack_Frame parameter as a numeric value makes it
    -- possible to specify the stack frame as a numeric argument from the
    -- keyboard.  If both a Stack_Frame and Path_Name are specified, the
    -- Path_Name will be interpreted as the string Stack_Frame & Path_Name.

    -- Task_Name may be either a hex number or string name for the task.
    -- Exception_Name may be either a simple name for a predefined exception,
    -- or a pathname to an Ada identified.

    -- A Task_Name parameter of "all" specifies all tasks.  A Task_Name
    -- parameter of "" is interpreted as the control context task if explicitly
    -- set, otherwise, all tasks.  Exceptions to this rule are the commands
    -- Run and Stack, for which a Task_Name parameter of "" specifies the
    -- last task to stop if the control context is not explicitly set.


    -- Commands to terminate debugging

    procedure Debug_Off (Kill_Job : Boolean := False);
    -- Debug_Off terminates debugging on the job.  The job will run to
    -- completion if Kill_Job is false.  Otherwise, the job is terminated.

    procedure Kill (Job : Boolean := True; Debugger : Boolean := False);
    -- Kill can be used to kill either the job being debugged, or the
    -- debugger itself.


    -- Commands to query and modify program state

    procedure Put (Variable    : Path_Name := "<SELECTION>";
                   Stack_Frame : Integer   := 0);
    -- Display the value of the given object.

    procedure Stack (For_Task : Task_Name := "";
                     Start    : Integer   := 0;
```





```
                     Count    : Natural   := 0);
    -- Display Count stack frames for the specified task starting from frame
    -- Start.

    procedure Modify (New_Value   : String    := "";
                      Variable    : Path_Name := "<SELECTION>";
                      Stack_Frame : Integer   := 0);
    -- Modify the value of the given object.


    -- Commands to display ADA source

    procedure Display (Location    : Path_Name := "<SELECTION>";
                       Stack_Frame : Integer   := 0;
                       Count       : Natural   := 0);
    -- Display the source code for the given Location in the debugger window.
    -- If the Location specifies a subprogram, package, or task, display
    -- Count lines of source code including line numbers.

    procedure Source (Location : Path_Name := ""; Stack_Frame : Integer := 0);
    -- Like Definition, display the Location in an ada image.


    -- Breakpoint handling commands; break 0 represents all breaks

    procedure Break (Location         : Path_Name := "<SELECTION>";
                     Stack_Frame      : Integer   := 0;
                     Count            : Positive  := 1;
                     In_Task          : Task_Name := "";
                     Default_Lifetime : Boolean   := True);
    -- Set a break at the given location for the specified task. Count is
    -- the number of times the location is executed before the break is active.
    -- When Default_Lifetime is true, the breakpoint is temporary or permanent
    -- as specified by the Permanent_Breakpoints option; if false, its
    -- permanence is the opposite of the option.

    -- The breakpoint will be given a unique number which can be used as the
    -- breakpoint parameter of the Remove and Activate commands.

    procedure Remove (Breakpoint : Natural; Delete : Boolean := False);
    -- Deactivate the given breakpoint.  With delete false, the breakpoint
    -- can be installed again with the Activate command.
    -- Use Show (Breakpoints) to display breaks.

    procedure Activate (Breakpoint : Natural);
    -- Install a previously removed breakpoint.


    -- Commands to control all or individual tasks
```

```
procedure Stop (Name : Task_Name := "");
-- Stops execution of the specified task and keeps it stopped until
-- started by a call to Execute or Run naming the task or "all".

procedure Execute (Name : Task_Name := "");
-- Starts execution of the specified task if stopped.

procedure Xecute (Name : Task_Name := "");
-- same as Execute.

procedure Hold (Name : Task_Name := "");
-- Stops execution of the specified task and put it in the held state
-- until explicitly released by the command Release or a call to Execute or
-- Run explicitly naming this task.  The held state differs from the
-- stopped state in that Execute ("all") will not run a held task.

procedure Release (Name : Task_Name := "");
-- Releases a task from the held state and moves it to the stopped
-- state.  The task can then be started by a call to Execute or Run naming
-- the task or "all".

type Task_Category is
   (All_Tasks,    -- all known tasks
    Blocked,      -- tasks not in debugger, but not currently running
    Held,         -- tasks held in debugger (Hold command)
    Not_Running,  -- tasks not running for any reason
    Running,      -- tasks that are currently ready to run
    Stopped);     -- tasks stopped in the debugger (eg, at breakpoints)

procedure Task_Display (For_Task : Task_Name     := "";
                        Task_Set : Task_Category := Debug.All_Tasks);
-- Display information about tasks in the given category.

type Stop_Event is
   (About_To_Return,      -- stop after last statement of a subprogram
    Begin_Rendezvous,     -- stop before first statement of accept body
    End_Rendezvous,       -- stop after last statement of accept body
    Local_Statement,      -- stop before next statement at same level
    Machine_Instruction,  -- stop before next instruction
    Procedure_Entry,      -- stop before first stmt/decl of called proc
    Returned,             -- stop before next statement in caller
    Statement);           -- stop before next statement

procedure Run (Stop_At : Stop_Event := Debug.Statement;
               Count   : Positive   := 1;
               In_Task : Task_Name  := "");
-- Execute the specified task until the stop event has occurred
-- Count times.
```





```
procedure Clear_Stepping (For_Task : Task_Name := "");
-- Cancel any stepping operations for the given task.


-- Exception handling commands

procedure Catch (Name        : Exception_Name := "<SELECTION>";
                 In_Task     : Task_Name      := "";
                 At_Location : Path_Name      := "");
-- Stop execution when the specified exception is raised.  Can be
-- limited to a particular task or location.  Name = "all" catches
-- all exceptions; Name = "implicit" will catch implicitly raised
-- exceptions.

procedure Propagate (Name        : Exception_Name := "<SELECTION>";
                     In_Task     : Task_Name      := "";
                     At_Location : Path_Name      := "");
-- Request that execution not be stopped when the given exception is raised.

procedure Forget (Name        : Exception_Name := "<SELECTION>";
                  In_Task     : Task_Name      := "";
                  At_Location : Path_Name      := "");
-- Cancel a catch or propagate request.


-- Tracing commands

type Trace_Event is
   (All_Events,            -- Produce message for all of below
    Call,                  -- Message for each subprogram entry
    Exception_Raised,      -- Message for each exception raised
    Machine_Instruction,   -- Message for each statement/decl
    Propagate_Exception,   -- Message for each frame popped by propagation
    Rendezvous,            -- Message for each rendezvous start and end
    Statement);            -- Message for each statement/decl

procedure Trace (On          : Boolean     := True;
                 Event       : Trace_Event := Debug.All_Events;
                 In_Task     : Task_Name   := "";
                 At_Location : Path_Name   := "<SELECTION>";
                 Stack_Frame : Integer     := 0);
-- Enable or disable tracing.  Tracing displays information about
-- the execution of the given_task when the specified Trace_Events
-- occur.

procedure Trace_To_File (File_Name : String := ">> FILE NAME <<");
-- Send trace output to the specified file.  The null string
-- causes output to go to the debugger window.
```

```
-- History commands

procedure History_Display (Start    : Integer   := 0;
                           Count    : Integer   := 0;
                           For_Task : Task_Name := "");
-- Display Count history entries for the given task.  If Start is positive,
-- it specifies the starting location from the newest entry; if negative,
-- from the oldest entry.

procedure Take_History (On          : Boolean     := True;
                        Event       : Trace_Event := Debug.All_Events;
                        For_Task    : Task_Name   := "";
                        At_Location : Path_Name   := "<SELECTION>";
                        Stack_Frame : Integer     := 0);
-- Enable or disable history taking for the given task and location.


-- Commands to query debugger state

type Context_Type is (Control, Evaluation);

procedure Context (Set         : Context_Type := Debug.Control;
                   To_Be       : Path_Name    := "<SELECTION>";
                   Stack_Frame : Integer      := 0);
-- Set either the control or evaluation context.  Control context
-- is generally used when a Task_Name parameter of "" is specified.
-- The evaluation context is used as a prefix for unqualified location
-- and object names.


type Option is
   (Addresses,                 -- Include machine information
    Break_At_Creation,         -- Tasks stop before first decl
    Declaration_Display,       -- Include declarations in program display
    Delete_Temporary_Breaks,   -- Delete (vs deactivate) temp breakpoints
    Display_Creation,          -- Trace message for each task creation
    Echo_Commands,             -- Echo command in debugger window
    Freeze_Tasks,              -- Stop all tasks when one stops
    Include_Packages,          -- Task display includes packages
    Interpret_Control_Words,   -- Memory display for control stacks
    Kill_Old_Jobs,             -- Kill last debug job when next is begun
    Machine_Level,             -- Allow certain machine level operations
    No_History_Timestamps,     -- History display option
    Optimize_Generic_History,  -- No generic instance in history
    Permanent_Breakpoints,     -- Default breakpoints to permanent (vs temp)
    Put_Locals,                -- Put displays locals as well as parameters
    Qualify_Stack_Names,       -- Use fully qualified names in stack display
    Require_Debug_Off,         -- Debug_Off needed before debug next job
    Save_Exceptions,           -- Save exception-handling state across jobs
```





```
    Show_Location,             -- Display source in image when task stops
    Timestamps);               -- Include timestamps in command log

procedure Enable  (Variable : Option; On : Boolean := True);
procedure Disable (Variable : Option; On : Boolean := False) renames Enable;
-- Enable or disable the specified option.

type Numeric is
   (Display_Count,     -- Default for Count in Display command
    Display_Level,     -- Number of levels to expand Put command's data
    Element_Count,     -- Max elements of array for Put to display
    First_Element,     -- Offset for start of Put's array display
    History_Count,     -- Default for Count in History_Display
    History_Entries,   -- History buffer size
    History_Start,     -- Default for Start in History_Display
    Memory_Count,      -- Default for Memory_Dump Count parameter
    Pointer_Level,     -- Number of pointers to expand in Put's data
    Stack_Count,       -- Default frame Count for Stack command
    Stack_Start);      -- Default for Start in Stack command

procedure Set_Value (Variable : Numeric; To_Value : Integer);

procedure Flag (Variable : String := ""; To_Value : String := "TRUE");

type State_Type is (All_State, Breakpoints, Contexts,
                    Exceptions, Flags, Histories, Libraries,
                    Special_Types, Steps, Stops_And_Holds, Traces,
                    -- internal debugger state
                    Active_Items, Exception_Cache, Inner_State, Statistics);

procedure Show (Values_For : State_Type := Debug.Breakpoints);
-- Display information about various debugger facilities.

type Information_Type is (Exceptions, Rendezvous, Space);

procedure Information (Info_Type : Information_Type := Debug.Exceptions;
                       For_Task  : Task_Name        := "");
-- Display information about the specified task.

procedure Comment (Information : String := "");
-- place a comment in the debugger window.

procedure Set_Task_Name (For_Task : Task_Name := "";
                         To_Name  : String    := "");
-- Set a task synonym for the specified task for use as a Task_Name
-- parameter to commands.

procedure Convert (Number : String := ""; To_Base : Natural := 0);
-- Hex/decimal conversion.
```

```
procedure Reset_Defaults;
-- Reset flags to initial values.
-- Unregister all special types.

procedure Current_Debugger (Target : String := "");
-- Set current debugger to the current window, or Target if
-- specified.  Subsequent calls to Debug will be directed to
-- the specified target or native debugger.


-- Machine-level commands

-- For the following commands, address format is #Segment, #Offset
-- memory format is one of CONTROL, TYP, QUEUE, DATA, IMPORT, CODE, SYSTEM

procedure Memory_Display (Address : String  := "";
                          Count   : Natural := 0;
                          Format  : String  := "DATA");

procedure Location_To_Address (Location    : Path_Name := "<SELECTION>";
                               Stack_Frame : Integer   := 0);
procedure Address_To_Location (Address : String := "");
procedure Exception_To_Name   (Implementation_Image : String := "");

procedure Memory_Modify (Address : String  := ">>HEX ADDRESS<<";
                         Value   : String  := ">>HEX VALUE<<";
                         Width   : Natural := 0;
                         Format  : String  := "DATA");
-- The format string is used to distinguish various addressing modes.
-- Width is interpreted according to the machine, where 0 is the
-- natural word width.

procedure Register_Display (Name        : String    := "";
                            For_Task    : Task_Name := "";
                            Stack_Frame : Integer   := 0;
                            Format      : String    := "");
-- "" implies display of interesting registers.
-- "ALL" displays all possible machine registers.

procedure Register_Modify (Name        : String    := ">>REGISTER NAME<<";
                           Value       : String    := ">>HEX VALUE<<";
                           For_Task    : Task_Name := "";
                           Stack_Frame : Integer   := 0;
                           Format      : String    := "");


procedure Object_Location (Variable : Path_Name := "<SELECTION>";
                           Options  : String    := "");
-- Display the machine location of the given Object.
```





```
-- Options describe various target specific kinds of information
-- to display.


procedure Attach_Process (Name : String := ""; Options : String := "");
-- Register the named process for control under the current debugger.


procedure Target_Request (Options : String := ""; To_File : String := "");
-- Perform target specific requests.
-- Output can be directed to the Debugger Window or to some other
-- file or device. Null value indicates debugger window.


procedure Connect (Remote_Machine : String := ""; Target : String := "");
-- Hook up a debugger to given machine.
-- Target should specify a target key - "" implies we can calculate
-- the target from the remote machine name.

procedure Invoke (Main_Unit : String  := "<IMAGE>";
                  Options   : String  := "";
                  Spawn_Job : Boolean := True);
-- Find or start a debugger to the given machine, and start debugging
-- the indicated job.

procedure Reconnect;
-- attempt to reestablish communication after failure

end Debug;
```

```
package Debug_Maintenance is

    procedure Wait_For_Job;
    -- Hang while debugger has running flag on.  Returns when program
    -- stops in debugger.

    procedure Show_Version;
    -- display the version of the current debugger;

end Debug_Maintenance;
```





```
package Dependents is

    type Display_Kind is (Subsystems, Views, Units, Parents, Item_Kinds,
                          Units_And_Kinds, Units_And_Items, Parents_And_Items);

    type Dependents_Level is (Unfiltered, Immediate,
                              Reference_Closure, Demote_Closure);

    procedure Show (Name    : String           := "<CURSOR>";
                    Display : Display_Kind     := Dependents.Units;
                    Level   : Dependents_Level := Dependents.Unfiltered;
                    Limit   : String           := "<ALL_WORLDS>";
                    Global  : Boolean          := True;
                    Options : String           := "");
    --
    -- invoke the dependents object_editor.
    --
    -- dependents are entities which depend on the identifier specified
    -- by the first argument. this dependence can be because of a direct
    -- naming of that identifier, because of being the second part of a
    -- two part item or because of a closure computation.
    -- dependents can be either compilation units or constructs
    -- within compilation units.
    --
    -- in the comments that follow the term "unit" will mean a
    -- compilation unit.
    -- the term "item" will refer to a construct within a compilation unit.
    -- the term "main_item" will refer to the unit or item whose
    -- dependents are being shown.
    -- the term "kind" will refer to the kind of an item.
    -- the term "parent" will refer to an ada program unit.
    --
    -- the first argument to dependents.show can be a naming expression
    -- which resolves to several def_ids in the same ada space.
    --   e.g. dependents.show ("directory.naming.resolve'spec");
    -- this will run the dependents analysis over all of the def_ids
    -- resolved to by the naming expression.
    -- one can also place marks on a group of def_ids in the same space
    -- and invoke the command as:
    --  dependents.show ("<marks>", options => "count=n");
    -- where n is the number of marks to be taken off the mark_stack.
    -- this will run the dependents analysis over all the def_ids
    -- removed from the mark_stack.
    --
    -- the line(s) of a dependents display before the first blank line
    -- denote the main_item(s).
    -- the succeeding nonblank lines denote dependents.
```

```
--
-- the display argument determines whether unit level or item
-- level dependents are shown and how the information is presented.
-- (see below).
--
-- the level argument determines what level of dependence should be
-- shown initially. this can be changed after the display is brought up.
--
-- limit, if not defaulted, restricts the displayed dependents to
-- those that occur within objects specified by the limit
-- naming expression.
--
-- if global = false then only uses within the unit containing the
-- identifier resolved to will be shown (by underlining them).
--
---------------------------------------------------------------------------
--
-- explanation of the display kinds.
--
-- these different displays can be produced using the elide/expand
-- and sort commands. the display kinds are ordered so that moving
-- through them produces more specific information about the dependents.
--
--  Subsystems
--    full names of subsystems containing views with dependents.
--
--  Views
--    full names of views containing units with dependents.
--
--  Units
--    names of compilation units that contain dependents or are
--    themselves dependents.
--    see below for how to control how compilation unit names
--    are displayed.
--
--  Parents
--    names of ada program units containing dependents.
--    these will be the names of procedures, functions, packages,
--    tasks, generics, or record_types which immediately contain
--    dependent items.
--    this display lists the parent name followed by the unit name.
--    example:
--
--       set_value    STACK_MANAGER'BODY !L1.L2.L3
--       get_value    STACK_MANAGER'BODY !L1.L2.L3
--       add_element  TEST'BODY          !U1.U2
--
--    this says that program unit set_value contains a dependent
--    as does get_value and add_element.
```





```
--
--    in the case where the parent ada program unit is itself a
--    compilation unit or in the case where an entire compilation
--    unit is a dependent, the parent name field will be given as
--    *comp_unit*.
--
-- Item_Kinds
--    this display shows the kind of the nearest ada construct
--    enclosing a dependent, followed by the parent name.
--    the constructs will be from the categories statements,
--    declarations, etc.
--    there will be one line per dependent construct.
--    example:
--
--       assign  set_value    STACK_MANAGER'BODY !L1.L2.L3
--       if      get_value    STACK_MANAGER'BODY !L1.L2.L3
--       call    add_element  TEST'BODY          !U1.U2
--
--    the first line says that an assignment statement in the
--    program unit set_value which is in the compilation unit
--    !L1.L2.L3.STACK_MANAGER'BODY has a dependent within it.
--
--    if the display level is demote_closure (see below) then
--    the construct shown will be the nearest enclosing demotable
--    item containing dependents.
--
--    sometimes an item level dependent will be an entire comp unit.
--
-- Units_And_Kinds
--    this display shows the same information as the previous one
--    but it presents each compilation unit first, followed by its
--    list of dependents. with a blank line between each comp unit.
--    example:
--
--       STACK_MANAGER'BODY !L1.L2.L3
--         assign set_value
--         if     get_value
--
--       TEST'BODY          !U1.U2
--         call   add_element
--
-- Units_And_Items
--    this display is similar to the previous one in that it
--    shows the name of each compilation unit preceded by a blank
--    line; but rather than showing just the name of the kind of
--    construct which has a dependent, it shows a compressed one
--    line form of the text of the ada construct.
--    the place within the construct that has a reference to the
--    main_item will be underlined. (not shown here).
```

```
--    example:
--
--      STACK_MANAGER'BODY !L1.L2.L3
--         Abc := Def (3);
--         if Def (5) then
--
--      TEST'BODY          !U1.U2
--         Foo (Stack_Manager.Def (4));
--
-- Parents_And_Items
--   this display is similar to the previous one with the change that
--   the one line form of the text of an ada construct is replaced
--   by a group of lines of the form
--
--       parent_kind parent_name
--         line 1 of ada text
--         ...
--         line n of ada text.
--
--   example:
--
--      STACK_MANAGER'BODY !L1.L2.L3
--         proc_body set_value
--           Abc := Def (3);
--
--         func_body get_value
--           if Def (5) then
--               Mumble;
--           end if;
--
--      TEST'BODY          !U1.U2
--         proc_body add_element
--           Foo (Stack_Manager.Def (4));
--
---------------------------------------------------------------------------
--
-- compilation unit and library names are always shown in upper case.
-- parent names and item kinds are shown in lower case.
--
-- full names of compilation units can be displayed either
-- with the library unit name first, some space, and then the
-- library name; or as the standard pathname with library name
-- first, followed immediately by the library unit name.
--
-- there is an option to control which format is used.
-- this option is either set when the command is invoked;
-- or if it is not explicitly given in the option string,
-- the session switch:
--    Dependents_In_Order_Pathnames is checked.
```





```
-- furthermore after a display has been brought up the value of this
-- option can be changed by the In_Order command
--
-- if pathnames are being shown in transposed order there is an option
-- to control whether library names are shown.
-- there is also a corresponding session switch called:
--    Dependents_Show_Library and a command called Libraries.
--
-- there is an option to control whether compilation states are
-- shown for the units or items in a display.
-- there is also a corresponding session switch called:
--    Dependents_Show_Unit_State and a command called States.
-- compilation states will be shown as a single letter from
-- S, I, C, preceding the unit or item.
--
-- there is a session switch called
--    Dependents_Delta0_Compatibility (which defaults to true)
-- which if set implies in_order pathnames and controls the
-- format of the first line.
-- this switch also controls the contents of the first line
-- in obsolescence windows.
--
---------------------------------------------------------------------------

-- dependents_level meanings:
--
--   unfiltered
--     dependent units directly from the ddb are shown without
--     checking if they actually have references to the main_item.
--     sometimes more units will be listed than those that actually
--     have dependents. this can occur in the case of a main_item
--     which is overloadable; or in the case of a unit which once had
--     a dependent but no longer does. also the unit containing
--     the main_item is always listed as a potential dependent.
--     this display is only possible for unit displays.
--     if an elision command is given to change the display to an
--     item display the units will be automatically filtered.
--     also if the main_item is a record field the initial display
--     will always be filtered.
--
--   immediate
--     units are checked for references.
--     only units which actually have dependents are shown.
--
--   reference_closure
--     in addition to showing immediate references:
--
--     * for types, packages, subprograms, exceptions
--     references to renames and derivations are included.
```

```
--      * for items in generic specs -  references to the corresponding
--      id in instantiations of that generic are shown.
--      * for subprogram parameters - parameters passed in calls
--      are shown.
--      * for generic parameters - actual parameters passed to the
--      parameter in instantiations are shown.
--
--      also, for any main_item which can occur as a default value for a
--      subprogram parameter, any call which uses the default will be
--      marked as a dependent of the main_item.
--
--    demote_closure
--      units or items which have to be demoted to demote the
--      main_item are shown.
--
--      for the demote/promote commands when region selections are used,
--      which display kind is being shown, determines whether whole
--      compilation units are demoted/promoted or individual items
--      within compilation units are demoted/promoted.
--      if the display kind is Units then comp units will be
--      demoted/promoted. otherwise items will be demoted/promoted.
--      (even though comp_unit names occupy a separate line in
--      certain elision levels and may be included in the selection).
--
--      when item level demote_closure is being shown, contiguous
--      dependent items will be merged into one line in the display.
--      (and will be demoted as a group).
--
--      scenario for editing the main_item:
--
--      * bring up an initial unit display.
--      * if you wish to demote item level dependents.
--        create a command window and type: items;
--        this will change the display kind to item level dependents.
--      * press the complete key.
--        this will change the display to demote_closure
--      * select the entire display.
--      * press the demote key.
--        this will demote all dependents and the main_item to source.
--      * to edit the demoted main_item or any of the demoted dependents
--        put the cursor on the line which represents that item or unit
--        and press the edit key.
--        this will bring up a window on the unit or item.
--      * make any changes, committing the windows.
--        when all changes have been made, return to dependents menu and
--      * select the display.
--      * press the promote key.
--        this will promote all source units or items in the display
--        to installed.
```





```
--        if any items fail to install they can be fixed and promoted
--        again with single line selections or if a region is selected and
--        it contains a mix of both source and installed items only the
--        source items will be promoted.
--
-- remember that certain kinds of items which one can produce an xref of
-- cannot be incrementally edited. these include enumeration literals,
-- record fields, subprogram and generic parameters.
--
---------------------------------------------------------------------------
--
-- options on show command.
--
--   IN_PLACE
--      brings up the window on top of the current window.
--      default is false.
--
--   NEW_WINDOW
--      always create a new window for this display.
--      default is to reuse an existing dependents window.
--      previous contents of a reused dependents window can be
--      revisited using object undo/redo or the Contents command.
--
--   MENU_ALWAYS
--      if menu_always is false (the default) and the only dependents
--      are within the same unit as the main_item then no window is
--      brought up and the dependents are underlined.
--      if menu_always is true then a window is always brought up
--      if there are any dependents.
--
--   COUNT=<integer>
--      count > 1 specifies that count-many contiguous def_ids
--      starting at the one selected are to have their dependents
--      shown. default is 1.
--
--   STATES
--      show unit/item compilation states, default is false.
--
--   IN_ORDER
--      show comp unit names with library first, default is false
--
--   LIBRARIES
--      show library names, default is true.
--
--   SOURCE
--      default is false.
--      search source units which are registered in the ddb
--      for dependents. also search source bodies/subunits of
--      units whose specs/bodies have been checked.
```

```
--      checking for dependents within source units will not in
--      general be exact. in the absence of semantic information
--      a heuristic approach is taken which in most cases (if the
--      identifier of the main_item is distinctive) will find
--      the set of dependents (and possibly some other items which
--      aren't actually dependents).
--
--   CHECK=<naming_expression>
--      check any source units in the naming expression for references.
--
--   RESTRICT = <argument>
--      see Restrict command below for legal argument values.
--
-----------------------------------------------------------------------
--
-- common commands which can be applied when the cursor is in a
-- dependents window.
--
-- * common.definition
--    visit the entity denoted by the line the cursor is on.
--    if the line denotes a compilation unit or parent with dependents
--    then they will be underlined and the cursor positioned to the
--    first of them.
--    if line denotes an item then the cursor will be positioned
--    on that item, which will be selected.
--    if the cursor line has a parent or unit name in it and the cursor
--    has been moved onto that parent or unit name then that parent
--    or unit will be visited.
--    if the cursor line is showing full ada text and the cursor has been
--    moved to a using identifier the definition of the using identifier
--    will be visited.
--    the node which would be visited by definition is the node which
--    this editor returns when name resolution of <CURSOR> is run
--    against it. so if while reading an xref display one wants to
--    run another xref off an item in the display this can be done
--    by positioning the cursor on the desired item in the xref
--    display and invoking the xref command.
--
-- * common.edit:
--    (demote if necessary and possible and) edit the item
--    of the line the cursor is on.
--    if the item is in the installed (or greater) state it
--    must be selected.
--
-- * common.enclosing:
--    find the image of the object for which these items are dependents.
--
-- * common.demote:
--    demote the entities corresponding to the set of lines selected.
```





```
--    if the entire image is selected the goal state will be source.
--    otherwise it will be the predecessor of the current state of the
--    entity.
--
-- * common.promote:
--    promote the entitites corresponding to the set of lines selected.
--
-- * common.format
--    if display is unfiltered, change it to immediate.
--
-- * common.semanticize
--    if display is unfiltered, change it to immediate.
--    if display is immediate, change it to reference_closure.
--
-- * common.complete
--    change display to demote_closure.
--
-- * common.undo and redo
--    move thru the images in this window.
--
-- * common.revert:
--    recompute the current dependents set of the main_item.
--
-- * common.object.delete
--    destroy the item selected.
--
-- * common.expand and elide
--    relative motion among the display kinds.
--    repeat count > 1 moves multiple steps.
--    if there is no selection the entire display is moved to a
--    new elision level.
--    if there is a selection, only the selected entity will be
--    expanded/elided.
--    if a selected item is at the final elision level and expand
--    is typed, the display will be replaced by the ada text
--    for an enclosing construct of that item.
--    this will continue with further expands until the program unit
--    containing the item fills the display.
--    if the display is showing just an expanded_item,
--    elide will transition the display back to the full dependents
--    display.
--
-- * common.sort_image
--    absolute addressing of display kinds based on format parameter.
--    1 = subsystems, 2 = views, 3 = units, 4 = parents, 5 = item_kinds,
--    6 = units_and_kinds, 7 = units_and_items, 8 = parents_and_items.
--    if a display has some lines at a different elision level from
--    the entire display level, 0 as a sort prefix will change all
--    lines to the current display level.
```

```
--
-- * common.object.child
--    if no selection, select the item under the cursor, otherwise
--    expand the item under the cursor.
--    if the entire display is showing an expanded_item, this command
--    will not require a selection to expand.
--
-- * common.object.next/previous
--    if no selection, select the item under the cursor.  If there is a
--    selection, move it to the next/previous item.

-- * common.object.parent
--    if no selection, select the item under the cursor.  If the item is
--    selected, expand the selection.
--
---------------------------------------------------------------------------
---------------------------------------------------------------------------
--
package Commands is
    -- commands which can be run off of a dependents window

    procedure Items;
    -- change the display to an item level display

    procedure Units;
    -- change the display to a unit level display

    procedure Parents;
    -- change the display to a parents display

    procedure Immediate;
    -- change the display to an immediate dependents display

    ----------------------------------------------------------------------

    procedure Libraries (Show : Boolean := True);
    -- control whether library names are displayed

    procedure States (Show : Boolean := True);
    -- control whether unit states are displayed

    procedure In_Order (Value : Boolean := True);
    -- control whether comp unit names have library names shown first

    procedure Contents;
    -- show a table of contents for the images in this window.

    ----------------------------------------------------------------------
```





```
    procedure Keep;
    -- if there is a selection then remove any units not selected
    -- from the display. it there is no selection remove all but the
    -- line the cursor is on.

    procedure Remove;
    -- it there is a selection then remove selected units from the
    -- display. if there is no selection remove the unit the cursor is on.

    procedure Show_All;
    -- redisplay any units removed by keep, or remove.

    procedure Limit (To : String := "<ACTIVITY>");
    -- limit the set of units displayed.

    procedure Unlimit;
    -- show any units that were removed by a limit command

    ------------------------------------------------------------

    procedure Uncode;
    -- demote all coded units selected in the display to installed

    procedure Source;
    -- demote all units selected in the display to source

    procedure Installed;
    -- promote all units selected in the display to installed

    procedure Coded;
    -- promote all units selected in the display to coded

    procedure Remake;
    -- promote all units selected in the display to their state when
    -- first displayed.

    ----------------------------------------------------------------------

    procedure Look_Through (Name  : String  := "<ACTIVITY>";
                            Limit : Boolean := False);

    -- the main_item must be in a spec.
    --
    -- if name is an activity:
    --    if the main_item is in a load_view find its corresponding
    --    id in the spec view implied by the activity and
    --    display that id's dependents.
    --    if limit is true then limit the dependents to views in the
    --    activity.
```

```
--
--   if the main_item is in a spec_view find its corresponding
--   id in the load view implied by the activity and
--   display that id's dependents.
--   limit is not relevant is this case.
--
-- if name is a view or world:
--   find the corresponding id for the main_item in the given
--   view or world and display that id's dependents.

procedure Succ (Repeat : Integer := 1);
-- reconstruct the display with the next def_id after the
-- current main id

procedure Pred (Repeat : Integer := 1);
-- reconstruct the display with the prior def_id to the current
-- main id

procedure Diana_Edit;
-- run diana_edit on the node the cursor is on.

------------------------------------------------------------

procedure Restrict (To : String);

-- restrict the display to a particular kind of reference.
-- the argument can be a sequence of one or more of the following
-- restriction kinds:
--
--   internal       - show only uses of the main_item (which must
--                    be in the visible part of a package)
--                    which are within that package
--   external       - show only uses of the main_item (which must
--                    be in the visible part of a package)
--                    which are outside of that package
--   defaulted      - show calls where a parameter is defaulted.
--   nondefaulted   - show calls where a parameter isn't defaulted.
--   used_ops       - show only operators made visible by a use clause
--   used_ids       - show only non operators made visible by a
--                    use clause
--   closure        - show only references which are in the
--                    reference_closure and not in the immediate set.
--   writes         - show places where an object is assigned to.
--
--   variables, constants, numbers, exceptions, types,
--   subtypes, generics, tasks, parameters, withs, uses,
--   subprograms, specs, bodies, renamings, instantiations,
--   compound_types, record_types, array_types,
--   derived_types, access_types, numeric_types, enum_types,
```





```
--   stmts, alternatives, handlers, accepts, assigns,
--   calls, loops, ifs, cases, returns, raises, selects,
--   aggregates, allocators, relationals, conversions,
--   pragmas
--
-- any unique prefix of a restriction kind can be given.
-- several restrictions can be given in one command by
-- separating them by a space. e.g.
--    restrict ("external alternatives"); or
--    restrict ("ext alt");
-- this means only show case alternatives which are external to
-- the package containing the main_item.
--
-- a restriction kind can be negated by prefixing it with
-- a not sign (~). e.g.
--    restrict ("~withs ~parameters);
-- this means only show contructs that aren't with clauses and
-- aren't parameter declarations.
--
-- restrict ("?"); displays the names of all restriction kinds.


procedure Unrestrict;
-- show any items that were removed by a restrict command

------------------------------------------------------------

procedure Display (Constructs : String);

-- find transitive uses of a main_item which is a type
-- or record field in a particular kind of construct.
-- possible values of the argument are:
--    aggregates,
--    allocators,
--    assigns,
--    cases,
--    creates,
--    assign_closure,
--    create_closure.
--
-- any unique prefix of the argument name can be given.
-- several arguments can be given by separating them by spaces.
--   e.g. display ("creates create_closure");
--
-- aggregates shows all aggregates over the main_item if it is
-- a compound type (record, array);
-- if the main_item is a discrete type it shows array aggregates
-- whose index type is the main_item.
--
```

```
    -- allocators shows all allocators whose result type is the main_item.
    --
    -- assigns show assigment statements where an object of the type
    -- is assigned. in the case of compound types it also shows where an
    -- assignment to a subcomponent is done.
    --
    -- cases shows all cases statements over the main_item
    -- (which must be discrete type).
    --
    -- creates shows places where a variable or constant of the type
    -- is declared; or where an allocator of an object of the type
    -- is done.
    --
    -- assign_closure, create_closure run the analysis over compound types
    -- which contain the main_item as a subcomponent.

    -- object undo will change the display back to a regular
    -- dependents display.
    --
    ------------------------------------------------------------

    procedure Replace (Target      : String  := "";
                       Replacement : String  := "";
                       Wildcard    : Boolean := False;
                       Main_Also   : Boolean := True);
    -- edit each source item in the display and do a search/replace
    -- of the target with replacement.

    ------------------------------------------------------------

    function Names return String;
    -- return a string of the form [u1,u2,...un]
    -- where ui is the ith unit showing in the display
    -- or in the selection if there is one.


---------------------------------------------------------------------------

    procedure Delete;
    -- for a window with a set of images within it,
    -- remove this image from the set and redraw the window
    -- with the prior one in the history.


---------------------------------------------------------------------------

    procedure Subunit_Levels (Number : Integer := Integer'Last);
    -- number of parent library unit names to show for the non in_order
    -- unit name field.
```





```
        -- a value of 1 means show the name of the innermost enclosing
        -- subunit only (if the enclosing unit is a library unit it
        -- will always be shown).
        -- a value of 2 means show the names of at most 2 of the innermost
        -- enclosing subunits and so on.
        -- a value of integer'last causes full names to be shown again.
        -- a value of 0 means don't show subunit names just show the
        -- library unit name.
        -- a value of integer'first means don't show the unit name field
        -- at all in the items and parents elision levels.
        -- a value of -x means remove abs (x) names from the front of
        -- a name.

        procedure Parent_Levels (Number : Integer := Integer'Last);
        -- number of parent names to show in the parent name field.

        procedure Unit_Length (Max : Integer := Integer'Last);
        -- max length name to display in the unit name field.

        procedure Parent_Length (Max : Integer := Integer'Last);
        -- max length name to display in the parent name field.
    end Commands;

    ---------------------------------------------------------------------------
    ---------------------------------------------------------------------------

    procedure Window (Kind     : String  := "xref";
                      Restart  : Boolean := True;
                      In_Place : Boolean := False;
                      Wrap     : Boolean := False);

    -- make visible a menu window.
    -- choices for kind include:
    --  xref, find, unused, errors, holds, ada_menu, menu.
    -- if restart = true bring up the most recently created.
    -- if there are multiple windows of the kind specified and
    -- restart = false then cycle thru the windows with successive calls.
    -- when at last window, wrap controls whether to go back to first.

    ---------------------------------------------------------------------------

    procedure Find (Pattern     : String       := "";
                    Units       : String       := "$@?";
                    Display     : Display_Kind := Dependents.Units_And_Kinds;
                    Wildcard    : Boolean      := False;
                    Ignore_Case : Boolean      := True;
                    Options     : String       := "");

    -- search the ada units specified by units for the given pattern.
```

```
    -- any units with occurences will be displayed in a menu.


    procedure Unused (In_Units : String := "<CURSOR>";
                      Check_Other_Units : Boolean := False;
                      Display : Display_Kind := Dependents.Units_And_Kinds;
                      Options : String := "");

    -- show the declarations in units that are not referenced.
    -- any units with unused declarations will be displayed in a menu.

    -- for find and unused the following are allowed.
    --
    -- options:
    --   in_place, new_window, restrict, libraries, states, in_order,
    --   menu_always.
    --
    -- common commands:
    --  definition, edit, enclosing, demote, promote, delete,
    --  elide, expand, sort, next, previous, parent, child, undo, redo,
    --  revert (for unused only).
    --
    -- dependents commands:
    --  libraries, in_order, states, items, units, parents,
    --  limit, unlimit,  restrict, unrestrict,  keep, remove, show_all,
    --  contents, replace, names, diana_edit,
    --  uncode, source, installed, coded, remake.
    --
    ---------------------------------------------------------------------------

    procedure Menu (Units      : String;
                    First_Line : String := "";
                    Name_Field : String := "";
                    Options    : String := "");

    -- display a menu of the units resolved to by units.
    --
    -- options allowed:
    --   in_place, new_window, libraries, states, in_order.
    --
    -- common commands:
    --   definition, edit, enclosing, demote, promote, parent, child,
    --   next, previous, undo, redo
    --
    -- dependents commands:
    --   replace, limit, unlimit, keep, remove, show_all, contents,
    --   in_order, states, libraries, uncode, source, installed, coded, remake.

    ---------------------------------------------------------------------------
```





```
    procedure Errors (Units : String := "<CURSOR>"; Options : String := "");

    -- display a menu of the error messages in the units specified.
    -- if <cursor> is given and the cursor is on a node with error messages
    -- then just display the messages on that node.
    --
    -- if an error message refers to another node that node can be accessed
    -- from the menu by moving the cursor to the end of the error message
    -- text and doing definition.
    --
    -- options allowed:
    --   in_place, new_window, in_order, libraries, menu_always,
    --   kind =
    --      all_errors - the default, show all messages.
    --      warnings   - show only warnings messages
    --      errors     - show only error messages, no warnings.
    --
    --   if menu_always is false then no menu will be brought up. the
    --   error nodes will be underlined in the ada unit.
    --
    ---------------------------------------------------------------------------

    procedure Holds (Count : Integer := 10; Options : String := "");

    -- show a menu of the items on the hold stack.

end Dependents;
```

```
package Diana_Tree is

    procedure Ada_Edit (Name : String := "<IMAGE>");

end Diana_Tree;
```





```
package Disk_Space is

    type Acceptable is (Any_Space, Any_Permanent_Space,
                        Committed_Permanent_Spaces,
                        Undeleted_Committed_Permanent_Spaces);

    type Traversals is (Poly_File_Space, Directory_Space, Eedb_Space,
                        Constant_Space, Moribund_Space, Backup_Database_Space);

    type Traversing is array (Traversals) of Boolean;

    All_Traversals : constant Traversing := Traversing'(others => True);
    Directory_Only : constant Traversing :=
       Traversing'(Directory_Space => True, others => False);
    No_Traversals  : constant Traversing := Traversing'(others => False);

    -- Possible decodings of a space.
    -- Class (R1000_Native_Code .. R1000_Cross_Code) are instruction spaces.
    -- Class (R1000_Import) is any import space.
    -- Class (Diana_Tree .. Other) are module spaces.
    -- Class (Diana_Tree .. Seg_Heap_Other) are all segmented heaps.
    -- Class (Poly_Text .. Poly_Other) are all Polymorphic_Io creations.
    -- Class (Backup_Master .. Backup_Tape) are Backup database spaces.
    -- Class (Garbage) is a garbage collected (by the Disk_Cleaner) space.

    type Class is (R1000_Native_Code, R1000_Cross_Code, R1000_Import,
                   Diana_Tree, Text_File, Image, Link_Pack,
                   Poly_Text, Poly_Object_Id, Poly_State, Poly_Other,
                   Backup_Id, Backup_Backup, Backup_Processor,
                   Backup_Disk, Backup_Tape, Backup_Master,
                   Configuration, Seg_Heap_Other, Garbage, Other);

    type Classes is array (Class) of Boolean;

    All_Classes     : constant Classes := Classes'(others => True);
    Module_Classes  : constant Classes := Classes'(R1000_Native_Code => False,
                                                   R1000_Cross_Code  => False,
                                                   R1000_Import      => False,
                                                   others            => True);
    Matching_Class  : constant Classes := Classes'(others => False);
    Unknown_Classes : constant Classes := Classes'(Poly_Other     => True,
                                                   Seg_Heap_Other => True,
                                                   Other          => True,
                                                   others         => False);

    type Space_Kind is (Instruction, Import, Module);
    Data : constant Space_Kind := Module;
```

```
-- Examine_Spaces locates all spaces known to the kernel and discards
-- any that are either unacceptable or can be reached through one of
-- the traversals.
-- The Summarize booleans cause listings of the space counts / sizes to
-- be printed.  List_Lost causes Space_Information for the unreachable
-- spaces to be printed.

procedure Examine_Spaces
             (Examine        : Traversing := Disk_Space.All_Traversals;
              Filter         : Acceptable :=
                 Disk_Space.Undeleted_Committed_Permanent_Spaces;
              Permit         : Classes    := Disk_Space.All_Classes;
              Summarize_All  : Boolean    := False;
              Summarize_Lost : Boolean    := True;
              List_Lost      : Boolean    := False;
              Verbose        : Boolean    := True);


-- Attempts to find the name of the object which contains the space
-- specified, and prints that name.  If the null space is specified
-- (the default values), then the names of all spaces are printed.
-- If the directory system is being searched, Vol_Hint /= 0 will
-- cause the search to attempt to avoid looking on the wrong volume.
-- Root_Name specifies where the directory system search should begin.

procedure Name_Space (Vp        : Natural    := 0;
                      Kind      : Space_Kind := Disk_Space.Instruction;
                      Segment   : Natural    := 0;
                      Vol_Hint  : Natural    := 0;
                      Root_Name : String     := "!";
                      Search    : Traversing := Disk_Space.All_Traversals;
                      Verbose   : Boolean    := True);


-- Searches just as with Name_Space, but will search for any space
-- with the same Family_Id as the space specified.

procedure Name_Family (Vp        : Natural    := 0;
                       Kind      : Space_Kind := Disk_Space.Instruction;
                       Segment   : Natural    := 0;
                       Vol_Hint  : Natural    := 0;
                       Root_Name : String     := "!";
                       Search    : Traversing := Disk_Space.All_Traversals;
                       Verbose   : Boolean    := True);


-- Interpret page 0 of the data segment in various ways.
-- Instruction spaces don't have data segments, so only useful for Modules.
```





```
    procedure Decode_Space (Vp      : Natural    := 0;
                            Kind    : Space_Kind := Disk_Space.Module;
                            Segment : Natural    := 0;
                            Match   : Classes    := Disk_Space.Matching_Class;
                            Verbose : Boolean    := True);


    -- *****************************************************************
    -- Do not use the following commands unless you know what you are doing.
    -- *****************************************************************

    type Mark_Type     is new Natural range 0 .. 1023;
    type Volume_Number is new Natural range 0 .. 31;
    type Block_Number  is new Natural range 0 .. 2 ** 24 - 1;

    type Usage_Array_Type is array (Mark_Type) of Natural;

    type Vol_Bit_Map_Array  is array (Block_Number range <>) of Boolean;
    type Vol_Usage_Array    is array (Block_Number range <>) of Mark_Type;
    type System_Usage_Array is
       array (Volume_Number range <>) of Usage_Array_Type;

    Unable_To_Acquire_Backup_Lock : exception;
    Garbage_Collection_Is_Running : exception;

    function First_Volume return Volume_Number;
    function Last_Volume  return Volume_Number;

    function Find_Storage_Consumed return System_Usage_Array;

    procedure Clean_Cache;

    function Get_Bit_Map (Volume : Volume_Number) return Vol_Bit_Map_Array;

    function Find_Current_Usage (Volume : Volume_Number) return Vol_Usage_Array;

end Disk_Space;
```

```
procedure Do_Step (Step      : String := "";
                   Step_File : String := "Command_Data.Steps");

pragma Loaded_Main;

--
-- This procedure is used by Rational in performing installation
-- of Rational products.
--
-- A file contains a list of valid step names and values.  See package
-- Parameter_Parser for legal format of the form
--         <STEP NAME> => <COMMANDS>
-- The Step parameter lists a set of Steps to be executed.  For example,
-- if the following steps are defined in the step file:
--     HELLO => ( Io.Put_Line ("Hello"); )
--     FACTORIAL => (
--         declare
--             N : Natural := 1;
--         begin
--             for I in 1 .. 4 loop
--                 N := N * I;
--             end loop;
--             Io.Echo (N);
--         end;
--     )
--     GOODBY => ( Io.Put_Line ("Goodby");  )
-- you execute Do_Install (Step => "Hello"), the string "Hello" would
-- be output to your I/O window.
--
-- To execute this procedure, the user must be a member of group privileged.
-- During execution of steps, privileged mode is enabled.
--
-- Errors (***, ++*, %%%) signify problems with a step which must be
-- resolved before continuing on to the next step.  Warnings (!!!)
-- should be resolved, yet it is permissible to continue on to the
-- next step at your own risk.
--
-- Predefined "steps"
--      LOAD_TAPE
--              Performs an Archive.Restore for a tape mounted on the
--              system tape drive.  Options to Archive are "REPLACE, PROMOTE"
--      TRACE   (boolean value) can be used to display the step
--              which is executed.  Default = FALSE.
--      EXECUTE (Boolean value) can be used to enable/disable execution.
--              Typically used in conjunction with TRACE to only display
--              the command that will be executed without actually executing.
--      PROMPT  Formats a step in a command window for manual execution.
```





```
--              Of the form PROMPT => <Step Name>.  Any steps after this one
--              will not be executed, and the procedure terminates at this point
--              allowing the user to make modifications to the command window
--              and then execute it.  For example:
--                  Do_Install ("PROMPT => Hello");
--              would result in a command window being created and the
--              contents, ready for execution, looking something like:
--
--              declare
--              begin
--                  Operator.Enable_Privileges;
--                  Io.Put_Line ("Hello");
--              end;
--
--      SEMANTICIZE (boolean value)
--              Does a semantic check on the step, reporting the results in
--              the log output.  Used for verifying master step file.
--              Default = FALSE.  For example, "Semanticize Hello".
--
--      PAUSE (boolean value)
--              When executing multiple steps, will pause after execution
--              of each step and prompt the user for continuation.  If
--              anything other than 'Y' or 'y' (or "" string indicating the
--              default) is entered, then no more steps are executed and
--              Do_Step halts at that point.  The default for PAUSE is
--              TRUE; to always pause between steps and ask for
--              confirmation to continue with the next step.
--
-- Step execution is done by calling Program.Run on the image of the step.
```

```
package Editor is

    package Cursor is
        procedure Down  (Repeat : Integer := 1);
        procedure Left  (Repeat : Integer := 1);
        procedure Right (Repeat : Integer := 1);
        procedure Up    (Repeat : Integer := 1);
        -- Quarter-plane motion

        procedure Forward  (Repeat : Integer := 1);
        procedure Backward (Repeat : Integer := 1);
        -- Stream motion, end of line N adjacent to beginning of line N+1

        procedure Next     (Repeat    : Integer := 1;
                            Prompt    : Boolean := True;
                            Underline : Boolean := True);
        procedure Previous (Repeat    : Integer := 1;
                            Prompt    : Boolean := True;
                            Underline : Boolean := True);
        -- Position the cursor at the next (previous) closest prompt or
        -- underline.  Prompt (Underline) false indicates not to look
        -- for the next Prompt (Underline).  Both false does nothing
    end Cursor;

    package Search is
        procedure Previous (Target : String := ""; Wildcard : Boolean := False);
        procedure Next (Target : String := ""; Wildcard : Boolean := False);
        procedure Replace_Previous (Target      : String  := "";
                                    Replacement : String  := "";
                                    Repeat      : Integer := 1;
                                    Wildcard    : Boolean := False);
        procedure Replace_Next (Target      : String  := "";
                                Replacement : String  := "";
                                Repeat      : Integer := 1;
                                Wildcard    : Boolean := False);
    end Search;

    package Char is
        procedure Capitalize      (Repeat : Integer := 1);
        procedure Delete_Backward (Repeat : Integer := 1);
        procedure Delete_Forward  (Repeat : Integer := 1);
        -- Stream deletion end of line N is adjacent to beginning
        -- of line N+1

        procedure Delete_Next     (Repeat : Integer := 1);
        procedure Delete_Previous (Repeat : Integer := 1);
        -- Quarter-plane deletion
```





```
        procedure Delete_Spaces (Remaining : Natural := 1);
        -- Delete spaces surrounding the cursor, leaving remaining spaces

        procedure Insert_String    (Value : String);
        procedure Insert_Character (Repeat : Integer := 1; Value : Character);
        procedure Lower_Case       (Repeat : Integer := 1);
        procedure Quote;
        procedure Tab_Backward     (Repeat : Integer := 1);
        procedure Tab_Forward      (Repeat : Integer := 1);
        procedure Tab_To_Comment;
        -- Tab to the comment column and insert comment marks

        procedure Transpose  (Offset : Integer := 1);
        procedure Upper_Case (Repeat : Integer := 1);
    end Char;

    package Line is
        procedure Beginning_Of    (Offset : Natural := 0);
        procedure Capitalize      (Repeat : Integer := 1);
        procedure Center          (Right_Margin : Natural := 0);
        procedure Copy            (Repeat : Integer := 1);
        procedure Delete          (Repeat : Integer := 1);
        procedure Delete_Backward (Repeat : Integer := 1);
        procedure Delete_Forward  (Repeat : Integer := 1);
        procedure End_Of          (Offset : Natural := 0);
        procedure Insert          (Repeat : Integer := 1);
        procedure Indent          (Repeat : Integer := 1);
        procedure Join            (Repeat : Integer := 1);
        procedure Lower_Case      (Repeat : Integer := 1);
        procedure Open            (Repeat : Integer := 1);
        procedure Transpose       (Offset : Integer := 1);
        procedure Upper_Case      (Repeat : Integer := 1);
        procedure Next            (Repeat : Integer := 1) renames Cursor.Down;
        procedure Previous        (Repeat : Integer := 1) renames Cursor.Up;
    end Line;

    package Word is
        procedure Beginning_Of;
        procedure Breaks          (Break_Set      : String  := "";
                                   Are_Delimiters : Boolean := True);
        procedure Capitalize      (Repeat : Integer := 1);
        procedure End_Of;
        procedure Delete          (Repeat : Integer := 1);
        procedure Delete_Backward (Repeat : Integer := 1);
        procedure Delete_Forward  (Repeat : Integer := 1);
        procedure Lower_Case      (Repeat : Integer := 1);
        procedure Next            (Repeat : Integer := 1);
        procedure Previous        (Repeat : Integer := 1);
        procedure Transpose       (Offset : Integer := 1);
```

```
      procedure Upper_Case      (Repeat : Integer := 1);
   end Word;

   package Image is
      -- repeat = 0 scrolls one page

      procedure Up           (Repeat : Integer := 0);
      procedure Down         (Repeat : Integer := 0);
      procedure Left         (Repeat : Integer := 0);
      procedure Right        (Repeat : Integer := 0);
      procedure Find         (Name : String);
      procedure Beginning_Of (Offset : Natural := 0);
      procedure End_Of       (Offset : Natural := 0);
   end Image;


   -- Many of the following packages implement a "stack" discipline.  For
   -- these packages, the following operations are supported:
   --
   -- Copy_Top     Push a copy of the top of stack
   -- Delete_Top   Delete the top element from the stack
   -- Next         Use the next value on the stack
   -- Previous     Use the previous value on the stack
   -- Push         Put the appropriate item on the stack
   -- Rotate       Rotate the stack; top becomes the bottom; value not
   --              used
   -- Swap         Interchange the top and next to top items; value not
   --              used
   -- Top          Use the top value on the stack

   package Screen is
      procedure Down  (Repeat : Integer := 1);
      procedure Left  (Repeat : Integer := 1);
      procedure Right (Repeat : Integer := 1);
      procedure Up    (Repeat : Integer := 1);
      procedure Dump  (To_File : String := ">>NAME<<");
      procedure Redraw;
      procedure Clear;
      -- Screen stack operations

      procedure Copy_Top;
      procedure Delete_Top;
      procedure Next     (Repeat : Integer := 1);
      procedure Previous (Repeat : Integer := 1);
      procedure Push     (Repeat : Integer := 1);
      procedure Rotate   (Repeat : Integer := 1);
      procedure Swap;
      procedure Top;
```





```
      -- Set terminal lines and columns for this session.
      -- Changes take effect at Set_Lines calls.
      procedure Set_Columns (Columns : Natural);
      procedure Set_Lines   (Lines : Natural);
   end Screen;

   package Window is
      procedure Beginning_Of (Offset : Natural := 0);
      procedure Child        (Repeat : Integer := 1);
      procedure Copy;
      procedure Delete;
      procedure Demote;
      procedure Directory;
      procedure End_Of       (Offset : Natural := 0);
      procedure Expand       (Lines : Integer := 4);
      procedure Focus;
      procedure Frames       (Maximum : Positive);
      procedure Join         (Repeat : Integer := 1);
      procedure Next         (Repeat : Integer := 1);
      procedure Parent       (Repeat : Integer := 1);
      procedure Previous     (Repeat : Integer := 1);
      procedure Promote;
      procedure Transpose    (Offset : Integer := 1);
   end Window;

   package Macro is
      procedure Start;
      procedure Finish;
      -- Start/Finish the definition of a keyboard macro

      procedure Execute (Repeat : Integer := 1; Prior : Natural := 0);
      -- Execute the current keyboard macro Repeat times.  If Prior /= 0
      -- execute the macro with that number.

      procedure Bind (Key : String := "");
      -- bind the current macro to the key name given, e.g. F1, M_F1.

      procedure Save (Expanded : Boolean := False);
      -- Save the current macro state in the user macro file.
      -- Expanded causes the file string to be saved in text form.

      procedure Restore;
      -- Recreate macro state from the user macro file.

   end Macro;

   package Hold_Stack is
      procedure Copy_Top;
      procedure Delete_Top;
```

```
      procedure Next     (Repeat : Integer := 1);
      procedure Previous (Repeat : Integer := 1);
      procedure Push     (Repeat : Integer := 1);
      procedure Rotate   (Repeat : Integer := 1);
      procedure Swap;
      procedure Top;
   end Hold_Stack;

   package Mark is
      procedure Copy_Top;
      procedure Delete_Top;
      procedure Next     (Repeat : Integer := 1);
      procedure Previous (Repeat : Integer := 1);
      procedure Push     (Repeat : Integer := 1);
      procedure Rotate   (Repeat : Integer := 1);
      procedure Swap;
      procedure Top;
   end Mark;

   package Region is
      procedure Beginning_Of;
      procedure Capitalize;
      procedure Comment;
      -- Add comment marks to the beginning of the lines in the region
      procedure Copy;
      procedure Delete;
      procedure End_Of;
      procedure Fill    (Column : Natural := 0; Leading : String := "");
      procedure Finish;
      procedure Justify (Column : Natural := 0; Leading : String := "");
      -- 0 argument uses default fill column
      procedure Lower_Case;
      procedure Move;
      procedure Off;
      procedure On;
      procedure Start;
      procedure Uncomment;
      procedure Upper_Case;
   end Region;

   package Set is
      procedure Insert_Mode      (On : Boolean := True);
      procedure Fill_Mode        (On : Boolean := True);
      procedure Fill_Column      (Column : Positive := 72);
      procedure Designation_Off;
      procedure Input_From       (File_Name : String := "<SELECTION>");
      procedure Input_Logging_To (File_Name : String := ">>Name<<");
      procedure Input_Logging_Off;
      procedure Tab_Off          (Column : Positive);
```





```
      procedure Tab_On           (Column : Positive);
      procedure Tab_Width        (Size : Positive := 4);
      -- Only to be bound on keys

      procedure Argument_Prefix;
      procedure Argument_Digit (Argument : Integer := 1);
      procedure Argument_Minus;
   end Set;

   package Key is
      procedure Define (Key_Name     : String  := ">>KEY NAME, e.g. CM_F1<<";
                        Command_Name : String  := ">>COMMAND NAME<<";
                        Prompt       : Boolean := False);
      procedure Name   (Key_Code : String := "");
      procedure Save;
      procedure Prompt (Key_Code : String := "");
   end Key;

   procedure Quit (Ignore_Changes : Boolean := False);
   procedure Alert;
   procedure Noop;

end Editor;
```

```
package File_Utilities is

    subtype Name is String;
    Current_Output : constant Name := "";

    procedure Difference (File_1            : Name    := "<REGION>";
                          File_2            : Name    := "<IMAGE>";
                          Result            : Name    := "";
                          Compressed_Output : Boolean := False;
                          Subobjects        : Boolean := False);
    -- Find differences between two versions of an object.
    -- If Subobjects is True, subobjects are compared as well.
    -- Compressed output omits lines that are the same in both objects.
    -- Non-compressed output shows every line from both objects,
    -- only showing common lines once.

    procedure Merge (Original : Name := "";
                     File_1   : Name := "";
                     File_2   : Name := "";
                     Result   : Name := "");
    -- merge two variants of the same object into new version with all changes
    -- Result defaults to Current_Output = ""

    procedure Strip (Source : Name := "<SELECTION>"; Target : Name := "");
    -- take the output of Merge or Difference and create a clean file

    procedure Compare (File_1      : Name    := "<REGION>";
                       File_2      : Name    := "<IMAGE>";
                       Subobjects  : Boolean := False;
                       Ignore_Case : Boolean := False;
                       Options     : String  := "");
    -- find the first difference between two objects
    -- Subobjects=true causes subunits or units in a library to be compared
    -- as well as the named units.
    -- Ignore_Case=true causes upper and lower case to be treated as
    -- equivalent.
    -- Options include: Ignore_Blank_Lines:   causes only on-blank lines
    --                                        to be considered in the compare
    --                  File_2_Has_Wildcards: Interpret characters in File_2
    --                                        as possible Wildcards.  Wildcard
    --                                        characters include:
    --                                          ^ - negate next char
    --                                          ? - match any char
    --                                          % - match any Ada ident char
    --                                          $ - match any Ada delimiter
    --                                          \ - quotes next char
    --                                          { - beginning of line
```





```
    --                                          } - end of line
    --                                          [ - start of class
    --                                          ] - end of class
    --                                          * - zero or more of prev item
    --
    -- Use of Ignore_Case or Ignore_Blank_Lines slows the compare operation
    -- moderately with respect to a straight compare.  File_2_Has_Wildcards
    -- slows the compare dramatically and should only be used if you have
    -- a lot of time to wait.  The wildcard compare is conducted on a line-
    -- by-line basis.

    function Equal (File_1      : Name    := "<REGION>";
                    File_2      : Name    := "<IMAGE>";
                    Subobjects  : Boolean := False;
                    Ignore_Case : Boolean := False;
                    Options     : String  := "") return Boolean;
    -- Indicates whether the two files are the same
    -- See notes under Compare, above.

    procedure Find  (Pattern     : String  := "";
                     File        : Name    := "<IMAGE>";
                     Wildcards   : Boolean := False;
                     Ignore_Case : Boolean := True;
                     Result      : Name    := "");
    function  Found (Pattern     : String  := "";
                     File        : Name    := "<IMAGE>";
                     Wildcards   : Boolean := False;
                     Ignore_Case : Boolean := True) return Natural;
    -- find instances of Pattern in File, optionally using Wildcards

    procedure Append (Source : Name := ""; Target : Name := "<SELECTION>");
    -- append the contents of one file to another

    procedure Dump (File        : Name     := "<SELECTION>";
                    Page_Number : Natural  := 0;
                    Word_Number : Natural  := 0;
                    Word_Count  : Positive := 64);
    -- display a hex dump of the file.  A "word" is 16 bytes.
    -- Defaults dump the first page of the file.

    procedure Sort (File   : Name   := "<IMAGE>";
                    Result : Name   := "";
                    Key_1  : String := "";
                    Key_2  : String := "";
                    Key_3  : String := "");
    --
    -- Sort File using Key_n as sort keys.
    --
    -- Key_1 is most signficant.  Key_2 is ignored if Key_1 not specified, etc.
```

```
    -- No keys cause ascending Ascii sort on full-line compare.
    --
    -- Key_n follow form parameter syntax, parameters are first-character
    -- unique, so any prefix of the names is sufficient.
    --
    --     FIELD => number
    --
    --         Field is a field on the line.  Fields are non-blank characters
    --         separated by blanks.  Field 1 is the first field.  Field 1
    --         always includes column 1, even if blank.  If no field is given,
    --         the entire line, blanks included is the field.
    --
    --     START_COLUMN => number (default is 1)
    --
    --         The starting column relative to the start of the field.
    --
    --     END_COLUMN => number (default is Integer'Last)
    --
    --         The ending column relative to the start of the field.
    --
    --     REVERSE => true | FALSE
    --
    --         True implies sort descending for this key.
    --
    --     NUMERIC => true | FALSE
    --
    --         Perform the sort on the numeric value of the field represented
    --         as a Long_Integer.
    --
    -- Examples:
    --
    --     "F=2, S=5, E=7, R, N" will sort the field 2, columns 5 through 7,
    --     descending (reversed) using a numeric comparison.  Fully specified,
    --     "Field => 2, Start_Column => 5, End_Column => 7, Reversed, Numeric"
    --
    --     "S=10, E=>15" will sort using Ascii ordering columns 10 through 15
    --     of the entire line.

end File_Utilities;
```





```
with Profile;
with Ftp_Defs;
with Ftp_Profile;
with Ftp_Name_Map;
with File_Transfer;

package Ftp is

    function Profile_Get return Profile.Response_Profile renames Profile.Get;
    -- This declaration is introduced to avoid a name collision.

    ----|| non-interactive transfers ||----

    -- Move a file to some other machine:

    procedure Put
                  (From_Local_File    : String := "<IMAGE>";
                   To_Remote_File     : String := "";
                   Remote_Machine     : String := Ftp_Profile.Remote_Machine;
                   Username           : String := Ftp_Profile.Username;
                   Password           : String := Ftp_Profile.Password;
                   Account            : String := Ftp_Profile.Account;
                   Remote_Directory   : String := Ftp_Profile.Remote_Directory;
                   Remote_Type        : Ftp_Name_Map.Machine_Type :=
                      Ftp_Profile.Remote_Type;
                   Append_To_File     : Boolean := False;
                   Transfer_Type      : Ftp_Defs.Type_Code :=
                      Ftp_Profile.Transfer_Type;
                   Transfer_Mode      : Ftp_Defs.Mode_Code :=
                      Ftp_Profile.Transfer_Mode;
                   Transfer_Structure : Ftp_Defs.Structure_Code :=
                      Ftp_Profile.Transfer_Structure;
                   Send_Port          : Boolean := Ftp_Profile.Send_Port_Enabled;
                   Response           : Profile.Response_Profile := Profile.Get);

    -- Get a file from some other machine:

    procedure Get
                  (From_Remote_File   : String                    := "";
                   To_Local_File      : String                    := "";
                   Remote_Machine     : String                    :=
                      Ftp_Profile.Remote_Machine;
                   Username           : String                    :=
                      Ftp_Profile.Username;
                   Password           : String                    :=
                      Ftp_Profile.Password;
                   Account            : String                    :=
```

```
        Ftp_Profile.Account;
      Remote_Directory    : String                    :=
        Ftp_Profile.Remote_Directory;
      Remote_Type         : Ftp_Name_Map.Machine_Type :=
        Ftp_Profile.Remote_Type;
      Append_To_File      : Boolean                   := False;
      Transfer_Type       : Ftp_Defs.Type_Code        :=
        Ftp_Profile.Transfer_Type;
      Transfer_Mode       : Ftp_Defs.Mode_Code        :=
        Ftp_Profile.Transfer_Mode;
      Transfer_Structure  : Ftp_Defs.Structure_Code   :=
        Ftp_Profile.Transfer_Structure;
      Send_Port           : Boolean                   :=
        Ftp_Profile.Send_Port_Enabled;
      Response            : Profile.Response_Profile  :=
        Ftp.Profile_Get);


-- The following operations are more interactive, that is,
-- they require that you first establish a connection to
-- a remote machine, and then interact with it to transfer
-- files.  Using FTP interactively allows you to do more
-- things than you can do non-interactively.

----|| login operations ||----

procedure Connect
            (To_Machine          : String := Ftp_Profile.Remote_Machine;
             Auto_Login          : Boolean := Ftp_Profile.Auto_Login;
             Username            : String := Ftp_Profile.Username;
             Password            : String := Ftp_Profile.Password;
             Account             : String := Ftp_Profile.Account;
             Remote_Directory    : String := Ftp_Profile.Remote_Directory;
             Remote_Roof         : String := Ftp_Profile.Remote_Roof;
             Remote_Type         : Ftp_Name_Map.Machine_Type :=
                Ftp_Profile.Remote_Type;
             Transfer_Type       : Ftp_Defs.Type_Code :=
                Ftp_Profile.Transfer_Type;
             Transfer_Mode       : Ftp_Defs.Mode_Code :=
                Ftp_Profile.Transfer_Mode;
             Transfer_Structure  : Ftp_Defs.Structure_Code :=
                Ftp_Profile.Transfer_Structure;
             Send_Port           : Boolean := Ftp_Profile.Send_Port_Enabled;
             Response            : Profile.Response_Profile := Profile.Get);

procedure Login
            (Username            : String := Ftp_Profile.Username;
             Password            : String := Ftp_Profile.Password;
             Account             : String := Ftp_Profile.Account;
```





```
             Remote_Directory    : String := Ftp_Profile.Remote_Directory;
             Remote_Roof         : String := Ftp_Profile.Remote_Roof;
             Remote_Type         : Ftp_Name_Map.Machine_Type :=
                Ftp_Profile.Remote_Type;
             Transfer_Type       : Ftp_Defs.Type_Code :=
                Ftp_Profile.Transfer_Type;
             Transfer_Mode       : Ftp_Defs.Mode_Code :=
                Ftp_Profile.Transfer_Mode;
             Transfer_Structure  : Ftp_Defs.Structure_Code :=
                Ftp_Profile.Transfer_Structure;
             Send_Port           : Boolean := Ftp_Profile.Send_Port_Enabled;
             Response            : Profile.Response_Profile := Profile.Get);

procedure Disconnect (Response : Profile.Response_Profile := Profile.Get);

procedure Abandon (Response : Profile.Response_Profile := Profile.Get);

-- Like Disconnect, but a bit stronger:
-- Guaranteed to terminate the connection,
-- even if the remote server is unresponsive.

----|| transfer parameter selection ||----

-- The following procedures override the FTP_Profile values,
-- but ONLY for the duration of the current connection.
-- To modify the FTP_Profile values, see FTP_Profile.Set.

procedure Use_Type
            (Value    : Ftp_Defs.Type_Code := Ftp_Profile.Transfer_Type;
             Response : Profile.Response_Profile := Profile.Get;
             Account  : String := Ftp_Profile.Account);

procedure Use_Mode
            (Value    : Ftp_Defs.Mode_Code := Ftp_Profile.Transfer_Mode;
             Response : Profile.Response_Profile := Profile.Get;
             Account  : String := Ftp_Profile.Account);

procedure Use_Structure
            (Value    : Ftp_Defs.Structure_Code  :=
                Ftp_Profile.Transfer_Structure;
             Response : Profile.Response_Profile := Profile.Get;
             Account  : String                   := Ftp_Profile.Account);

procedure Use_Account
            (Account  : String                   := Ftp_Profile.Account;
             Response : Profile.Response_Profile := Profile.Get);

procedure Send_Port (Enabled  : Boolean := Ftp_Profile.Send_Port_Enabled;
                     Response : Profile.Response_Profile := Profile.Get);
```

```
procedure Use_Remote_Type
             (Value : Ftp_Name_Map.Machine_Type := Ftp_Profile.Remote_Type;
              Response : Profile.Response_Profile := Profile.Get);

procedure Use_Remote_Roof
             (Value    : String := Ftp_Profile.Remote_Roof;
              Response : Profile.Response_Profile := Profile.Get);

function Current_Remote_Type return Ftp_Name_Map.Machine_Type
    renames Ftp_Profile.Current_Remote_Type;
function Current_Remote_Roof return String
    renames Ftp_Profile.Current_Remote_Roof;
function Current_Connection  return File_Transfer.Connect_Id;
-- Returns the File_Transfer connection associated with the
-- caller's current session.  File_Transfer operations, such
-- as querying the outcome of the last transaction, can be
-- performed on the resulting value.

----|| remote directory operations ||----

procedure Change_Working_Directory
             (Remote_Directory : String := Ftp_Profile.Remote_Directory;
              Response         : Profile.Response_Profile := Profile.Get;
              Account          : String := Ftp_Profile.Account);

procedure Cwd (Remote_Directory : String := Ftp_Profile.Remote_Directory;
               Response         : Profile.Response_Profile := Profile.Get;
               Account          : String := Ftp_Profile.Account)
    renames Change_Working_Directory;

procedure List (Remote_Pathname : String := "";
                Verbose         : Boolean := True;
                To_Local_File   : String :=
                   "";       -- default is current output
                Response        : Profile.Response_Profile := Profile.Get;
                Account         : String := Ftp_Profile.Account);

procedure Delete (Remote_File : String;
                  Response    : Profile.Response_Profile := Profile.Get;
                  Account     : String := Ftp_Profile.Account);


----|| status operations ||----

procedure Status (Argument : String                      := "";
                  Response : Profile.Response_Profile := Profile.Get);
-- Display the status of the current connection.
```





```
procedure Status_All (Argument : String                      := "";
                      Response : Profile.Response_Profile := Profile.Get);
-- Display the status of all current connections.

procedure Remote_Status (Argument : String := "";
                         Response : Profile.Response_Profile := Profile.Get;
                         Account  : String := Ftp_Profile.Account);

procedure Remote_Help (Argument : String := "";
                       Response : Profile.Response_Profile := Profile.Get;
                       Account  : String := Ftp_Profile.Account);

procedure Show_Profile (Response : Profile.Response_Profile := Profile.Get)
    renames Ftp_Profile.Show;
-- Display the settings in the current profile.

----|| file transfer operations ||----

procedure Store (From_Local_File : String := "<IMAGE>";
                 To_Remote_File  : String := "";
                 Append_To_File  : Boolean := False;
                 Response        : Profile.Response_Profile := Profile.Get;
                 Remote_Type     : Ftp_Name_Map.Machine_Type :=
                    Ftp.Current_Remote_Type;
                 Account         : String := Ftp_Profile.Account);

procedure Retrieve
             (From_Remote_File : String := "";
              To_Local_File    : String := "";
              Append_To_File   : Boolean := False;
              Response         : Profile.Response_Profile := Profile.Get;
              Remote_Type      : Ftp_Name_Map.Machine_Type :=
                 Ftp.Current_Remote_Type;
              Account          : String := Ftp_Profile.Account);

----|| transferring file sets ||----

-- These operations are like their single-file cousins,
-- above, except that they take wildcards, and move a
-- bunch of files at a crack.  They're designed to support
-- moving files between a subtree of the local directory
-- system (whose root is designated by "Local_Roof"), to
-- and from an isomorphic subtree of the remote directory
-- system (whose root is designated by "Remote_Roof".
-- File names are transformed between the naming conventions
-- of the R1000 and whatever remote operating system is
-- involved (designated by Remote_Type).  The transformation
-- is complex: it is encapsulated in package FTP_Name_Map.
```

```
-- If you want to flatten a file_set, that is, move files from
-- many directories on one machine into a single directory on
-- another, specify "" as the source roof (local_roof for Put,
-- remote_roof for Get).

-- Get_List and Retrieve_List retrieve files using a list
-- stored in a file on your local machine.  This file must
-- contain text, with one fully-qualified file name (in the
-- form used by the REMOTE machine) per line.  No comments,
-- no extra white space.

-- Get_Set and Retrieve_Set depend on the remote machine to
-- produce a list of files from the File_Set you specify:
-- using the same primitive as List (Verbose => False).
-- Sometimes this doesn't work, either because the wild
-- card semantics on that machine won't stretch, or because
-- the remote FTP server doesn't support wild cards.

procedure Put_Set
              (From_Local_File_Set : String := "<IMAGE>";
               Local_Roof : String := "$";
               Remote_Roof : String := Ftp_Profile.Remote_Roof;
               Remote_Machine : String := Ftp_Profile.Remote_Machine;
               Username : String := Ftp_Profile.Username;
               Password : String := Ftp_Profile.Password;
               Account : String := Ftp_Profile.Account;
               Remote_Directory : String := Ftp_Profile.Remote_Directory;
               Remote_Type : Ftp_Name_Map.Machine_Type :=
                  Ftp_Profile.Remote_Type;
               Append_To_File : Boolean := False;
               Transfer_Type : Ftp_Defs.Type_Code :=
                  Ftp_Profile.Transfer_Type;
               Transfer_Mode : Ftp_Defs.Mode_Code :=
                  Ftp_Profile.Transfer_Mode;
               Transfer_Structure : Ftp_Defs.Structure_Code :=
                  Ftp_Profile.Transfer_Structure;
               Send_Port : Boolean := Ftp_Profile.Send_Port_Enabled;
               Response : Profile.Response_Profile := Profile.Get);

procedure Get_Set
              (From_Remote_File_Set : String := "";
               Local_Roof : String := "$";
               Remote_Roof : String := Ftp_Profile.Remote_Roof;
               Remote_Machine : String := Ftp_Profile.Remote_Machine;
               Username : String := Ftp_Profile.Username;
               Password : String := Ftp_Profile.Password;
               Account : String := Ftp_Profile.Account;
               Remote_Directory : String := Ftp_Profile.Remote_Directory;
               Remote_Type : Ftp_Name_Map.Machine_Type :=
```





```
                  Ftp_Profile.Remote_Type;
               Append_To_File : Boolean := False;
               Transfer_Type : Ftp_Defs.Type_Code :=
                  Ftp_Profile.Transfer_Type;
               Transfer_Mode : Ftp_Defs.Mode_Code :=
                  Ftp_Profile.Transfer_Mode;
               Transfer_Structure : Ftp_Defs.Structure_Code :=
                  Ftp_Profile.Transfer_Structure;
               Send_Port : Boolean := Ftp_Profile.Send_Port_Enabled;
               Response : Profile.Response_Profile := Profile.Get);

procedure Get_List
              (Remote_File_List   : String := "";
               Local_Roof         : String := "$";
               Remote_Roof        : String := Ftp_Profile.Remote_Roof;
               Remote_Machine     : String := Ftp_Profile.Remote_Machine;
               Username           : String := Ftp_Profile.Username;
               Password           : String := Ftp_Profile.Password;
               Account            : String := Ftp_Profile.Account;
               Remote_Directory   : String := Ftp_Profile.Remote_Directory;
               Remote_Type        : Ftp_Name_Map.Machine_Type :=
                  Ftp_Profile.Remote_Type;
               Append_To_File     : Boolean := False;
               Transfer_Type      : Ftp_Defs.Type_Code :=
                  Ftp_Profile.Transfer_Type;
               Transfer_Mode      : Ftp_Defs.Mode_Code :=
                  Ftp_Profile.Transfer_Mode;
               Transfer_Structure : Ftp_Defs.Structure_Code :=
                  Ftp_Profile.Transfer_Structure;
               Send_Port          : Boolean := Ftp_Profile.Send_Port_Enabled;
               Response           : Profile.Response_Profile := Profile.Get);

procedure Store_Set
              (From_Local_File_Set : String := "<IMAGE>";
               Local_Roof          : String := "$";
               Remote_Roof         : String := Ftp.Current_Remote_Roof;
               Remote_Type         : Ftp_Name_Map.Machine_Type :=
                  Ftp.Current_Remote_Type;
               Append_To_File      : Boolean := False;
               Response            : Profile.Response_Profile := Profile.Get;
               Account             : String := Ftp_Profile.Account);

procedure Retrieve_Set
              (From_Remote_File_Set : String := "";
               Local_Roof : String := "$";
               Remote_Roof : String := Ftp.Current_Remote_Roof;
               Remote_Type : Ftp_Name_Map.Machine_Type :=
                  Ftp.Current_Remote_Type;
               Append_To_File : Boolean := False;
```

```
            Response : Profile.Response_Profile := Profile.Get;
            Account : String := Ftp_Profile.Account);

   procedure Retrieve_List
            (Remote_File_List : String := "";
             Local_Roof       : String := "$";
             Remote_Roof      : String := Ftp.Current_Remote_Roof;
             Remote_Type      : Ftp_Name_Map.Machine_Type :=
                Ftp.Current_Remote_Type;
             Append_To_File   : Boolean := False;
             Response         : Profile.Response_Profile := Profile.Get;
             Account          : String := Ftp_Profile.Account);


   ----|| constants of imported types ||----

   Ascii         : constant Ftp_Defs.Type_Code := Ftp_Defs.Ascii;
   Ebcdic        : constant Ftp_Defs.Type_Code := Ftp_Defs.Ebcdic;
   Image         : constant Ftp_Defs.Type_Code := Ftp_Defs.Image;
   Binary        : constant Ftp_Defs.Type_Code := Ftp_Defs.Binary;
   Local_Binary  : constant Ftp_Defs.Type_Code := Ftp_Defs.Local_Binary;
   Local_Byte    : constant Ftp_Defs.Type_Code := Ftp_Defs.Local_Byte;
   Ascii_Cc      : constant Ftp_Defs.Type_Code := Ftp_Defs.Ascii_Cc;
   Ebcdic_Cc     : constant Ftp_Defs.Type_Code := Ftp_Defs.Ebcdic_Cc;
   Ascii_Telnet  : constant Ftp_Defs.Type_Code := Ftp_Defs.Ascii_Telnet;
   Ebcdic_Telnet : constant Ftp_Defs.Type_Code := Ftp_Defs.Ebcdic_Telnet;

   Stream     : constant Ftp_Defs.Mode_Code := Ftp_Defs.Stream;
   Block      : constant Ftp_Defs.Mode_Code := Ftp_Defs.Block;
   Compressed : constant Ftp_Defs.Mode_Code := Ftp_Defs.Compressed;

   File  : constant Ftp_Defs.Structure_Code := Ftp_Defs.File;
   Recrd : constant Ftp_Defs.Structure_Code := Ftp_Defs.Recrd;
   Page  : constant Ftp_Defs.Structure_Code := Ftp_Defs.Page;

   Rational : constant Ftp_Name_Map.Machine_Type := Ftp_Name_Map.Rational;
   R1000    : constant Ftp_Name_Map.Machine_Type := Rational;
   Unix     : constant Ftp_Name_Map.Machine_Type := Ftp_Name_Map.Unix;
   Aos      : constant Ftp_Name_Map.Machine_Type := Ftp_Name_Map.Aos;
   Mv       : constant Ftp_Name_Map.Machine_Type := Aos;
   Vms      : constant Ftp_Name_Map.Machine_Type := Ftp_Name_Map.Vms;
   Vax      : constant Ftp_Name_Map.Machine_Type := Vms;
   Mvs      : constant Ftp_Name_Map.Machine_Type := Ftp_Name_Map.Mvs;

end Ftp;
```





```
package Gateway is

    procedure Create (Name              : String := ">>OBJECT NAME<<";
                      Gateway_Class     : String := ">>GATEWAY CLASS<<";
                      Value_Assignments : String := "";
                      Options           : String := "";
                      Response          : String := "<PROFILE>");
    --
    -- Create a Gateway object of the specified gateway class.
    -- Value_Assignments provides a means of assigning properties
    -- initial values.  The string consists of zero or more property
    -- specifications separated by commas or semicolons.  Each property
    -- specification is of the form:
    --
    --      Property_Name => Value
    -- or
    --      Property_Name(Subobject_Name) => Value
    --
    -- where Property_Name is the name of the gateway property to be assigned
    -- a value (it may include embedded "." characters).  The value is
    -- a value acceptable to the Parameter_Parser which includes numeric
    -- literals, identifiers, directory pathnames, and any sequence of
    -- characters enclosed in balanced '(', ')' characters.  The escape
    -- character is '\' so that, for example, "\)" is an uninterpreted
    -- right parenthesis.
    --
    -- Options specifies any additional parameters for this operation
    -- (primarily for future additions).
    --

    procedure Set_Property (Gateway_Name       : String := "<SELECTION>";
                            Property_Name      : String := ">>PROPERTY NAME<<";
                            New_Property_Value : String := ">>PROPERTY VALUE<<";
                            Response           : String := "<PROFILE>");
    -- Set the named property in the named gateway object(s) to the named
    -- value.  Multiple objects may be specified via directory-name wildcards.

    procedure Edit (Name : String := "<CURSOR>"; In_Place : Boolean := False);
    -- Run editor on properties of a gateway object.

    procedure Display (Name : String := "<CURSOR>"; Properties : String := "@");
    -- Display the properties of the specified gateway object(s) whose name
    -- matches Properties, as follows:
    --   "@", "*", and "" specify all properties;
    --   embedded '@'s and '*'s are treated as wildcards;
    --   and matching is done on the property_name without the subobject name.
```

```
    -- Multiple objects may be specified via directory-name wildcards

    procedure Property_Edit (Name     : String  := "<CURSOR>";
                             In_Place : Boolean := False);
    --
    -- Run the editor on the properties of the specified gateway object.
    -- The "data" image of the object, if any, is not generated or displayed
    -- by this operation.

    -------------------------
    -- Editor operations:
    -------------------------

    procedure Insert (Spec : String := ">>Property_Name := Value<<";
                      Gateway_Object_Name : String := "");
    -- The property values displayed in the specified or current DTIA property
    -- Display Window are changed as indicated.
    -- (Generated in response to Object."I" on a DTIA property Display Window)
    -- The spec string specifies one or more properties to be set; the
    -- syntax is identical that of Create's Value_Assigments parameter.

    procedure Change (Image               : String := ">>New Property Value<<";
                      Gateway_Object_Name : String := "");
    -- The highlighted property in the current or specified DTIA property
    -- display window is changed to the value of the given image.
    -- (Generated in response to Edit on a DTIA property display window.)

    procedure Write_Properties (Gateway_Object_Name : String := "");
    -- The contents of the specified Gateway object's property display window
    -- are stored in the object which is then committed.
    -- A null name causes the current DTIA property display window's
    -- properties are written to its Gateway object.

end Gateway;
```





```
package Gateway_Class is

    procedure Build (Gateway_Class_Directory : String := "<IMAGE>";
                     Gateway_Text_Description : String := "Gateway_Definition";
                     Gateway_Binary_Description : String := "Gateway_Class";
                     Response : String := "<Profile>");
    --
    -- Compile the gateway class description from "Gateway_Definition"
    -- producing the gateway class object "Gateway_Class".
    -- Gateway_Class_Directory specifies the directory in
    -- which the gateway class definitions exists.
    -- Gateway_[Text,Binary]_Description are the source/destination of
    -- the construction.
    --
    -- In general, only the directory name need be specified since the
    -- filenames within !Machine.Gateway_Classes are standardized.
    -- Gateway_Class_Directory is evaluated relative to the context
    -- !Machine.Gateway_Classes  so that a simple name can be used
    -- for the gateway class name.

    procedure Activate (Gateway_Class_Name : String :=
                            ">>SIMPLE GATEWAY CLASS NAME<<";
                        Response           : String := "<PROFILE>");
    --
    -- Activate the gateway class from !Machine.Gateway_Classes.<Class_Name>.
    -- This makes the gateway class available for use on the system.
    --
    -- Open the gateway class definition file in !machine.<Class_Name>.Class.
    -- Place the pointer to it in the in-memory cache and hold the
    -- update lock on the gateway class file.
    --
    -- Start the global server for the gateway class, if there is one.

    procedure Deactivate (Gateway_Class_Name : String :=
                              ">>SIMPLE GATEWAY CLASS NAME<<";
                          Response           : String := "<PROFILE>");
    --
    -- Remove the active class entry and release the lock on the file.
    -- Operations on objects of inactive gateway classes are not
    -- allowed.  In addition, after the gateway class is deactivated, images
    -- of objects of that gateway classe will be removed from screens,
    -- servers for that gateway class will be terminated, and further
    -- operations disallowed until the gateway class is reactivated.

    procedure Display (Gateway_Class_Name : String := "@");
    -- Display a formatted report of registered gateway
    -- classes and information about them whose names match
```

```
    -- the specified naming expression.  The Gateway_Class_Name parameter
    -- is resolved in the context !Machine.Gateway_Classes.

    procedure Boot_Time_Initialization;
    --
    -- Call from !Machine.Initialize.  Activates all gateway classes
    -- for which the file Activate_On_Boot is present in the gateway
    -- class directory.

end Gateway_Class;
```





```
with Machine;
package Job is

    subtype Id is Machine.Job_Id;
    -- start, stop and terminate a job

    procedure Kill    (The_Job : Id; The_Session : String := "");
    procedure Disable (The_Job : Id; The_Session : String := "");
    procedure Enable  (The_Job : Id; The_Session : String := "");
    procedure Interrupt;

    procedure Connect    (The_Job : Id := 0);
    procedure Disconnect (The_Job : Id := 0);

    procedure Set_Termination_Message (S : String := "");

end Job;
```

```
with Profile;
with Compilation;

package Library is

    subtype Name is String;
    -- Lexically and syntactically an Ada Name.

    subtype Simple_Name is String;
    -- A simple Ada name.  Basically, an identifier or operator.

    subtype Context_Name is Name;

    -- Treatment of context.  There is a current context that constitutes
    -- the assumed naming context.  Names are resolved in this context.

    -- The following characters modify the context:
    --  !   specifies the Universe context
    --  $   specifies the enclosing library for the current context.
    --  $$  specifies the enclosing world for the current context.
    --  ^   specifies the parent of the current context.
    --  @   matches any single name segment (or part thereof)
    --  ?   matches 0 or more name segments, only the last of which may be a
    --      world.
    --  ??  matches 0 or more name segments.

    -- The special stings "<IMAGE>", etc., attempt to get the designated
    -- object from the current selection/image.

    -- Note that many commands are recursive by default (they are
    -- recognizable as such by the presence of a Recursive parameter). When
    -- the Recursive parameter is true, all descendents of the specified
    -- objects partake in the operation. When Recursive is false, just the
    -- specified objects partake.

    -- The effects of the Recursive option can also be obtained using "?"
    -- wildcards, but with more writing.  In any case, an object is operated
    -- on only once whether it is introduced by an input parameter or the
    -- recursive option or both.


    Error : exception renames Profile.Error;

    -- Only the single exception Error is raised
```





```
    procedure Resolve (Name_Of      : Name    := "<TEXT>";
                       Target_Name  : Name    := "";
                       Objects_Only : Boolean := True;
                       Response     : String  := "<PROFILE>");

    -- Print the Full name for Name_Of.  Defaults to the current selection's
    -- text.


    procedure Enclosing_World (Levels   : Positive := 1;
                               Response : String   := "<PROFILE>");

    -- Enclosing_World is equivalent to Context ("^$$");


    procedure Context (To_Be    : Context_Name := "$";
                       Response : String       := "<PROFILE>");

    -- Set the job context to To_Be.  When To_Be is already the job context,
    -- only printing takes place.


    procedure Copy (From       : Name    := "<REGION>";
                    To         : Name    := "<IMAGE>";
                    Recursive  : Boolean := True;
                    Response   : String  := "<PROFILE>";
                    Copy_Links : Boolean := True;
                    Options    : String  := "");

    -- Copy version From resulting in version To; see table below.

    -- To designates an object that will exist after the copy has
    -- completed.  For Ada objects, changing the simple name may require
    -- user intervention before installation.

    -- To is interpreted in the current context or specified full
    -- context and must be unique.

    -- The object designated by To will be the same class as From.

    -- Objects representing devices cannot be copied.

    -- Any situation that would require demoting unrelated declarations
    -- results in an error, suppressing the copy.

    -- Recursive applies to objects that contain other objects and indicates
    -- that these contained objects should be copied.

    -- If Copy_Links is true, then link packs for any worlds copied are
```

```
-- duplicated, and any link which pointed to the source for a copy is
-- altered to point to the destination.  If Copy_Links is false, any
-- copied worlds will have empty link packs.

-- If a world and its switch file are copied, then the copied unit will
-- point to the copy of the switch file.  If the switch file is not
-- copied, then the unit and its original will reference the same switch
-- file.

-- Ada units are copied as source.

-- Copy and Move subsume the functionality of Copy_Into and Move_Into
-- from previous releases.  Whether a Copy/Move is "to" or "into" is
-- determined by the type of object specified by the From and To
-- parameters.  The chart below gives the details.

-- If wildcards/substitution characters are involved in the From and To
-- parameters, this matrix is applied AFTER these wildcards have been
-- expanded.  If the source is over-specified (e.g., "?" is used with
-- the recursive switch) a source object is copied only once.

--                          COPY/MOVE to/into matrix
--
--             \ TO
--              +---------+---------+---------+--------+--------+----------
--     FROM     | Non-Ada | Library | Subunit | World  | Drctry | No Object
--              | Object  | Unit    |         |        |        |
--     ---------+---------+---------+---------+--------+--------+----------
--     Non-Ada  |         |         |         |        |        |
--     Object   | TO (1)  | Error   | Error   | INTO   | INTO   |    TO
--              |         |         |         |        |        |
--     ---------+---------+---------+---------+--------+--------+----------
--     Library  |         |         |         |        |        |
--      Unit    | Error   |   TO    |   TO    | INTO   | INTO   |    TO
--       (2)    |         |         |         |        |        |
--     ---------+---------+---------+---------+--------+--------+----------
--     Subunit  |         |         |         |        |        |
--       (2)    | Error   | INTO    |   TO    | INTO   | INTO   |    TO
--              |         |         |         |        |        |
--     ---------+---------+---------+---------+--------+--------+----------
--     World    |         |         |         |        |        |
--       (3)    | Error   | Error   | Error   | TO (4) | TO (4) |    TO
--              |         |         |         |        |        |
--     ---------+---------+---------+---------+--------+--------+----------
--     Drctry   |         |         |         |        |        |
--       (3)    | Error   | Error   | Error   | TO (4) | TO (4) |    TO
--              |         |         |         |        |        |
--     ---------+---------+---------+---------+--------+--------+----------
--
```





```
--    Notes:
--
--    1.  User can make any "TO" an "INTO" by appending ".name" to To;
--        Appending ".#" would yield target with same simple name as From.
--
--    2.  Any class mismatch is an error.
--
--    3.  Subunits of unit are involved if Recursive switch is set;
--        nesting of subunits is preserved.
--
--    4.  Subcomponents of library are involved if Recursive switch is set;
--        relative nesting of subcomponents is preserved.
--
--    5.  Contents of source library are merged with contents of
--        target library.
--


procedure Move (From       : Name    := "<REGION>";
                To         : Name    := "<IMAGE>";
                Recursive  : Boolean := True;
                Response   : String  := "<PROFILE>";
                Copy_Links : Boolean := True;
                Options    : String  := "");

-- Equivalent to Copy (Existing, ...); Delete (Existing);

subtype Volume is Natural range 0 .. 31;

Nil : constant Volume := Volume'First;

type Kind is (World, Directory, Subpackage);

procedure Create (Name     : Library.Name := ">>LIBRARY NAME<<";
                  Kind     : Library.Kind := Library.Directory;
                  Vol      : Volume       := Library.Nil;
                  Model    : String       := "!Model.R1000";
                  Response : String       := "<PROFILE>");
--
-- Create a library of the specified type.  The Nil volume represents
-- the 'best' volume (The 'best' volume does not necessarily mean the
-- volume with the most space.  The 'best' volume calculation takes into
-- account the percentage of a volume that is available and an estimate
-- of the real consumption of previously allocated worlds).  Vol is
-- ignored for Subpackages, which are not control points, and must be on
-- the same volume as their parent.  When creating a World, links are
-- copied from Model (unless it is "").
```

```
procedure Rename (From     : Name        := "<SELECTION>";
                  To       : Simple_Name := ">>NEW SIMPLE NAME<<";
                  Response : String      := "<PROFILE>");

-- Change the name of an existing library unit or managed object.
-- References to library units are not changed -- only the actual
-- name of the unit.  Various other restrictions apply.


procedure Delete (Existing : Name                        := "<SELECTION>";
                  Limit    : Compilation.Change_Limit := "<DIRECTORIES>";
                  Response : String                      := "<PROFILE>")
    renames Compilation.Delete;

-- Delete versions of objects designated by Existing.  Either an object
-- must be selected, or the name of an object supplied.
-- Results will be reversible with Undelete, unless retention count = 0.


procedure Destroy (Existing  : Name                     := "<SELECTION>";
                   Threshold : Natural                  := 1;
                   Limit     : Compilation.Change_Limit := "<DIRECTORIES>";
                   Response  : String                   := "<PROFILE>")
    renames Compilation.Destroy;

-- Destroy versions and associated declarations designated by Existing.
-- Destroyed versions are expunged and cannot be undeleted.


procedure Undelete (Existing : Name   := "<CURSOR>";
                    Response : String := "<PROFILE>");

-- Undelete an Existing version.

-- Only a fixed number of deleted versions will be retained.  Excess
-- versions will be automatically expunged, at which time they can no
-- longer be undeleted.


Default_Keep_Versions : constant := -1;

-- Keep the default number of deleted versions.


procedure Expunge (Existing      : Name    := "<IMAGE>";
                   Keep_Versions : Integer := 0;
                   Recursive     : Boolean := True;
                   Response      : String  := "<PROFILE>");
```





```
-- Make deletions permanent.  Recursive causes subobjects to be
-- expunged.  Keep_Versions deleted versions will be retained.
-- Recursive causes subobjects to be touched.  Use Recursive => false
-- and "?" wildcard to avoid expunging nested worlds.


procedure Set_Retention_Count
             (Existing      : Name    := "<IMAGE>";
              Keep_Versions : Integer := Library.Default_Keep_Versions;
              Recursive     : Boolean := True;
              Response      : String  := "<PROFILE>");

-- Set the default number of deleted versions of an object which are
-- retained.  Default is the same as the object's parent.  Recursive
-- causes subobjects to be touched.  Use Recursive => false and "?"
-- wildcard to avoid setting retention count for nested worlds.


procedure Freeze (Existing  : Name    := "<IMAGE>";
                  Recursive : Boolean := True;
                  Response  : String  := "<PROFILE>");

-- Prevent further changes to an object.  Recursive causes subobjects to
-- be frozen.  Use Recursive => false and "?" wildcard to avoid freezing
-- nested worlds.


procedure Unfreeze (Existing  : Name    := "<IMAGE>";
                    Recursive : Boolean := True;
                    Response  : String  := "<PROFILE>");

-- Permit changes to an object.  Recursive causes subobjects to be
-- unfrozen.  Use Recursive => false and "?" wildcard to avoid
-- unfreezing nested worlds.


procedure Default (Existing : Name   := "<SELECTION>";
                   Response : String := "<PROFILE>");

-- Set the default Version for the existing object and print the result
-- as a message.


procedure Set_Subclass (Existing    : Name   := "<SELECTION>";
                        To_Subclass : String := "";
                        Response    : String := "<PROFILE>");

-- Set the subclass of an object.  A null string for To_Subclass
-- requests the system to set the subclass to its 'best guess'.
```

```
type Field is (Object,       -- Ada name.
               Version,      -- Version name.
               Class,        -- Directory class name.
               Subclass,     -- Subclass of the object.
               Updater,      -- User to last update object.
               Update_Time,  -- Time of last update.
               Creator,      -- User who created object.
               Create_Time,  -- Time of creation.
               Reader,       -- User to last read object.
               Read_Time,    -- Time of last read.
               Size,         -- Current size of object.
               Status,       -- Source, Installed, Coded, Elaborated, etc.
               Frozen,       -- Is this object frozen.
               Retain,       -- Max. number of deleted versions retained.
               Declaration   -- Ada declaration of object.
               );

type Fields is array (Field) of Boolean;
Verbose_Format : constant Fields := Fields'(Object .. Update_Time => True,
                                            Size .. Retain        => True,
                                            others                => False);
Ada_Format : constant Fields := Fields'(Status      => True,
                                        Declaration => True,
                                        others      => False);
All_Fields : constant Fields := Fields'(others => True);
Terse_Format : constant Fields := Fields'(Object => True, others => False);

procedure List (Pattern    : Name    := "<IMAGE>@";
                Displaying : Fields  := Library.Terse_Format;
                Sorted_By  : Field   := Library.Object;
                Descending : Boolean := False;
                Response   : String  := "<PROFILE>";
                Options    : String  := "");

procedure Verbose_List (Pattern    : Name    := "<IMAGE>{@'V(ALL)}";
                        Displaying : Fields  := Library.Verbose_Format;
                        Sorted_By  : Field   := Library.Object;
                        Descending : Boolean := False;
                        Response   : String  := "<PROFILE>";
                        Options    : String  := "") renames List;

procedure File_List (Pattern    : Name    := "<IMAGE>@'C(FILE)";
                     Displaying : Fields  := Library.Verbose_Format;
                     Sorted_By  : Field   := Library.Object;
                     Descending : Boolean := False;
                     Response   : String  := "<PROFILE>";
                     Options    : String  := "") renames List;
```





```
procedure Ada_List (Pattern    : Name    := "<IMAGE>@'C(ADA)";
                    Displaying : Fields  := Library.Ada_Format;
                    Sorted_By  : Field   := Library.Declaration;
                    Descending : Boolean := False;
                    Response   : String  := "<PROFILE>";
                    Options    : String  := "") renames List;

procedure Space (For_Object   : Name     := "<IMAGE>";
                 Levels       : Positive := 2;
                 Recursive    : Boolean  := True;
                 Each_Object  : Boolean  := False;
                 Each_Version : Boolean  := False;
                 Space_Types  : Boolean  := False;
                 Response     : String   := "<PROFILE>";
                 Options      : String   := "");
-- Show the space utilization (in pages) for For_Object.  Also
-- display space usage for contained libraries to depth specified
-- by Levels.  The space includes subobjects and contained libraries,
-- unless Recursive is false, in which case only the space for the
-- specified object is displayed.  Thus, if Recursive is true, the
-- space is cumulatively totalled.
--
-- Each_Object causes the individual space the each object to be included
-- in the display in addition to libraries.
--
-- If Space_Types is true, a different display showing space broken down
-- by category (including the object itself, code segment, attribute
-- spaces, and list files) is displayed.  In this case, the Each_Version
-- parameter will show information for each version of each object.
-- Each_Version is used only if Space_Types true.  Levels is used only
-- if Space_Types is false.

procedure Compact_Library (Existing : Name   := "<SELECTION>";
                           Response : String := "<PROFILE>");
-- This procedure may be used to reduce the amount of storage consumed
-- by frequently modified directories which are used to store files.

-- Quiet forms similar to those in Library_Object_Editor, but
-- these commands work based on the current context rather than
-- the current image.

procedure Create_World (Name     : Library.Name := ">>WORLD NAME<<";
                        Kind     : Library.Kind := Library.World;
                        Vol      : Volume := Library.Nil;
                        Model    : String := "!Model.R1000";
                        Response : String := "<PROFILE>") renames Create;
```

```
    procedure Create_Directory (Name     : Library.Name := ">>DIRECTORY NAME<<";
                                Kind     : Library.Kind := Library.Directory;
                                Vol      : Volume       := Library.Nil;
                                Model    : String       := "";
                                Response : String       := "<PROFILE>")
        renames Create;


    procedure Create_Unit (Name     : Library.Name := ">>ADA NAME<<";
                           Kind     : Library.Kind := Library.Subpackage;
                           Vol      : Volume := Library.Nil;
                           Model    : String := "";
                           Response : String := "<PROFILE>") renames Create;


    procedure Display (Name : Library.Name := "[]");
    -- Display the named object in a library window.


    procedure Reformat_Image (Existing : Name   := "<SELECTION>";
                              Response : String := "<PROFILE>");
    -- Cause the image for a unit to be reconstructed.

end Library;
```





```
with Links_Implementation;
package Links is

    subtype World_Name is String;

    -- The string name for any directory object may be given for a world
    -- parameter, to indicate the world that contains the object.

    subtype Link_Name is String;

    -- An Ada simple name. When used as an in-parameter, except in Add and
    -- Replace, it may contain wildcard characters.  In Add and Replace it
    -- may contain substitution characters.

    subtype Source_Name is String;

    -- A directory string name that specifies an existing Ada Library Unit.
    -- (The unit does not have to be installed, but its declaration must be
    -- in a library.) May contain wildcard characters when used as an
    -- in-parameter.

    subtype Source_Pattern is String;

    -- A string (containing wildcards) which will be matched against the
    -- full names of the objects denoted by links.

    subtype Link_Kind is Links_Implementation.Link_Kind;
    Internal : constant Link_Kind := Links_Implementation.Internal;
    External : constant Link_Kind := Links_Implementation.External;
    Any      : constant Link_Kind := Links_Implementation.Any;

    -- A link is Internal if its source object is in the world of the link
    -- pack; otherwise it is External.


    procedure Add (Source   : Source_Name := ">>SOURCE NAMES<<";
                   Link     : Link_Name   := "#";
                   World    : World_Name  := "<IMAGE>";
                   Response : String      := "<PROFILE>");

    -- For each Ada library unit defined by Source, a link is created in the
    -- link pack for World.  The Source object is associated with the simple
    -- Ada name given by Link.  The operation fails if the specified Link name
    -- already exists in the pack, unless the new link is compatible with the
    -- old link.  The new link is defined to be compatible with the old link
    -- iff both links refer to the same object or the object referred to be the
    -- old link has been deleted.
```

```
procedure Replace (Source   : Source_Name := ">>SOURCE NAMES<<";
                   Link     : Link_Name   := "#";
                   World    : World_Name  := "<IMAGE>";
                   Response : String      := "<PROFILE>");

-- For each Ada Library unit defined by Source, a link is created in
-- the link pack for World.  The Source object is associated with the
-- simple Ada name given by Link. If a link of the same name
-- already exists, it is replaced by the new definition.


procedure Delete (Link     : Link_Name      := ">>LINK NAMES<<";
                  Source   : Source_Pattern := "?";
                  Kind     : Link_Kind      := Links.Any;
                  World    : World_Name     := "<IMAGE>";
                  Response : String         := "<PROFILE>");

-- The Links that match both the Source and Link wildcards and the
-- specified kind are deleted from the link pack of the given World.


procedure Copy (Source_World : World_Name     := ">>WORLD NAME<<";
                Target_World : World_Name     := "<IMAGE>";
                Link         : Link_Name      := "@";
                Source       : Source_Pattern := "?";
                Kind         : Link_Kind      := Links.Any;
                Response     : String         := "<PROFILE>");

-- The Links of Source_World that match the specified Source and Link
-- names and the given Link_Kind are copied to Target_World.


procedure Display (World    : World_Name     := "<IMAGE>";
                   Link     : Link_Name      := "@";
                   Source   : Source_Pattern := "?";
                   Kind     : Link_Kind      := Links.Any;
                   Response : String         := "<PROFILE>");

-- Lists the links that match the given wild cards in the given world


procedure Dependents (Link     : Link_Name      := "@";
                      Source   : Source_Pattern := "?";
                      Kind     : Link_Kind      := Links.Any;
                      World    : World_Name     := "$$";
                      Response : String         := "<PROFILE>");
```





```
    -- Computes the Library Units of the world that are installed or coded
    -- and reference any of the Link commands specified by the Source and
    -- Link parameters.


    procedure Edit  (World : World_Name := "<IMAGE>");
    procedure Visit (World : World_Name := "<IMAGE>");

    -- Enters the links object editor.  If there is no links window for the
    -- world to be edited, edit will create a new window, and visit will
    -- reuse an existing window of there is one.


    procedure Insert (Source : Source_Name := ">>SOURCE NAME<<");

    procedure Update (Source : Source_Name := ">>SOURCE NAME<<");

    -- Insert and Update peform the same function as Add and Replace, but
    -- they must be run in a command window off a links image.


    procedure Expunge (World    : World_Name := "<IMAGE>";
                       Response : String     := "<PROFILE>");

end Links;
```

```
with Io;
with Diana;
with Directory;
with Error_Messages;
with Machine;
with Profile;
with Simple_Status;
package Log is

    subtype Name is String;  -- an unambiguous string name

    procedure Set_Log (To_Be  : Name               := ">>FILE NAME<<";
                       Filter : Profile.Log_Filter := Profile.Filter);
    -- Set Current_Output to To_Be, changing the profile to direct log
    -- output to Use_Current_Output.  Change the Log_Filter to Filter.
    -- If To_Be cannot be created, Current_Output is not redirected, but
    -- no exception is raised.

    procedure Reset_Log (Filter : Profile.Log_Filter := Profile.Filter);
    -- Equivalent to IO.Reset_..., but changes Log_Filter

    procedure Put_System_Messages
                (Response : Profile.Response_Profile := Profile.Get);
    -- Copy contents of the message log for the current job into Current_Output

    procedure Put_Job_Messages
                (For_Job  : Machine.Job_Id;
                 Response : Profile.Response_Profile := Profile.Get);
    -- Copy contents of the message log for specified job into Current_Output

    procedure Put_Condition
                (Status   : Simple_Status.Condition;
                 Response : Profile.Response_Profile := Profile.Get);
    -- Display contents of Status in Current_Output.

    procedure Put_Line (Message  : String;
                        Kind     : Profile.Msg_Kind         := Profile.Note_Msg;
                        Response : Profile.Response_Profile := Profile.Get);
    -- Appends the Message to the end of the Current_Output as described by
    -- the given response profile. If Profile.Includes (Kind, Response) is
    -- true, then the messages is generated as described below; otherwise
    -- the Put_line call returns immediately.

    -- The Time, Date and Symbol prefixes are printed first, in the order
    -- and format specified by the Profile.Prefixes (Response) array.
    -- If the Profile.Symbols prefix is requested, a unique three-character
    -- string is generated for each possible vale of Kind:
```





```
    --   -- KIND --   Symbol           Excplanation

    --  Position_Msg    >>>
    --       Identifies the location in a file or program
    --       to which subsequent messages refer.
    --  Sharp_Msg       ### \
    --  Dollar_Msg      $$$  + Available for user-defined purposes
    --  At_Msg          @@@ /
    --  Debug_Msg       ???
    --  Auxiliary_Msg   :::
    --  Note_Msg        ---
    --       Supplemental information.
    --  Positive_Msg    +++
    --       Indicates that a major step in the process has
    --       completed successfully. e.g. a unit has been
    --       compiled, or generation of an output file is
    --       complete.
    --  Warning_Msg     !!!
    --       Indicates a minor problem in processing a major
    --       step of the process.  Warnings generally do not
    --       lead to negative messages (see below).
    --  Negative_Msg    ++*
    --       Indicates that a major step in the process has
    --       completed unsuccessfully. e.g. a unit has failed
    --       to compile, or generation of an output file is
    --       could not be accomplished.
    --  Error_Msg       ***
    --       Indicates a significant problem within a major
    --       step of the process that has been detected by
    --       the command.  Error messages will
    --       frequently be followed by negative messges
    --  Exception_Msg   %%%
    --       Indicates that a command caught an unexpected
    --       exception.

    -- The text of the message follows the prefixes.  If the message line
    -- exceeds Profile.Width (Response), it is continued on the next line.
    -- Each continuation line starts with the same prefixes as the first
    -- line, except that the three-character string "..." is used instead
    -- of the symbols in the table above.  (If no Symbols prefix is
    -- requested by the Profile.Prefixes (Response), the symbol string
    -- "... " is inserted between the rightmost prefix and the message text.)

    procedure Copy (Log_File    : Name               := "<IMAGE>";
                    Destination : Name               := "";
                    Filter      : Profile.Log_Filter := Profile.Filter);
    -- Once a log file has been generated with symbol prefixes, the
    -- following procedures may be used to copy the file while filtering
    -- out unwanted messages.  The default destination is Current_Output
```

```
procedure Filter (Log_File    : Name    := "<IMAGE>";
                  Destination : Name    := "";
                  Auxiliaries : Boolean := True;
                  Diagnostics : Boolean := True;
                  Notes       : Boolean := True;
                  Positives   : Boolean := True;
                  Negatives   : Boolean := True;
                  Positions   : Boolean := True;
                  Warnings    : Boolean := True;
                  Errors      : Boolean := True;
                  Exceptions  : Boolean := True;
                  Sharps      : Boolean := True;
                  Dollars     : Boolean := True;
                  Ats         : Boolean := True);

procedure Summarize (Log_File    : Name    := "<IMAGE>";
                     Destination : Name    := "";
                     Auxiliaries : Boolean := True;
                     Diagnostics : Boolean := True;
                     Notes       : Boolean := False;
                     Positives   : Boolean := True;
                     Negatives   : Boolean := True;
                     Positions   : Boolean := False;
                     Warnings    : Boolean := False;
                     Errors      : Boolean := False;
                     Exceptions  : Boolean := False;
                     Sharps      : Boolean := False;
                     Dollars     : Boolean := False;
                     Ats         : Boolean := False) renames Filter;

procedure Filter_Errors (Log_File    : Name    := "<IMAGE>";
                         Destination : Name    := "";
                         Auxiliaries : Boolean := True;
                         Diagnostics : Boolean := True;
                         Notes       : Boolean := False;
                         Positives   : Boolean := False;
                         Negatives   : Boolean := True;
                         Positions   : Boolean := False;
                         Warnings    : Boolean := True;
                         Errors      : Boolean := True;
                         Exceptions  : Boolean := False;
                         Sharps      : Boolean := False;
                         Dollars     : Boolean := False;
                         Ats         : Boolean := False) renames Filter;

procedure Set_Error  (To_Be : Name := ">>FILE NAME<<");
procedure Set_Input  (To_Be : Name := "<REGION>") renames Io.Set_Input;
procedure Set_Output (To_Be : Name := ">>FILE NAME<<");
-- Set_Output and Set_Error deal with interaction with profiles that
```





```
-- direct Log output to streams other than Current_Output.

procedure Pop_Error  renames Io.Pop_Error;
procedure Pop_Input  renames Io.Pop_Input;
procedure Pop_Output renames Io.Pop_Output;

procedure Reset_Error  renames Io.Reset_Error;
procedure Reset_Input  renames Io.Reset_Input;
procedure Reset_Output renames Io.Reset_Output;

procedure Flush (Response : Profile.Response_Profile := Profile.Get);
-- force any log output into the log file

procedure Save (Response : Profile.Response_Profile := Profile.Get);
-- make the current contents of the log file permanent; calls flush

generic
    type Object_Type is private;
    with function Full   (Object : Object_Type) return String;
    with function Simple (Object : Object_Type) return String;
    with function Is_Nil (Object : Object_Type) return Boolean;
    with function Nil                           return Object_Type;

procedure Put_Line_Generic
             (Object1  : Object_Type;
              Message  : String                   := "";
              Object2  : Object_Type              := Nil;
              Kind     : Profile.Msg_Kind         := Profile.Note_Msg;
              Response : Profile.Response_Profile := Profile.Get);

procedure Put_Line (Object1  : Directory.Object;
                    Message  : String                   := "";
                    Object2  : Directory.Object         := Directory.Nil;
                    Kind     : Profile.Msg_Kind         := Profile.Note_Msg;
                    Response : Profile.Response_Profile := Profile.Get);

procedure Put_Line (Object1  : Directory.Version;
                    Message  : String                   := "";
                    Object2  : Directory.Version        := Directory.Nil;
                    Kind     : Profile.Msg_Kind         := Profile.Note_Msg;
                    Response : Profile.Response_Profile := Profile.Get);

procedure Put_Line (Object1  : Diana.Tree;
                    Message  : String                   := "";
                    Object2  : Diana.Tree               := Diana.Empty;
                    Kind     : Profile.Msg_Kind         := Profile.Note_Msg;
                    Response : Profile.Response_Profile := Profile.Get);

-- Enters a message into the log, if messages of the Kind specified
```

```
    -- are to be included.

    -- If the message does go into the log, the name of the specified
    -- object(s) is computed and inserted into the text of the message.
    -- The location for the name of the first object is indicated by the
    -- symbol "<1>"; if this string is not found in the message, the
    -- name of the object is placed at the beginning of the message.
    -- The location for the name of the object object is indicated by the
    -- symbol "<2>"; if this string is not found in the message, the
    -- name of the object, if not nil, is placed at the end of the message.

    -- Directory.Naming.Unique_Full_Name is used to generate the name of
    -- the object when the symbols given above are used or if no symbols
    -- are found.  The symbols "<<1>>" and "<<2>>" cause the value of
    -- Directory.Naming.Get_Simple_Name to be used instead.

    procedure Put_Errors (Errors   : Error_Messages.Errors;
                          Response : Profile.Response_Profile := Profile.Get);
    -- Enter the Error messages into the log.

    function Image (Kind : Profile.Msg_Kind) return String;
    -- Returns the three-letter prefix used for the indicated Msg_Kind.

end Log;
```





```
with System_Utilities;

package Mail is

    procedure Create (Mailbox  : String := ">>SIMPLE NAME<<";
                      For_User : String := "");

    -- Create a mailbox.  For_User defaults to current user.
    -- The user is added to the local machine name map.
    --


    procedure Edit (Mailbox : String := "MAIN"; For_User : String := "");

    -- Enters the mail object editor.  If no window exists for the given
    -- mailbox, a new window will be created.
    -- For_User defaults to current user.


    procedure Expunge;

    --  This command is only valid in a command window off of a mailbox
    --  window.  The given mailbox is expunged.


    procedure Answer (To_All : Boolean := False);

    --  This command may be used in a mailbox window or a read mail window.
    --  It causes an answer window to be created with certain fields
    --  initialized with appropriate default values.
    --  In a mailbox window, the selected message is answered.  In a read
    --  mail window, the current message is answered.


    procedure Reply (To_All : Boolean := False) renames Answer;


    procedure Forward;

    --  Same as answer, except a forward window is created.


    procedure Remail;

    --  Same as answer, except a remail window is created.
```

```
procedure Send;

--  Brings up a send window.


procedure Reload_Name_Map;


--  Causes the mail object editor to read in the name map.


function New_Messages return Natural;

-- returns number of unread messages in the mailbox;  if the mailbox is
-- locked by another session or user the function returns 99999999.


procedure Send_Message (To       : String := ">>USER_NAMES<<";
                        Subject  : String := "";
                        Text     : String := "";
                        Cc       : String := "";
                        From     : String := System_Utilities.User_Name;
                        Response : String := "<PROFILE>");

-- Composes and sends a message with content as specified by the parameters

procedure Notify;

-- Display the number of unread messages in the message window banner.

----------------------------------------------------------------------

--  The mail object editor manages three types of windows:
--      1)  Mailbox windows
--      2)  Read message windows and
--      3)  Transmit windows (answer, forward, remail, and send).
--  The following is a list of supported common commands per window type.



--  Mailbox window common commands:
--
--  Release (obj-x and obj-g).  Deletes the window and closes the mailbox.
--  Any windows on objects within this mailbox will also be deleted.

--  Copy.  Copy the selected message(s) from one mailbox to another.

--  Delete.  Delete the selected message(s).  If the selected message(s)
--  is already flagged for deletion, the message(s) are expunged.
```





```
--  Definition.  If the cursor is on a message header, brings up a window
--  on the underlying message.  If the cursor is on a mailbox name, brings
--  up a window on the underlying mailbox.

--  Format.  Reads in new messages, and positions the cursor on the first
--  new message.
--
--  Insert.  Bring up a send window.
--
--  Move.  Move selected message from one mailbox to another.

--  Redo.  If the selected message is an undeliverable notification a
--  send window is brought up with the original message ready to be
--  resent.  Otherwise a reply window is brought up.

--  Revert.  Same as format.

--  Sort_Image.  Change the sort order in which the messages are displayed.

--  Undo.  Undeleted selected message(s).



--  Read message window common commands:

--  Release.  Delete the window, and position the cursor on the next
--  undeleted message in the mailbox window.

--  Delete.  Same as Release, but underlying message is deleted.

--  Enclosing.  Go to enclosing mailbox window.

--  Redo.  Same as redo in a mailbox window.

--  Transmit window common commands:

--  Release.  Obj-G deletes the window.  Obj-X send the message, then
--  deletes the window.

--  Edit.  Make the window editable.

--  Enclosing.  If in a send window, goes to the users main mailbox.
--  Otherwise goes to the message being answered, forwarded or remailed.

--  Promote.  Sends the message, and makes the window read only.

--  Semanticize.  Check names in the to and cc lists, and underlines
--  names which can't be found.
```

```
    -- The sort fields specified by the format parameter are as follows:
    -- 1  From
    -- 2  Date Received
    -- 3  Date Sent
    -- 4  Read/Unread
    -- 5  Deleted/Non-deleted
    -- Use the negatives of these numbers for reverse sorting.


    -- Mail Switches:

    --  Mail_Multiple_Message_Windows
    --    Boolean indicating whether read window should be reused
    --
    --  Mail_Default_Sort_Order
    --    Integer  (see comment on sort orders)

end Mail;
```





```
package Menu_Operations is

    -----------------------------------------------------------------------
    -- This package is not yet fully implemented.  Most routines will    --
    -- cause the exception Nonexistent_Page_Error to be raised.  It is   --
    -- included as part of the environment in anticipation of the        --
    -- requirements of future Rational products.                         --
    -----------------------------------------------------------------------

    subtype Menu_Item is Integer;
    procedure Dispatch  (Item : Menu_Item; Parameter : String := "");
    procedure Fastpatch (Item : Menu_Item; Parameter : String := "");

    -- Dialog box completion operations.  Do not call directly!

    subtype Box_Id is Natural;
    procedure Click_Ok (Id : Box_Id := 0; Parms : String := "");
    --
    -- unless id /= 0 and parms non null, read the input from the
    -- keyboard in raw mode.

    type Subsystem_Kind     is (Spec_Load, Combined);
    type View_Kind          is (Spec_Load, Combined);
    type Switch_Kind        is (Library_Switches, Session_Switches);
    type Activity_Copy_Kind is (Exact_Copy, Value_Copy, Differential);
    type Goal_State         is (Archived, Source, Installed, Coded);
    type Compilation_Limit  is (This_View_Or_World, All_Views_Or_Worlds);
    type Backup_Kind        is (Full, Incremental);

    ---------------
    -- File Menu --
    ---------------

    package File is
        procedure New_File;      -- Create a new file
        procedure New_File       (Name : String);
        procedure New_Ada;
        procedure New_World;
        procedure New_World      (Name            : String;
                                  Acl             : String;
                                  Default_Acl     : String;
                                  Retention_Count : Natural;
                                  Model           : String);
        procedure New_Directory;
        procedure New_Directory (Name : String);

        procedure New_Subsystem;
```

```
procedure New_Subsystem     (Name                 : String;
                             Kind                 : Subsystem_Kind;
                             Initial_View_Kind    : View_Kind;
                             Initial_View_Name    : String;
                             Model                : String;
                             Comments_For_History : String);
procedure New_Working_View;
procedure New_Working_View  (Name                 : String;
                             Copy_Of_View         : String;
                             Kind                 : View_Kind;
                             Join                 : Boolean;
                             Imports              : String;
                             Model                : String;
                             Comments_For_History : String);
procedure New_Spec_View;
procedure New_Spec_View     (Name                 : String;
                             Copy_Of_View         : String;
                             Join                 : Boolean;
                             Imports              : String;
                             Model                : String;
                             Comments_For_History : String);
procedure New_Release_View;
procedure New_Release_View  (Name                 : String;
                             Copy_Of_View         : String;
                             Join                 : Boolean;
                             Imports              : String;
                             Kind_Of_View         : View_Kind;
                             Model                : String;
                             Comments_For_History : String;
                             Configuration_Only   : Boolean);
procedure New_Code_View;
procedure New_Code_View     (Name                 : String;
                             Copy_Of_View         : String;
                             Comments_For_History : String);
procedure New_Switch_File;
procedure New_Switch_File   (Name : String; Kind : Switch_Kind);
procedure New_Activity;
procedure New_Activity      (Name      : String;
                             Copy_Of   : String;
                             Copy_Mode : Activity_Copy_Kind);

procedure New_Mailbox;
procedure New_Mailbox     (Name : String);
procedure New_Venture;
procedure New_Venture
          (Name : String; Notes : String; Make_Default : Boolean);
procedure New_Work_Order;
procedure New_Work_Order (Name            : String;
                          Notes           : String;
```





```
                          Venture         : String;
                          Work_Order_List : String);
procedure New_Work_Order_List;
procedure New_Work_Order_List
          (Name : String; Venture : String; Make_Default : Boolean);
procedure New_System;
procedure New_System     (Name : String; Initial_View_Name : String);
procedure New_Gateway;
procedure New_Gateway    (Name : String; Gateway_Class : String);

procedure Open;          -- Open a window on an object
procedure Open                 (Name : String; For_Edit : Boolean);
procedure Open_With_Demote     (Name : String);
procedure Open_With_Check_Out  (Name                 : String;
                                Comments_For_History : String);
-- should include comment about implicit accept changes
procedure Close;         -- Close a window.
procedure Close_Commit (Do_Commit : Boolean);

procedure Save;          -- Commit changes
procedure Revert;        -- Revert to previous value.  Ask about
                         -- losing changes if necessary.
procedure Revert_Lose_Changes;
procedure Print;         -- Print an object or set of objects
procedure Print (Object_Or_Image                        : String;
                 From_First_Page                        : Natural;
                 To_Last_Page                           : Natural;
                 Display_As_Twoup                       : Boolean;
                 Display_Border                         : Boolean;
                 Display_Filename                       : Boolean;
                 Display_Date                           : Boolean;
                 Ignore_Display_Parameters_For_Postscript : Boolean;
                 Highlight_Reserved_Words_For_Ada       : Boolean;
                 Other_Options                          : String;
                 Number_Of_Copies                       : Natural;
                 Printer                                : String;
                 Effort_Only                            : Boolean);

procedure Properties;   -- View and edit properties of an object.
procedure File_Properties      (Name        : String;
                                Acl         : String;
                                Frozen      : Boolean;
                                Controlled  : Boolean;
                                Checked_Out : Boolean);  -- ***
procedure World_Properties     (Name            : String;
                                Acl             : String;
                                Default_Acl     : String;
                                Switch_File     : String;
                                Retention_Count : Natural); -- ***
```

```
    procedure Subsystem_Properties (Name : String); -- ***
    procedure View_Properties      (Name : String);    -- ***
    procedure Copy;          -- Copy an object or objects
    procedure Copy                 (Source : String; Destination : String);
    procedure Move;          -- Move an object or objects
    procedure Move                 (Source : String; Destination : String);
    procedure Delete;        -- Delete an object or objects
    procedure Delete               (Name : String);
    procedure Delete_Confirm       (Name : String);
    procedure Run;           -- Run a command
    procedure Run                  (Command_Name : String;
                                    Parameters   : String;
                                    On_Machine   : String);
    procedure Quit;          -- Logoff
    procedure Quit_Confirm;
end File;


---------------
-- Edit menu --
---------------

package Edit is
    procedure Format;                 -- Reformat object as appropriate
    procedure Cut;              -- delete selected text
    procedure Copy;             -- Copy selected text to buffer
    procedure Paste;            -- Paste in text from cut buffer
    procedure Copy_To_Clipboard; -- Copy selected text to Motif clipboard
    procedure Search_Forward;   -- search text
    procedure Search_Backward;  -- search text toward beginning
    procedure Replace;          -- search and replace
    procedure Check_Spelling_Word;
    procedure Check_Spelling_Document; -- Spelling checker
    procedure Insert_File;      -- Insert contents of file
    procedure Overwrite_Mode;   -- enter overwrite mode
    procedure Insert_Mode;      -- enter insert mode
    procedure Fill_Mode;        -- enter fill mode
    procedure No_Fill_Mode;     -- exit fill mode
    procedure Rename_Ada;       -- withdraw Ada unit to rename.
    procedure Underlines_Off;
end Edit;

-----------------
-- Region menu --
-----------------

package Region is
    procedure Unselect;
    procedure Beginning_Of;
```





```
    procedure End_Of;
    procedure Capitalize;
    procedure Uppercase;
    procedure Lowercase;
    procedure Make_Comment;
    procedure Uncomment;
    procedure Fill;
    procedure Justify;
end Region;

-------------------
-- Traverse Menu --
-------------------

package Traverse is
    procedure Next_Item;
    procedure Previous_Item;
    procedure Definition;
    procedure Enclosing;
    procedure Other_Part;
    procedure Home_Library;
    procedure Resolve_Name;
    procedure Resolve_Name (Name : String);
end Traverse;

------------------
-- Compile Menu --
------------------

package Compile is
    procedure Semanticize;
    procedure Code;
    procedure Install;
    procedure Install_Confirm;
    procedure Source;
    procedure Source_Confirm;
    procedure Archive;
    procedure Archive_Confirm;
    procedure Options;
    procedure Promote;
    procedure Promote (Name  : String;
                       Goal  : Goal_State;
                       Limit : Compilation_Limit);
    procedure Demote;
    procedure Demote  (Name  : String;
                       Goal  : Goal_State;
                       Limit : Compilation_Limit);
    procedure Show_Usage;
    procedure Show_Unused;            -- not used anywhere
```

```
      procedure Show_Unused_Local;          -- unused in this unit
      procedure Load;
      procedure Load     (Name : String; Result : String);
      procedure Parse;
      procedure Parse    (Name : String; Destination_Directory : String);
      procedure Build_Private_Part;
      procedure Build_Body;
      procedure Make_Separate;
      procedure Make_Inline;
   end Compile;

   -------------
   -- CM menu --
   -------------

   package Cm is
      procedure Check_Out;
      procedure Check_Out (Name : String; Comments : String);
      procedure Check_Out_Confirm (Name : String; Comments : String);
      procedure Check_In;
      procedure Check_In (Name : String; Comments : String);
      procedure Abandon;
      procedure Abandon (Name : String);
      procedure Abandon_Confirm (Name : String);
      procedure Accept_Changes;
      procedure Accept_Changes
             (Name : String; Entire_View : Boolean; Source : String);
      procedure Accept_Changes_Confirm
             (Name : String; Entire_View : Boolean; Source : String);
      procedure Join;
      procedure Join (Name                 : String;
                      To_View              : String;
                      Comments_For_History : String);
      procedure Sever;
      procedure Sever (Name : String; Comments_For_History : String);
      procedure Imports;
      procedure Imports (Views_To_Import      : String;
                         Into_View            : String;
                         Only_Change_Existing : Boolean);
      procedure Properties;
   end Cm;

   ----------------
   -- Debug Menu --
   ----------------

   package Debug is
      procedure Go;
      procedure Stop;
```





```
      procedure Step;
      procedure Step_Local;
      procedure Step_Returned;
      procedure Break_Here;
      procedure Break;
      procedure Break (Location  : String;
                       Count     : Natural;
                       In_Task   : String;
                       Permanent : Boolean);
      procedure Activate_Break;
      procedure Activate_Break (Break_List : String);
      procedure Remove_Break;
      procedure Remove_Break (Break_List : String; Delete : Boolean);
      procedure Show_All;
      procedure Show_Breakpoints;
      procedure Show_Exception_Handling;
      procedure Show_Stops_And_Holds;
      procedure Show_Stepping;
      procedure Show_Tracing;
      procedure Show_History;
      procedure Context_Control;
      procedure Context_Control (Location : String);
      procedure Context_Evaluation;
      procedure Context_Evaluation (Location : String);
      procedure Information_Tasks;
      procedure Information_Exceptions;
      procedure Information_Rendezvous;
      procedure Information_Space;
      procedure Put_Selection;
      procedure Put_Parameters;
      procedure Stack;
      procedure Source;
      procedure Modify;
      procedure Modify (Variable : String; Value : String);
      procedure Quit;
      procedure Quit (Kill_Job : Boolean; Kill_Debugger : Boolean);
   end Debug;

   ------------------
   -- Session menu --
   ------------------

   package Session is
      procedure Search_List;
      procedure Switches;
      procedure Profile;
      procedure Profile     (Error_Reaction       : String;
                             Line_Width           : Natural;
                             Activity             : String;
```

```
                              Log_Filter             : String;
                              Log_Prefixes           : String;
                              Log_File               : String;
                              Remote_Passwords_File  : String;
                              Remote_Sessions_File   : String);
        procedure Disable_Job;
        procedure Disable_Job (Number : Natural);
        procedure Enable_Job;
        procedure Enable_Job  (Number : Natural);
        procedure Kill_Job;
        procedure Kill_Job    (Number : Natural; Session : String := ""); -- ??
        procedure Users;
        procedure My_Jobs;
        procedure All_Jobs;
        procedure Machine_Information;
        procedure End_Of_Input;
    end Session;

    package Tools is
        procedure Read_Mail;
        procedure Send_Mail;
        procedure Find_Image;
        procedure Image_Directory;
        procedure Macro_Begin;
        procedure Macro_End;
        procedure Macro_Execute;
        procedure Macro_Bind_To_Key;
        procedure Screen_Push;
        procedure Screen_Pop;
        procedure Operator_Backup;
        procedure Operator_Backup       (Start_At   : String;
                                         Kind       : Backup_Kind;
                                         Tape_Drive : String);
        procedure Operator_Verify_Backup;
        procedure Operator_Verify_Backup
                (Start_At : String; Tape_Drive : String);
        procedure Operator_Backup_History;
        procedure Operator_Edit_User;
        procedure Operator_Edit_User    (User_Name           : String;
                                         Password            : String;
                                         Add_To_Groups       : String;
                                         Remove_From_Groups  : String;
                                         Create_Mailbox      : Boolean);
        procedure Operator_Edit_Group;
        procedure Operator_Edit_Group   (Group_Name   : String;
                                         Create       : Boolean;
                                         Delete       : Boolean;
                                         Add_Users    : String;
                                         Remove_Users : String);
```





```
        procedure Operator_Force_Logoff;
        procedure System_Manager_Password_Policy;
        procedure System_Manager_Report;
        procedure System_Manager_Report (Report_Kinds : String;
                                         Start_Time   : String;
                                         End_Time     : String;
                                         Result_File  : String);
        procedure System_Manager_Shutdown;
        procedure System_Manager_Shutdown
                     (At_Time                         : String;
                      Reason                          : String;
                      Message_To_Send_Periodically    : String);
        procedure System_Manager_Cancel_Shutdown;
        procedure Show_Locks;
        procedure Bind_To_Key;
    end Tools;

    package Help is
        procedure Explain_Underline;
        procedure On_Command;
        procedure On_Key;
        procedure On_Keybindings;
        procedure On_Help;
    end Help;

    procedure Load_Image_Palette;
    procedure Load_Debug_Palette;

end Menu_Operations;
```

```
package Message is

    -- Write message in the message window of other user's sessions.
    -- Send selects an individual user; Send_All sends to all logged in users.

    procedure Send     (Who : String; Message : String);
    procedure Send_All (Message : String);

end Message;
```





```
with Calendar;

package Network is

    procedure Show; -- show all currently open connections.

    procedure Close_All; -- close all connections.

    procedure Show_Hosts; -- show all known hosts.

    procedure Show_Host (Host_Name : String := ""); -- show the named host.

    procedure Time (From_Host : in String := "");
    -- Display the time of day, as reported by the given host.

    function Get_Time (From_Host : String) return Calendar.Time;

    function Get_Time_Zone return Integer;

end Network;
```

```
with System_Utilities;
with Terminal;

package Operator is

    procedure Disk_Space;
    procedure Create_User (User     : String  := ">>USER NAME<<";
                           Password : String  := "";
                           Volume   : Natural := 0;
                           Response : String  := "<PROFILE>");
    -- create a user with the given password on volume (0 => Most Available)

    procedure Delete_User (User     : String := ">>USER NAME<<";
                           Response : String := "<PROFILE>");
    -- delete user;  Operator capability is required (or privileged mode)

    procedure Change_Password (User         : String := ">>USER NAME<<";
                               Old_Password : String := "";
                               New_Password : String := "";
                               Response     : String := "<PROFILE>");

    procedure Create_Session (User     : String := ">>USER NAME<<";
                              Session  : String := ">>SESSION NAME<<";
                              Response : String := "<PROFILE>");

    procedure Create_Group (Group    : String := ">>GROUP NAME<<";
                            Response : String := "<PROFILE>");
    -- Create the named group.  It must currently not exist.  It has
    -- no initial members.

    procedure Delete_Group (Group    : String := ">>GROUP NAME<<";
                            Response : String := "<PROFILE>");
    -- Delete the named group.  This operation cannot be used to delete the
    -- group with the same name as an existent user.  Delete_User will
    -- get rid of the group associated with a user.  Acl entries
    -- that refer to a deleted group become inoperative and will be
    -- reclaimed during the next access list compaction.

    procedure Add_To_Group (User     : String := ">>USER NAME<<";
                            Group    : String := ">>GROUP NAME<<";
                            Response : String := "<PROFILE>");
    -- Add the specified user to the specified group.
    -- Operator privilege is required to execute this operation.

    procedure Remove_From_Group (User     : String := ">>USER NAME<<";
                                 Group    : String := ">>GROUP NAME<<";
                                 Response : String := "<PROFILE>");
```





```
    -- Remove the specified user to the specified group.
    -- Operator privilege is required to execute this operation.

    procedure Display_Group (Group    : String := ">>GROUP NAME<<";
                             Response : String := "<PROFILE>");
    -- Display the names of users in the specified group on Current_Output.

    procedure Enable_Privileges (Enable : Boolean := True);
    function  Privileged_Mode return Boolean;
    -- If the caller is a member of the predefined group "privileged",
    -- calling this procedure actually enables or disables the
    -- extra capabilities that such a job can have.  General usage is
    -- to not enable privileged mode unless it is really needed so
    -- as to avoid accidently doing something that would normally be
    -- stopped by access control.  All tasks in the job become
    -- privileged when the mode is enabled.  No output is produced
    -- by any of these procedures.  Failure to acquire privileged mode
    -- is indicated only by the absence of the privileges.  Privileged_Mode
    -- returns false in this case.

    procedure Enable_Terminal  (Physical_Line : Terminal.Port;
                                Response      : String := "<PROFILE>");
    procedure Disable_Terminal (Physical_Line : Terminal.Port;
                                Response      : String := "<PROFILE>");
    -- (Dis)allow login on the specified terminal port

    procedure Force_Logoff (Physical_Line  : Terminal.Port;
                            Commit_Buffers : Boolean := True;
                            Response       : String  := "<PROFILE>");
    -- Force a user off of the specified terminal.
    -- Try to commit modified buffers if Commit_Buffers is true.
    -- Each of these operations requires operator capability.

    procedure Set_System_Time (To_Be    : String := ">>TIME<<";
                               Response : String := "<PROFILE>");
    -- Requires operator capability.

    procedure Shutdown_Warning (Interval : Duration := 3600.0);
    -- Note that Interval is rounded to the nearest minute.  Less than
    -- 30.0 is rounded to 0.

    function Get_Shutdown_Interval return Duration;

    procedure Archive_On_Shutdown (On : Boolean := True);
    function  Get_Archive_On_Shutdown return Boolean;
    -- Archive_On_Shutdown causes the next shutdown to store internal
    -- state in "archive" form, allowing upgrades and conversion of
    -- internal data structures.  It typically takes several hours to
    -- complete a shutdown or restart with archive conversions.
```

```
procedure Show_Shutdown_Settings;
procedure Cancel_Shutdown;

procedure Shutdown (Reason      : String :=
                      "COPS";            -- Customer operations
                    Explanation : String := "Cause not entered");
-- Shutdown the machine.  Enter the cause and explanation in the system
-- log, wait for the Shutdown interval to expire, then log users
-- off and shutdown the machine.
-- Enter Reason = "?" to get list of reasons.  The shutdown will not
-- happen unless Reason is a legal value.

procedure Explain_Crash;
-- Reads a shutdown cause and explanation from current input and enters
-- these in the machine's error log.  Corresponds to the information
-- entered by shutdown.

procedure Limit_Login (Sessions : Positive := Positive'Last);
procedure Show_Login_Limit;
function  Get_Login_Limit return Positive;
-- Control over the number of simultaneously active user sessions

procedure Internal_System_Diagnosis;
-- Requires Operator capability

subtype Days is Positive;

procedure Set_Password_Policy
             (Minimum_Length  : Natural := 0;
              Change_Warning  : Days    := Operator.Days'Last;
              Change_Deadline : Days    := Operator.Days'Last);
-- Passwords must be at least Minimum_Length characters long.
-- Passwords must be changed periodically.  Change_Warning days after the
-- last change, the user will be notified at login that the account
-- password should be changed.  Change_Deadline days after the last change,
-- the user will be unable to login without changing the password.
-- The default values introduce no restrictions.
-- Requires Operator capability.

procedure Show_Password_Policy
             (For_User : String := System_Utilities.User_Name);
-- Show the current policy along with the expiration dates for the user(s)
-- specified.

function Get_Minimum_Password_Length return Natural;
function Get_Password_Warning        return Days;
function Get_Password_Deadline       return Days;
-- Return the values last set by Set_Password_Policy.
```





```
    function Get_User_Warning
                (For_User : String := System_Utilities.User_Name) return String;
    function Get_User_Deadline
                (For_User : String := System_Utilities.User_Name) return String;
    -- Return the image of the date on which the Warning (Deadline) for
    -- changing the password will be reached.  Can be processed by
    -- Time_Utilities if necessary to have numeric value.  Format is mm/dd/yy.

end Operator;
```

```
with Machine;
with Simple_Status;

package Program is

    subtype Job_Id    is Machine.Job_Id;
    subtype Condition is Simple_Status.Condition;

    procedure Run (S        : String := "<SELECTION>";
                   Context  : String := "$";
                   Response : String := "<PROFILE>");
    -- sets root of job_garbage_unit, dangerous to run concurrently in one job

    procedure Run_Job (S        : String   := "<SELECTION>";
                       Debug    : Boolean  := False;
                       Context  : String   := "$";
                       After    : Duration := 0.0;
                       Options  : String   := "";
                       Response : String   := "<PROFILE>");

    procedure Create_Job (S        :        String   := "<SELECTION>";
                          Job      : out    Job_Id;
                          Status   : in out Condition;
                          Debug    :        Boolean  := False;
                          Context  :        String   := "$";
                          After    :        Duration := 0.0;
                          Options  :        String   := "";
                          Response :        String   := "<PROFILE>");

    -- Run_Job and Create_Job are identical except that Create_Job
    -- returns the job number of the job just started and a status indicating
    -- success or failure.
    --
    -- Debug => True starts the debugger on the newly started job
    --
    -- The following options are defined:
    --
    --      Output       Specifies the name of the new job's output file.
    --      Input        New job's standard input file.
    --      Error        New job's error file.
    --                   File names given are resolved in the directory
    --                   context of the caller, NOT the Context parameter.
    --
    --      User         Causes the new job to run with the identity
    --                   of this user.  Password must be valid unless
    --                   running job is privileged.  If not specified
    --                   new job runs with same identity as parent.
```





```
    --
    --      Password     Password used in conjunction with User.
    --
    --      Session      Session used in conjunction with User.


    function Started_Successfully (Status : Condition) return Boolean;
    -- True => Job has been started successfully

    procedure Wait_For (Job : Job_Id);
    -- Wait until the job specified has terminated.

    procedure Change_Identity (To_User  :        String := "";
                               Password :        String := "";
                               Options  :        String := "";
                               Status   : in out Condition);

    -- Change the identity of the calling job to the specified
    -- user.  Password must be supplied and correct unless the
    -- caller is privileged.  Options specifies additional
    -- characteristics to be changed.  If To_User is null,
    -- the options are processed.

    -- Note that only the access control identity is changed.
    -- The actual username and session of the job are NOT changed.
    -- This operation should never be used to change identity and
    -- execute untrusted code.  The identity can always be changed
    -- back to the original job identity.

    -- Options presently defined are:
    --      Privileged          -- enable privileged mode.  The specified user
    --                          -- must be a member of group PRIVILEGED
    --      Privileged => False -- disable privileged.  No effect if caller
    --                          -- was not already privileged.
    --      Restore_Identity    -- Change the identity back to the original
    --                          -- identity of the job.  Password is not
    --                          -- required to do this.

    function Current (Subsystem  : String := ">>SUBSYSTEM NAME<<";
                      Unit       : String := ">>PROCEDURE NAME<<";
                      Parameters : String := "";
                      Activity   : String := "<ACTIVITY>") return String;

    -- Constructs a procedure call suitable for Run or Run_Job that references
    -- the appropriate view, has the appropriate quotes, etc.  Unit name is
    -- the Ada name to be called; it will be found anywhere in the
    -- view.  If the procedure being called has parameter they may be
    -- provided.  If the current view of !Subsystem is Rev8_4_0 and package
    -- View is in the Commands directory, then:
```

```
    --
    -- Current ("!Subsystem", "View.Initial",
    --          "(P1 => ""!New_Tool"", P2 =>1)") returns:
    --
    -- "!Subsystem.Rev8_4_0.Units.Commands".View.Initial
    --       (P1 => "!New_Tool", P2 => 1);

end Program;
```





```
with Directory;

package Queue is

    procedure Print (Name    : String := "<IMAGE>";
                     Options : String := "<DEFAULT>";
                     Banner  : String := "<DEFAULT>";
                     Header  : String := "<DEFAULT>";
                     Footer  : String := "<DEFAULT>");

    procedure Print_Version (The_Version : Directory.Version;
                             Options     : String := "<DEFAULT>";
                             Banner      : String := "<DEFAULT>";
                             Header      : String := "<DEFAULT>";
                             Footer      : String := "<DEFAULT>");
    --
    -- The Print and Print_Version procedures are the provided user interfaces
    -- for sending files to a printer.  They queue object(s) to be printed and
    -- echo request IDs in the message window with corresponding objects.
    --
    --
    -- NOTE : if a value is not specified for a parameter (<DEFAULT> is
    --        indicated) then the value supplied in the session switch
    --        file is used; if a session switch is not defined or
    --        unavailable then the default specified here is used.
    --
    -- BANNER:  String to be used on the banner page
    --          (truncated at 11 characters), user's id is the default
    --          Specifying the null string ("") will inhibit the generation
    --          of a banner page.
    --
    -- HEADER:  User supplied page header; default is none.
    --
    -- FOOTER:  User supplied page footer; default is none.
    --          (see R1000 documentation for headers or footers
    --          containing Line-Feeds or exceeding Width characters)
    --
    -- OPTIONS: A form parameter for setting various formatting and
    --          spooling options; default is "Format=>(Wrap, System_Header").
    --
    --
    -- The Currently available Options and semantic rules for these options are
    -- described at the end of this package and in detail in the documentation.
    --

    procedure Cancel (Request_Id : Positive);
    -- cancels a request by ID obtained from Print or Queue
```

```
-- Extreme measures for wedged spooler
procedure Kill_Print_Spooler;
procedure Restart_Print_Spooler;

-- The remaining procedures do NOT use any session switches.

subtype Class_Name is String;

All_Classes         : constant Class_Name := "all";
All_Spooler_Devices : constant String     := "all";

-- The following procedures provide information on the state
-- of the print spooler.

procedure Display (Class : Class_Name := "all");
-- print the current contents of the Queue

procedure Classes (Which        : Class_Name := "all";
                   Show_Devices : Boolean    := True);
-- Display information about one or all classes

procedure Devices (Which        : String  := "all";
                   Show_State   : Boolean := True;
                   Show_Classes : Boolean := True);
-- Display information about one or all devices


-- The following procedures are used to define queues in the spooler.

procedure Create  (Class : Class_Name := "");
procedure Destroy (Class : Class_Name := ""; Reroute : Class_Name := "");
-- Create/Destroy a class.
-- When a class is destroyed any requests in that class are rerouted to
-- the class specified (the default class if none is specified).

procedure Default (Class : Class_Name := "");
-- set Default Class or print current Default (if null string provided)

procedure Add (Device : String := ""; Options : String := "XON_XOFF");
-- Options :
--        XON_XOFF, RTS, DTR indicate what flow control is to be used.
--        Host => name indicates that a telnet connection is to be used
--        If Host is given, Socket may be specified:  Socket => (0, 23).
--
--        Options can also have the value FTP.  In this case, Device is
--        the name of a file whose first line is a host name, whose
--        second line is a directory name, whose third line is a suffix
--        to append to each file name, and whose fourth line is the
--        name of a remote passwords file.  Each print request will
```





```
--        be transferred to the specified host and directory using an
--        FTP login for the host from the specified remote passwords file.
--        The directory name in the file must have any trailing punctuation
--        so that a simple filename can be concatenated to it.  A log file
--        is created in !Machine.Queues.Ftp under the device name (which
--        is the simple name of the device file) to record any FTP
--        problems.

procedure Remove (Device : String := ""; Immediate : Boolean := False);
-- Associate/Disassociate a device with the print spooler.

procedure Register   (Device : String := ""; Class : Class_Name := "");
procedure Unregister (Device : String := ""; Class : Class_Name := "");
-- Associates/disassociates a class and a device.
-- If a class is not associated with a device then items spooled to that
-- class can not be printed.

procedure Enable  (Device : String := "all");
procedure Disable (Device : String := ""; Immediate : Boolean := False);
-- Allows/Disallows printing on device(s)

--------------------------------------------------------------------------------
--
-- Description of the Options available for Print and Print_Version.
--
-- The following is a list of legal options.
--
--     BANNER_PAGE_USER_TEXT => text
--
--           Text appears on the banner page (if one is generated) after the
--           "Banner".
--
--     CLASS => class name
--
--           Class to which printout is to be queued.  (default is <DEFAULT>)
--
--     COPIES => number
--
--           Number of copies of the printout (default is 1)
--
--     LENGTH => number
--           Number of printed lines available on a page (default is 60).
--
--     NOTIFY =>  Mail | MESSAGE | None
--
--           Type of notification desired upon completion of the print request.
--
--     ORIGINAL_RAW => true | FALSE
```

```
--
--          DO NOT make a copy of the file to be printed.  Notification is set
--          to Message and each file is spooled separately with a banner page.
--          Class must NOT be Remote.
--
--     PostScript => ( <PostScript_Options> )
--
--          Specify to print using PostScript rules.  PostScript options and
--          functionality are described below.  The null options string, (),
--          invokes the PostScript printer with default parameters.
--
--     FORMAT => ( <Format_Options> )
--
--          The printer is to be treated as a conventional Ascii device with
--          the specified options, which are described below.  FORMAT with the
--          null options string, (), is the default unless other options are
--          specified.
--
--     RAW => true | FALSE
--
--          DO NOT interpret the input.  This option can be useful for
--          preformatted text or binary data.
--
--     SPOOL_EACH_ITEM => true | FALSE
--          Spool each file as a separate job.
--
-- Exactly one of the Format, Original_Raw, PostScript, or Raw can be supplied
-- for any print request.  If any of these are specified in the Options
-- parameter, then the corresponding session switch is ignored.
--
-- <Format_Options>
--     The following is a list of legal <format_options>.  Unless
--     otherwise specified, the Boolean options are assumed to be False.
--
--     NUMBERING => true | FALSE
--
--          Provide line numbering.
--
--     SYSTEM_HEADER  => number
--
--          Produce a system page header on each page.
--
--     TAB_WIDTH => number
--
--          Number of spaces to replace a tab character (Ascii.HT) with
--          (default is 8).  0 causes tabs to be sent to the printer.
--
--     TRUNCATE => true | FALSE
--
```





```
--          Truncate lines longer than Width.
--
--     WIDTH => number
--
--          Number of characters to be printed on a line (default is 80).
--
--     WRAP => true | FALSE
--
--          Wrap lines longer than Width.
--
--
--
-- <PostScript_Options>
--
--     FORMAT => PostScript | plain_text | fancy | letter | image | AUTOMATIC
--
--          Broadly specifies how the file is to be printed, whether the file
--          to be printed is a PostScript program (such as generated by a text
--          formatter) or plain text that must be prepared for printing.
--
--          AUTOMATIC is the default, in which case the file is looked at to
--          determine it's type.  If the file begins with a % it is processed
--          as a PostScript program, if it begins with Ascii.Nul it is printed
--          as an IMAGE, otherwise it is processed as PLAIN_TEXT.
--
--          LETTER format is similar to PLAIN_TEXT except that the defaults for
--          TWOUP, BORDER, DATE, FILENAME, WRAP, and NUMBER are all False.
--
--          FANCY format is similar to PLAIN_TEXT, except that Ada
--          reserved words are emboldened and comments are Italicized.
--
-- The following options apply to both PostScript and Plain_Text files.
--
--     STATS => TRUE | false
--
--          Causes statistics on the size of files and their print speed to be
--          included in job messages.
--
--     FLOW => true | FALSE
--
--          By default (FLOW=false), each file printed starts on a new sheet
--          of paper.  When FLOW is true, however, a file will start on the
--          right half of a sheet if not occupied by the previous file.
--          Setting FLOW to true forces TWOUP = true and REVERSED = false.
--
--     REVERSED => TRUE | false
--
--          If true, the default, the pages are reversed before printing so
--          that the stack of pages in the printer's output tray are in the
```

```
--          correct order with the first page on top.  If false, the pages
--          will be printed in the order they appear in the file.
--
--     CHATTY => TRUE | false
--
--          If true, the default, messages will be generated in the message
--          window before accessing each file in the print request When false,
--          PostScript issues a message only when all files have been printed
--          and under error conditions.
--
--     PAGES = <integer> [..<integer>]
--
--          Specifies the range of pages to be printed.  The first page in the
--          file is numbered 1.  The default is to print all pages in the
--          file.  If only one integer is given, that one page is printed.
--
--     HEADER => true | FALSE
--
--          If true, a header page is printed that identifies the file that is
--          being printed and the circumstances of its printing.
--
--     TWOUP => TRUE | false
--
--          If true, two file pages are printed per sheet of printer paper.
--          The image of each page is 2/3 the size of a full page.  The
--          default for this option for plain text files is true; for
--          PostScript files, it is false.
--
--     OUTLINES => TRUE | false
--
--          If true, a solid box is drawn around the text for each page.
--          BORDER is an alternative name for this option. The default for
--          this option for PLAIN_TEXT files is true; for PostScript files, it
--          is false.
--
--     DATE => true | false
--
--          If true, the time and date at the time of queueing is printed in
--          the lower-left corner of each page, outside the outline box if
--          present.  The default for this option for plain text files is
--          true; for PostScript files, it is false;
--
--     FILENAME => true | false
--
--          If true, the full name of the file is printed in the upper-left
--          corner of each page, outside the outline box if present. The
--          default for thisoption for plain text files is true; for
--          PostScript files, it is false;
--
```





```
-- The following options apply to PLAIN_TEXT files only.  All combinations are
-- valid.
--
--     NUMBER => TRUE | false
--
--          If true, a page number is printed in the upper right corner of
--          each page, outside the outline box, if present.  The numbering
--          starts again at 1 for each file printed.
--
--     WIDE => true | FALSE
--
--          If true, each page is printed in landscape orientation, i.e., with
--          the lines of text parallel to the longer side of the page.
--
--     RULES => true | FALSE
--
--          If true, faint dashed lines are drawn every other line of the
--          output.
--
--     SIZE = <integer>
--     SPACING = <integer>
--
--          Specifies the point-size of the typeface used to generate the
--          output and the vertical spacing of each line measured in points.
--          These point sizes determine the number of lines per page and the
--          number of characters per line according to the following formulae:
--
--          For the WIDE format:
--
--              Lines/Page       =   540 / Spacing
--              Characters/Line  =  1200 / Size
--
--          For the ~WIDE (narrow) format:
--
--              Lines/Page       =   720 / Spacing
--              Character/Line   =   900 / Size
--
--          The default SPACING is SIZE + 1; The default SIZE is 11 (yielding
--          a SPACING of 12). In ~WIDE format this allows for 60 lines of
--          81-character lines.
--
--     FONT = <font name>
--
--          Specifies the typeface to be used in printing the file.  Any
--          built-in PostScript font may be specified.  The default is
--          /Courier-Bold.  If <font-name> begins with a '/', PostScript
--          assumes the font is already resident and uses the <font name> to
--          define the font to use.  If <font name> does not begin with '/',
--          PostScript assumes it is the name of a file containing PostScript
```

```
--          for a downloadable font.  This file is sent to the printer before
--          any files are processed by PostScript.  The simple name of the
--          file, capitalized as it appears in the font option, is used to set
--          the font for the plain_text file.
--
--    CHOP => true | FALSE
--
--          If false, the default, a line longer than the line length defined
--          by the above formulae is broken at the rightmost blank within the
--          line and the extra text is printed on the next line justified to
--          the right margin.  If true, long input lines will be clipped at
--          the boundaries of the imageable area (7.5 x 10.0 inches).
--
-- The following options affect the IMAGE format:
--
--    X = number
--    Y = number
--
--          Specifies, in inches,  the coordinates of the lower left corner of
--          the first image.  The default coordinate is (0.25, 0.25), a point
--          1/4 inch from the lower left corner of the paper.
--
--    DX = number
--    DY = number
--
--          Specifies the offset from the previous image coordinate to the
--          coordinate for the next image.  Dx is added to the X coordinate
--          for each successive image until the resulting coordinate would be
--          outside the bounds of the paper, at which time X is reset to its
--          original value and Dy is added to the Y coordinate.  When the Y
--          coordinate exceeds the bounds of the paper, a new page is started
--          at the original X, Y coordinate.
--
--    WIDTH = number
--    HEIGHT = number
--
--          Specifies the maximum width and height allowed for the image. The
--          default values specify a full page image.
--
--    DISTORT => true | FALSE
--
--          If true, the image will be magnified so that the image fills
--          exactly the box defined by width and height.  If false, the image
--          will be magnified as large as possible while retaining the aspect
--          ratio of the image.
--
--    ASPECT => number
--
--          Overrides the aspect ratio of the image.
```





```
--
--    CAPTION => text
--
--          Text to be rendered below the printed image.
--
--    PROLOG => text
--    EPILOG => text
--
--          PostScript code to be sent before and after each image.  The
--          following regards action taken on files when the PostScript option
--          is specified and a list of legal <PostScript_options>.
--
-- The following "commands" will be regonized when embedded in an input file
-- when using a PostScript printer.  These commands must begin in the first
-- column of a line and must be capitalized as shown above.
--
--    %%INCLUDE naming-expression
--
--          Recognized in all formats except Image.  Causes the files named in
--          the expression to be opened and processed as if they were part of
--          the input file.  %%INCLUDEs can be nested to 10 deep.
--
--    %%ASCII naming-expression
--
--          Recognized in PostScript format only.  Causes the named files to
--          be opened and sent to the destination without further
--          interpretation by PostScript (nested commands are ignored).
--
--    %%BINARY naming-expression
--
--          Recognized in PostScript format only.  Causes the named files to
--          be opened and sent to the destination as strings of hexadecimal
--          numbers.  The %%BINARY command should be preceded by PostScript
--          code that will prepare the printer to receive hexadecimal data.
--
-------------------------------------------------------------------------------

end Queue;
```

```
package Remote is

    procedure Run (Machine      : String := ">>machine_name<<";
                   Command      : String := "<IMAGE>";
                   File_Context : String := "$";
                   Run_Context  : String := "<DEFAULT>";
                   Options      : String := "";
                   Response     : String := "<PROFILE>");

    -- Run a command on another machine.
    -- The default naming context in which the command will run is
    -- given by Run_Context. <DEFAULT> means use the same context
    -- as that on the current machine.
    -- If that doesn't exist on the target ! is used.
    -- File_Context is the default context for opening files
    -- on the target.

    ---------------------------------------------------------------------------

    procedure Show (Machine     : String := ">>machine_name<<";
                    Object_Name : String := "<CURSOR>";
                    Response    : String := "<PROFILE>");

    -- type the contents of an object which is on another machine.

    ---------------------------------------------------------------------------

end Remote;
```





```
package Remote_Passwords is

    -- The commands in this package can be used to add, change, delete,
    -- and display entries in a remote passwords file.  By default,
    -- these commands access the remote passwords file for the current
    -- session.  The For_Session parameter in these commands allows you
    -- to specify a non-default session; in this case, the operation
    -- applies to the remote passwords file of the given session.
    --
    -- A remote passwords file is a text file that specifies
    -- the username and password to be used when accessing a remote
    -- host.  This file may contain one or more entries of the following
    -- form:
    --
    --      HOST_NAME      USERNAME       PASSWORD_VALUE
    --
    -- HOST_NAME must identify a machine to which the user has access.
    -- USERNAME must be a valid username on the specified machine.
    -- PASSWORD_VALUE must be one of the following:
    --
    --     "Ada-style_quoted_string"
    --     <PROMPT>
    --     <DES:hexadecimal_string>
    --     ""
    --
    -- For example, a remote passwords file might contain entries such
    -- as the following:
    --
    --     machine1  username1   password1
    --     machine1 "username1" "password1"
    --     machine2  guest       ""
    --     machine3  username3   <PROMPT>
    --     machine4  operator    <DES:29A1EB449C1A03F6>
    --
    -- In this example, the two entries for machine1 are equivalent.
    -- The entry for machine2 provides for a guest user who has no
    -- password; the entry for machine3 causes username3 to be prompted
    -- for a password; the entry for machine4 indicates that the
    -- password for operator has been encrypted using DES.

    type Encryption_Method is (None, Hex, Des);
    -- Represents the forms of encryption that can be applied to the
    -- passwords entered into the remote passwords file.  The Encryption
    -- parameter in these commands allows you to specify the type of
    -- encryption to be used when a password is added or changed.  The
    -- value None causes the clear text password to be entered without
    -- encryption.
```

```
   procedure Create
               (New_File        : String := ">>REMOTE PASSWORDS FILE<<";
                Source_File     : String := "";
                Source_Password : String := "<PROMPT>";
                Encryption      : Encryption_Method := Remote_Passwords.Des;
                Response        : String := "<PROFILE>");
   --
   -- Creates a new remote passwords file using Source_File and
   -- Source_Password to initialize the contents of the file.  After
   -- Source_File has been decrypted with Source_Password, New_File is
   -- created and populated with encrypted source entries for the
   -- current user as specified by the Encryption parameter.  Any
   -- source entries that cannot be decrypted by Source_Password are
   -- omitted from New_File.

   procedure Set_Default (Existing_File : String := "<IMAGE>";
                          For_Session   : String := "";
                          Response      : String := "<PROFILE>");
   --
   -- Establishes Existing_File as the default remote passwords
   -- file for For_Session.  This information is stored in the
   -- Profile.Remote_Passwords switch of the session switch file
   -- associated with For_Session.  When no switch file exists, it
   -- is created before storing the value.

   procedure Add (New_Hostname : String            := ">>REMOTE HOST<<";
                  New_Username : String            := ">>REMOTE USER<<";
                  New_Password : String            := "<PROMPT>";
                  Encryption   : Encryption_Method := Remote_Passwords.Des;
                  For_Session  : String            := "";
                  Response     : String            := "<PROFILE>");
   --
   -- Adds an entry for New_Hostname to the remote passwords file of
   -- the calling user.  When an entry for New_Hostname already exists,
   -- an error message is generated and the remote passwords file is
   -- left unchanged.

   procedure Change (Existing_Hostname : String := ">>REMOTE HOST<<";
                     New_Username      : String := ">>REMOTE USER<<";
                     New_Password      : String := "<PROMPT>";
                     Encryption        : Encryption_Method
                         := Remote_Passwords.Des;
                     For_Session       : String := "";
                     Response          : String := "<PROFILE>");
   --
   -- Changes the entry for Existing_Hostname in remote passwords file
   -- of the calling user.  When the entry for Existing_Hostname does
   -- not exist, it is Add'ed.
```





```
   procedure Delete (Existing_Hostname : String := ">>REMOTE HOST<<";
                     For_Session       : String := "";
                     Response          : String := "<PROFILE>");
   --
   -- Removes the entry for Existing_Hostname from the remote passwords
   -- file of the calling user.  When an entry for Existing_Hostname
   -- does not exist, a warning message is generated.

   procedure Show_Encryption
               (Of_Password : String            := "<PROMPT>";
                Encryption  : Encryption_Method := Remote_Passwords.Des);
   --
   -- Displays the encrypted form of Of_Password for the calling user
   -- in the message window.  The value displayed is identical to what
   -- would be directly reflected in the remote passwords file if the
   -- same password was used in the file manipulation operations that
   -- are listed below.

   procedure Update (Old_Password : String := "<PROMPT>";
                     For_Session  : String := "";
                     Response     : String := "<PROFILE>");
   --
   -- Updates the remote passwords file after the user's Rational
   -- password has been changed.


   ---------------------------
   -- Remote Passwords File --
   ---------------------------

   function Get_Default (For_Session : String := "";
                         Response    : String := "<WARN>") return String;
   --
   -- Returns a naming expression for the default remote passwords
   -- file for For_Session.  When errors occur while resolving this
   -- file, the value "<>" is returned.

end Remote_Passwords;
```

```
with Machine;

package Scheduler is

    subtype Job_Id       is Machine.Job_Id;
    subtype Cpu_Priority is Natural range 0 .. 6;
    subtype Milliseconds is Long_Integer;

    procedure Disable (Job : Job_Id);
    procedure Enable  (Job : Job_Id);
    function  Enabled (Job : Job_Id) return Boolean;

    function Get_Cpu_Priority (Job : Job_Id) return Cpu_Priority;

    type Job_Kind is (Ce, Oe, Attached, Detached, Server, Terminated);
    function Get_Job_Kind (Job : Job_Id) return Job_Kind;

    type Job_State is (Run, Wait, Idle, Disabled, Queued);
    function Get_Job_State (Job : Job_Id) return Job_State;

    -- returns the current state of job.
    --   RUN:      the job is currently runnable
    --   WAIT:     the job is runnable but being withheld by the scheduler.
    --   IDLE:     the job isn't using cpu time and has no unblocked tasks.
    --   DISABLED: an external agent has disabled the job from running.
    --   QUEUED:   the job is DETACHED and must wait for another to complete.

    function Get_Cpu_Time_Used (Job : Job_Id) return Milliseconds;
    -- returns the number of milliseconds of cpu time used by the job.
    -- belongs on the previous page, put here for compatability reasons

    function Disk_Waits (Job : Job_Id) return Long_Integer;
    -- returns the number of disk_waits the job has done since last initialized

    function Working_Set_Size (Job : Job_Id) return Natural;
    -- returns the number of pages in the job's working set.

    subtype Load_Factor is Natural;
    -- for run queues, number of tasks * 100

    procedure Get_Run_Queue_Load (Last_Sample     : out Load_Factor;
                                  Last_Minute     : out Load_Factor;
                                  Last_5_Minutes  : out Load_Factor;
                                  Last_15_Minutes : out Load_Factor);
    -- number of runnable tasks * 100
```





```
    procedure Get_Disk_Wait_Load (Last_Sample     : out Load_Factor;
                                  Last_Minute     : out Load_Factor;
                                  Last_5_Minutes  : out Load_Factor;
                                  Last_15_Minutes : out Load_Factor);
    -- number of tasks waiting for a page on the disk wait queue * 100

    procedure Get_Withheld_Task_Load (Last_Sample     : out Load_Factor;
                                      Last_Minute     : out Load_Factor;
                                      Last_5_Minutes  : out Load_Factor;
                                      Last_15_Minutes : out Load_Factor);
    -- returns the average number of tasks withheld from running by
    -- the scheduler * 100. In this call LAST_SAMPLE is the number of tasks
    -- held at the last 100ms scheduling cycle.

    procedure State;
    -- print scheduler state

    procedure Display (Show_Parameters : Boolean := True;
                       Show_Queues     : Boolean := True);
    -- display current scheduler state and queues

    type Job_Descriptor is
        record
            The_Cpu_Priority             : Cpu_Priority;
            The_State                    : Job_State;
            The_Disk_Waits               : Long_Integer;
            The_Time_Consumed            : Milliseconds;
            The_Working_Set_Size         : Natural;
            The_Working_Set_Limit        : Natural;
            The_Milliseconds_Per_Second  : Natural;
            The_Disk_Waits_Per_Second    : Natural;
            The_Maps_To                  : Job_Id;
            The_Kind                     : Job_Kind;
            The_Made_Runnable            : Long_Integer;
            The_Total_Runnable           : Long_Integer;
            The_Made_Idle                : Long_Integer;
            The_Made_Wait                : Long_Integer;
            The_Wait_Disk_Total          : Long_Integer;
            The_Wait_Memory_Total        : Long_Integer;
            The_Wait_Cpu_Total           : Long_Integer;
            The_Min_Working_Set_Limit    : Long_Integer;
            The_Max_Working_Set_Limit    : Long_Integer;
        end record;

    function Get_Job_Descriptor (Job : Job_Id) return Job_Descriptor;
    -- use to get a consistent snapshot of a job's statistics.

    generic
        with procedure Put (Descriptor : Job_Descriptor);
```

```
procedure Traverse_Job_Descriptors (First, Last : Job_Id);
-- use to get a consistent, efficient snapshot of a range of
-- job's statistics.


procedure Set (Parameter : String := ""; Value : Integer);
function  Get (Parameter : String) return Integer;

-- Programmatic versions of set and display
--      initial parameters          Default      Units
--      CPU_Scheduling                    1       1 or 0 (true or false)
--      Percent_For_Background           10       %
--      Min_Foreground_Budget          -250       milliseconds (-5000..0)
--      Max_Foreground_Budget           250       milliseconds (0..5000)
--      Withhold_Run_Load               130       load * 100
--      Withhold_Multiple_Jobs            0       1 or 0 (true or false)

--      Memory_Scheduling                 1       1 or 0 (true or false)
--      Environment_Wsl               11000       pages
--      Min_Ce_Wsl                      400       pages
--      Max_Ce_Wsl                     1000       pages
--      Min_Oe_Wsl                      250       pages
--      Max_Oe_Wsl                     2000       pages
--      Min_Attached_Wsl                 50       pages
--      Max_Attached_Wsl               2000       pages
--      Min_Detached_Wsl                 50       pages
--      Max_Detached_Wsl               4000       pages
--      Min_Server_Wsl                  400       pages
--      Max_Server_Wsl                 1000       pages
--      Daemon_Wsl                      200       pages
--      Wsl_Decay_Factor                 50       pages
--      Wsl_Growth_Factor                50       pages
--      Min_Available_Memory           2048       pages
--      Page_Withdrawl_Rate               1       n*640 pages/sec (n in 0..64)

--      Disk_Scheduling                   1       1 or 0 (true or false)
--      Max_Disk_Load                   250       Load_Factor
--      Min_Disk_Load                   200       Load_Factor

--      Foreground_Time_Limit            60       seconds
--      Background_Streams                3
--      Stream_Time N                2,5,20       minutes
--      Stream_Jobs N                 3,0,0       jobs
--      Strict_Stream_Policy              0       1 or 0 (true or false)

procedure Set_Job_Attribute (Job       : Job_Id;
                             Attribute : String := "Kind";
                             Value     : String := "Server");
```





```
function  Get_Job_Attribute
            (Job : Job_Id; Attribute : String := "Kind") return String;

-- These interfaces exist to deal with ongoing changes to scheduler
-- characteristics without requiring new procedures.
--
-- The default parameters to Set_Job_Attributes make the indicated job
-- a server.
--
-- See the documentation for other attributes.


procedure Set_Wsl_Limits         (Job : Job_Id; Min, Max : Natural);
procedure Get_Wsl_Limits         (Job : Job_Id; Min, Max : out Natural);
procedure Use_Default_Wsl_Limits (Job : Job_Id);

-- Each class of job has a default for working set min and max.
-- Set_Parameter lets you change the default value. Set_Wsl_Limits lets
-- you override the default for a specific job.  Use_Default_Wsl_Limits
-- restores the values to the defaults, cancelling any prior Set_Wsl_Limits
-- call.
-- Get_Wsl_Limits returns the current values for a specific job.
-- Min and Max specify the range (in number of pages) in which the
-- working set limit is set.  The scheduler chooses the working set
-- limit based on prevailing conditions on the machine.  If Min and
-- Max are the same, the a fixed limit is specified.
-- Min must be less than or equal to Max and Max less than the memory size.
-- Error messages are sent to an output window in the case of errors.
-- No message of any kind if success.

end Scheduler;
```

```
package Search_List is

    -- Conceptually a search list is a sequence of component names of
    -- libraries.  A component name could have wild characters, and would
    -- therefore resolve to many libraries.  The resolution of a name
    -- depends on the resolution of the libraries, order being important.
    -- Furthermore, the resolution of a component name or an Ada name
    -- depends on the context in which such resolution is done.  For
    -- instance, the component name "$" meaning enclosing library resolves
    -- to different libraries depending on the current context.

    -- A separate image comes up for each Edit with different parameters.
    -- Most commands take in defaulted Session and User parameters.  The
    -- defaults refer to the present user and session.

    procedure Display (Session : String := ""; User : String := "");

    -- Displays the Session Search List Components in a text-io image.

    procedure Display_Libraries;

    -- Displays the resolution of all the Libraries of the Search List
    -- in the present context in a text-io image.

    procedure Show_List (Session : String := ""; User : String := "");

    -- Shows the Session Search List

    procedure Show_Item (Component : String := "<CURSOR>");

    -- Displays the library indicated by a Search List component provided it
    -- resolves to a unique library.  By default, displays the library at
    -- the cursor.

    procedure Set_Up (Component : String := ">>SEARCH LIST<<";
                      Session   : String := "";
                      User      : String := "");

    -- Initialize Search List.  Replaces entire previous contents.

    procedure Reset_To_System_Default
                 (Session : String := ""; User : String := "");

    -- Resets to system default search list.
```





```
    procedure Add (Component : String  := ">>LIBRARY NAME<<";
                   Position  : Integer := Integer'Last;
                   Session   : String  := "";
                   User      : String  := "");

    -- Adds Component in the indicated Position in the Search
    -- List Components image.  If defaulted, and cursor is on the
    -- Search List image, then that is the location of the addition
    -- Otherwise, addition is at end.

    procedure Replace (New_Component : String := ">>LIBRARY NAME<<";
                       Old_Component : String := "<SELECTION>";
                       Session       : String := "";
                       User          : String := "");

    -- Replace Old_Component (the component indicated by selection is the
    -- default) by New_Component in Search List image.

    procedure Delete (Component : String := "<SELECTION>";
                      Session   : String := "";
                      User      : String := "");

    -- Remove Component (the component indicated by the current selection is
    -- the default) from the Search List image.

    procedure Release;

    -- Removes current image from the screen

    procedure Save (File_Name : String := ">>FILE NAME<<";
                    Session   : String := "";
                    User      : String := "");

    -- Save the search list of the given user's session

    procedure Revert (File_Name : String := "";
                      Session   : String := "";
                      User      : String := "");

    -- Revert the search list for the given user's session from the named
    -- file.  If the file name is defaulted, the search list is reverted
    -- from the permanent search list maintained for this user's session

end Search_List;
```

```
package Speller is

    -----------------------------------------------------------------
    -- Top level commands:                                         --
    --                                                             --
    -- These are the standard commands which are used to invoke the --
    -- speller, and to bring up the speller window.                --
    --                                                             --
    -----------------------------------------------------------------

    procedure Check_Text (Data : String := "<TEXT>");
    --
    -- Check the selected text, or (if no selection exists or the
    -- cursor is not in the selection) the word nearest the cursor.

    procedure Check_Image;
    --
    -- Have the speller underline all of the unknown words in the
    -- current image.  Automatic corrections will be made immediately.

    procedure Check_File (Name : String := "<IMAGE>");
    --
    -- Run the speller in a batch mode over the named file.
    -- Send all unknown words to the log file.

    procedure Speller_Window (In_Place : Boolean := False);
    --
    -- Bring up the speller window.
    -- Dictionary and Switch settings may be altered in this window
    -- before running the Check_Image procedure above.


    ----------------------------------------------------------------
    -- Keys:                                                      --
    --                                                            --
    -- These procedures are meant to be bound to keys, and invoked --
    -- when the cursor is on a misspelled word.                   --
    --                                                            --
    ----------------------------------------------------------------

    procedure Learn_Word (The_Word : String := ""; Dictionary : Natural := 0);
    --
    -- Insert the current word into a dictionary.

    procedure Exchange_Word (Choice : Positive := 1);
    --
    -- Replace the current word with the indicated alternative.
```





```
    procedure Learn_Replacement (The_Word   : String   := "";
                                 Choice     : Positive := 1;
                                 Dictionary : Natural  := 0);
    --
    procedure Learn_Replacement (The_Word    : String  := "";
                                 Replacement : String  := ">>Correction<<";
                                 Dictionary  : Natural := 0);
    --
    -- Make an automatic correction from the current word to the
    -- indicated alternative.

    procedure Explain_Next;
    --
    -- Simply a combination of Editor.Cursor.Next and Common.Explain.


    -----------------------------------------------------------------
    -- Speller OE Commands:                                        --
    --                                                             --
    -- These procedures are meant to configure the speller.        --
    -- They can be run from any window, but are easier to run from --
    -- the speller window.                                         --
    --                                                             --
    -----------------------------------------------------------------

    procedure Set_Switch (To_Value    : Boolean := True;
                          Switch_Name : String  := ">>Speller Switch<<");
    --
    -- Change the value of a speller switch.

    procedure Create (Dictionary_Name : String := ">>New name<<");
    --
    -- Create (and open for update) a new dictionary.

    procedure Open (Dictionary_Name : String  := "<CURSOR>";
                    Writable        : Boolean := False);
    --
    -- Open another auxiliary dictionary.

    procedure Save (Dictionary : Natural := 0);
    --
    -- Save any changes which have been made to a dictionary.

    procedure Close (Dictionary : Natural; Save_Changes : Boolean := True);
    --
    -- Close a dictionary.  All changes will be saved.
```

```
    function Number (Dictionary_Name : String := "<CURSOR>") return Natural;
    --
    -- Return the number of a named dictionary.

    procedure Read (File_Name  : String  := "<CURSOR>";
                    Dictionary : Natural := 0);
    --
    -- Insert a specially formatted text file into an open,
    -- writable dictionary.

    procedure Write (To_File    : String  := ">>New file name<<";
                     Dictionary : Natural := 0);
    --
    -- Dump the contents of a dictionary to a text file.

end Speller;
```





```
package Switches is

    -- This is the command-level interface to the Switch file facility

    subtype File_Name is String;

    -- An unambiguous Directory string name for a switch file or a
    -- Directory or World. In the latter case, the file asscoiated with
    -- that Directory or World is implied.

    Default_File : constant File_Name := "";

    -- The default file is the selected object if it is a switch file,
    -- otherwise it is the switch file associated with the current
    -- enclosing library.

    subtype Composite_Name is String;

    -- an expanded Ada name whose prefix is a processor and whose simple
    -- name is a switch of that processor. (If the switch name is unique,
    -- the processor name can be omitted.)

    -- "Semantics.Ignore_Minor_Errors", "Cg.Enable_Environment_Debugger"

    subtype Value_Image is String;

    -- Processor/Switch dependent. Will follow Ada conventions where
    -- possible. E.g. the value images of Boolean valued switches are "true"
    -- and "False"

    subtype Specification is String;

    -- A specification of the settings for selected switches in the form of
    -- a sequence of Ada assignment statements.  The lefthand side of the
    -- assignment is the name of the switch and the righthand side is the
    -- image of the value to be assigned to that switch.

    -- e.g.,
    -- "Ignore_Minor_Errors := true; Cg.Enable_Environment_debugger := false;"

    procedure Define (File     : File_Name := ">>SWITCH FILE<<";
                      Response : String    := "<PROFILE>");

    -- Creates an empty switch file with the given name. (File must not
    -- denote an existing object.)
```

```
procedure Associate (File     : File_Name := "<SELECTION>";
                     Library  : String    := "<IMAGE>";
                     Response : String    := "<PROFILE>");

-- The specified File is associated with the given Library.
-- Association is by-reference. Any subsequent changes to the specified
-- File will be reflected immediately in the associated library.

function Associated (Library : String := "<IMAGE>") return File_Name;

-- Returns the name of the switch file associated with the given Library.
-- Returns the null string if no switch file has been asociated.

procedure Set (Spec     : Specification := ">>SWITCHES<<";
               File     : File_Name     := "<SWITCH>";
               Response : String        := "<PROFILE>");

-- In the given switch file, the values of the switches named in the
-- specification are updated to the values in that spec.

procedure Display (Names    : Composite_Name := "@.@";
                   File     : File_Name      := "<SWITCH>";
                   Response : String         := "<PROFILE>");

-- The switches in the given file whose names match the wildcard Names
-- specification are listed to the current output file.

procedure Edit (File : File_Name := "<SWITCH>");

-- Brings up a new Switch Display Window containing the contents of the
-- specified file. This window becomes the current Switch Display Window

procedure Visit (File : File_Name := "<SWITCH>");

-- Changes the current Switch Display Window to display the contents of
-- the specified switch File.  The existing contents are committed
-- before the new file is displayed. A new Switch Display Window is
-- created if none have yet been created by the user.

procedure Insert (Spec : Specification := ">>SWITCHES<<");

-- The switch values displayed in the current Switch Display Window are
-- changed as indicated. (Generated in response to Object."I" on a
-- Switch Display Window)
```





```
procedure Change (Image : Value_Image := ">>SWITCH VALUE<<");

-- The highlighted switch in the current Switch Display Window is
-- changed to the value of the given image. (Generated in response to
-- Object."Z" on a Switch Display Window.)

procedure Write (File : File_Name := ">>SWITCH FILE<<");

-- The contents of the Current Switch Display Window are copied to the
-- specified switch file.

procedure Create (File     : File_Name := ">>SWITCH FILE<<";
                  Category : Character := 'L';
                  Response : String    := "<PROFILE>");

-- Creates an empty switch file of the specified Category with the
-- given name. File should not exist. If it exists and is a File
-- object, a new, empty version will be created of the indicated
-- category.

Of_Session : constant File_Name := "<SESSION>";

-- Switch File_Name used to denote the switches asociated with the
-- current session.

Of_Library : constant File_Name := "<SWITCH>";

-- Switch File_Name used to denote the switches asscoiated with the
-- enclosing library.

procedure Edit_Session_Attributes;

-- Equivalent to Edit (Switches.Of_Session);

procedure Dissociate (Library  : String := "<IMAGE>";
                      Response : String := "<PROFILE>");

-- Sever the association between the specified library and any switch
-- file.

end Switches;
```

```
package System_Backup is

    subtype Id   is Natural;
    type    Kind is (Full, Primary, Secondary);
    -- Full backup is self-sufficient
    -- Primary incremental is a differential from last Full backup
    -- Secondary incremental is a differential from last Primary

    procedure Backup (Variety : Kind := System_Backup.Full);
    -- Take a backup of kind Variety.

    procedure History (Entry_Count       : Positive := 10;
                       Full_Backups_Only : Boolean  := False;
                       Tape_Information  : Boolean  := False);

    -- print a list of Entry_Count previous backups.  Full_Backups_Only
    -- implies showing only Full backups.  Tape_Information implies a list
    -- of tapes involved in each.

    generic
        with procedure Backup_Starting  (Is_Full : Boolean);
        with procedure Backup_Finishing (Was_Successful : Boolean);
    procedure Backup_Generic (Variety : Kind; Wait_Until : String);

    -- Complete form of Backup.
    --
    -- Backup starts at the time specified in wait_until.  The tape is mounted
    -- now, then Backup pauses until the time specified.  Backup_Starting
    -- is then called, the backup happens, then Backup_Finishing is called.
    --
    -- The formal procedures are provided to allow setting up the machine for
    -- a backup at the last moment.  Usually used to alter scheduling
    -- parameters and the like.

end System_Backup;
```





```
procedure Accept_Tokens (Product         : String   := ">>PRODUCT NAME<<";
                         Donation        : Positive := 0;
                         Resulting_Count : Positive := 0;
                         Code            : String   := ">>AUTHORIZATION CODE<<";
                         Authorization   : String   := "";
                         Response        : String   := "<PROFILE>");

-- Accept tokens from a Donate_Tokens invocation or from purchase of new
-- tokens.
--
-- If Donate_Tokens is used, the Authorization field will be empty and
-- Accept_Tokens MUST BE RUN ON THE SAME DAY as Donate_Tokens for the same
-- product.  When an Authorization is used, it must match the one used to
-- generate the code.  Accept_Tokens may only be run once for any
-- particular Code.
--
-- Records the new number of authorized tokens in the machine error log and
-- makes the new tokens available immediately.
```

```
procedure Allow_Token_Conversion
            (Current_Product    : String       := ">>AUTHORIZED PRODUCT<<";
             New_Product        : String       := ">>REPLACEMENT PRODUCT<<";
             Token_Count        : Positive     := 0;
             Remote_Machine_Id  : Long_Integer := 0;
             Authorization      : String       := "";
             Date               : String       := "";
             Response           : String       := "<PROFILE>");

-- Produces the authorization code necessary to run Convert_Tokens.
--
-- This command may only be run from machines with Rational Site ID's, but
-- should not be exported to customer machines.  It also requires the
-- caller have "operator capability".
--
-- Authorization parameter, if non-default, specifies the factory
-- authorization code for for computing the code to be used by the
-- customer.  This authorization must, along with Code, be unique on the
-- customer machine and cannot be re-used.
--
-- Date parameter, if non-default, specifies the date on which the transfer
-- will take place.  This allows for planning and time-zone differences.
--
-- Date and Authorization are mutually exclusive.  If neither is specified,
-- it is as if Date were specified to be Today.
```





```
procedure Analyze_Disks_For_Backup (Report_Space_Address : Boolean := False;
                                    To_Console           : Boolean := False;
                                    To_System_Error_Log  : Boolean := False;
                                    Start_Volume         : Natural := 1);

-- This procedure should be run only by Rational personnel or under their
-- direction!

-- Performs an analysis of the disk structure(s)
--
-- If there is an inconsistency in the internal disk structure(s) Backup will
-- typically fail with an !Lrm.Assertion error and then log a message such as:
--
--      Backup Inconsistent_Space space = <258,Module,156694>
--
-- This procedure gathers information about the disk in the same fashion as
-- Backup.  Whenever something inconsistent is detected, a diagnostic message
-- is displayed.  Unlike Backup which would then immediately terminate, this
-- procedure will attempt to continue and should report all inconsistencies.
--
-- This procedure will take a significant amount of time; for large disks, it
-- will take 30 or more minutes for each disk.  The basic algorithm it uses is:
--
--      For each disk volume
--          Check all VPs
--          For Each Segment
--              Check each disk block associated with this segment
--              (Note: since a disk block can be shared among segments, disk
--                     blocks can be "visited" more than once.)
--          End loop
--      End loop
--
--
-- NOTES
--    1. This procedure must be scheduled to avoid overlap with the Disk client
--       and/or Backup.  By default, if disk collection starts during this
--       procedure, this procedure will be killed.  This default behavior can
--       be changed by executing !Tools.Disk_Daemon.Set_Backup_Killing(false)
--
--    2. The Diagnostic messages display all information in DECIMAL
--
-- It should be noted that a likely outcome when a corrupted system
-- is discovered by this procedure is a system crash.
```

```
procedure Change_Boot_Microcode (To_Version : String);

-- This procedure is used to change the name of the microcode file
-- specified in the boot configuration file.
--
-- Using this procedure allows the configation to be updated while the machine
-- is up.  The change will take affect on the next reboot.
--
-- The parameter TO_VERSION should be the simple name of the microcode file,
--
-- (ie. M207_41)

-- The procedure will verify that the name is legal, and then check that the
-- microcode specified exists in the DFS.  The procedure will not update the
-- configuration if the microcode does not in the DFS file system.

-- This procedure requires OPERATOR privileges be enabled.

-- Note: This procedure does not support series 100 systems.
```

```
procedure Check_And_Correct_Users (Do_Correction : Boolean := True;
                                   Trace         : Boolean := False);

-- Check each user to make sure its home directory exists and is properly
-- linked with the user.  If not, and Do_Correction is true, fix it so it is.
-- The proper linkage can be broken if the home world is deleted and
-- recreated but the user is not deleted and recreated.
```





```
procedure Check_Universe_Acls;
```

```
procedure Convert_Tokens (Current_Product : String := ">>AUTHORIZED PRODUCT<<";
                          New_Product     : String := ">>REPLACEMENT PRODUCT<<";
                          Token_Count     : Positive := 0;
                          Code            : String := ">>AUTHORIZATION CODE<<";
                          Authorization   : String := "";
                          Response        : String := "<PROFILE>");

-- Convert tokens among Login, Full Session, and Fundamental Session.
-- Converts all of the tokens of the Current product to the New one.
-- Normally used to upgrade from Fundamental to Full, but can also be used
-- to allow transfer of Logins to a 400 (by doing the transfer, then
-- converting them to the proper kind).
```

```
procedure Destroy_Library (Existing    : String  := ">>OBJECT To Destroy<<";
                           Effort_Only : Boolean := True;
                           Response    : String  := "<PROFILE>");

-- Destroy the named library(ies) and all of their contents.  The library
-- need not be empty.  This procedure will work hard to get rid of all
-- of the objects beneath the named libraryes.  Subsystems may be included.
--
-- It is a good idea to run with Effort_Only first to make sure you are not
-- accidently naming anything unexpected.  In general, the actions of
-- this commands are not reversable.
--
-- This is a good command to use to destroy large directory areas or to
-- delete home worlds of users that have been deleted.
```

```
procedure Destroy_User (Existing    : String := "!Users.[>>Users to destroy<<]";
                        Effort_Only : Boolean := True;
                        Response    : String := "<PROFILE>");

-- Destroy the named users.  The home world, group, searchlists, and error
-- logs for each user are destroyed.
--
-- It is a good idea to run with Effort_Only first to make sure you are not
-- accidently naming anyone unexpected.  In general, the actions of
-- this command is not reversable.
```





```
procedure Display_Switches (Pattern  : String    := "@.@";
                            Category : Character := 'A';
                            Hidden   : Boolean   := True;
                            Sort_Key : String    := "4");

-- This procedure is useful for determining the actual name of a hidden
-- switch or for determining whether an instantiation of the switch
-- definition is currently elaborated on a machine.

-- Lists the switches currently defined in the system, one line per
-- switch containing the following six information fields

-- Hidden Catagory Processor Name Type Default_Value

-- The parameters function as follows:

-- Pattern: Specification of switches to display.  Default displays all
--          switches.  "@" will display all switches with a unique
--          switch-name-only abbreviation.  "?" will display both the
--          full name and the abbreviation (if unique).  Any pattern
--          involving #, @, ?, and sets are acceptable.  Other examples
--          are:
--                   [semantics,r1000_cg].@
--                   @cg@.@

-- Category:
--   A     => All categories
--   L     => Library switches only
--   S     => Session switches only

-- Hidden:
--   True  => Hidden switches are included in the display
--   False => Only visible switches are included

-- Sort_Key: List in desired order, major to minor, column numbers on which
--           to sort the display.
--   "4"   => Sort by simple switch name
--   "34"  => Sort by processor and switch name within processor
--   "134" => Separate hidden/non-hidden; sort by processor and switch name
--   ...
```

```
procedure Donate_Tokens (Product                 : String       := ">>PRODUCT<<";
                         Donation                : Positive     := 0;
                         Resulting_Remote_Count  : Positive     := 0;
                         Remote_Machine_Id       : Long_Integer := 0;
                         Remote_Site             : String       := ">>SITE ID<<";
                         Response                : String       := "<PROFILE>");

-- Give up Donation authorized tokens to the machine with whose ID is
-- Remote_Machine_Id.  For verification, the number of authorized
-- tokens that will result from the corresponding Accept_Tokens on the
-- remote machine must be provided as Resulting_Remote_Count.
--
-- Transfers are only valid within a single Site, as defined by Rational
-- constracts.  The corresponding Accept_Tokens will fail if the Sites are
-- different.  It is not possible to either Donate or Accept Tokens from a
-- machine without a valid Site ID.  See Set_Site for details.
--
-- Operation fails if the number of active tokens on the local machine
-- exceeds the newly computed limit.
--
-- Prints the corresponding Accept_Tokens call and records the donation
-- and resulting authorization change in the machine error log.
--
-- Requires Operator privileges.
--
-- A machine may not both Donate and Accept Tokens in the same day.
```

```
procedure Find_Block (Volume_Number : Natural; Block_Number : Natural);

-- Locate the object containing the specified disk block
-- by scanning the entire directory system.  This is likely to
-- take a really long time and consume lots of machine resources.
-- If found, the string name of the object will be reported.
```





```
procedure Find_Null_Acls (Scope : String := "!??'V(all)");

-- Scan through all objects specified by Scope and report
-- any that have null access lists.
```

```
function Get_Machine_Id return Long_Integer;

-- Returns the Machine_ID for the current machine.
```

```
function Get_Site return String;

-- Returns the Site ID for the current machine.  "" if there is none.
```

```
procedure Image_Tree_Consistency (Unit     : String := "<IMAGE>";
                                  Response : String := "<PROFILE>");

-- Check that Ada units and their images are consistent.
-- Reports those that are correct and those that aren't.
-- Incorrect reports are non-specific; diagnosis requires Rational assistance.
-- Inconsistencies are rare; almost all that do occur can be fixed by
-- Library.Reformat_Image.
```

```
procedure Last_Gasp_Destroy (Name : String);

------------------------------------------------------------------------------
--  Name     - Name of some object, withing a directory, to be "deleted".
--
-- Forceably removes an object from a directory.  Use this as a last-resort
-- when you've tried everything else while trying to delete an object or a
-- set of objects.  This should normally only be used if you are consistently
-- getting VERSION_ERROR on a particular object.  This is probably the wrong
-- thing to do if you are getting "Key not found" errors.
--
--       **** DO NOT USE THIS CASUALLY. ****
--
-- This routine generates permanent garbage on the machine.  Use this only
-- as a last resort.  The object is removed from the directory and "dropped".
-- It is no longer findable or garbage collectable.
--
-- Before using this routine:
--
--  a) Have you tried Cmvc_Maintenance.Check_Consistency?
--
--  b) Have you tried Compilation.Set_Target_Key("*R1000", "$$.??'c(Library)");
--     (Note the '*', it turns off checking.)
--
--  c) Have you tried Repair_Directory on the containing World and all contained
--     Libraries in the world?
--
--  d) Have you tried Program_Library_Maintenance.Build?
--
--  e) Have you tried Cdb_Maintenance.Destroy?
--
--  f) Have you tried Repair_Cg_Attrs?
--
--  g) Have you tried Low_Level_Destroy?
--
-- If you use this routine:
--
--  a) Use the Uncode procedure first on all Coded units that must be
--     destroyed, so as to delete and eliminate CG attribute spaces if
--     at all possible.
--
--  b) Use the Uninstall procedure first, to delete semantic attribute spaces
--     if at all possible.
--
--  c) Use this routine.
--
--  d) Use Repair_Directory on the directory containing the destroyed objects.
```





```
--     It will print messages about cleaning up some things; this is expected.
--
--  e) Use Program_Library_Maintenance.Build to make sure that the program
--     library is functional.  "Verify" is not good enough; use "Build".
--
--  f) Use Cmvc_Maintenance.Check_Consistency if this is in a View.  Use it on
--     all views that import this view.
--
--  g) Take a full backup instead of an incremental next time around.
------------------------------------------------------------------------------

procedure Low_Level_Destroy (Objects  : String := "<SELECTION>";
                             Response : String := "<PROFILE>");

------------------------------------------------------------------------------
--  Objects     - Specifies what objects to destroy
--  Response    - Specifies the user profile to use
--
-- Do everything in our power to destroy a set of objects.  The Objects
-- specified, as well as any and all sub-objects are destroyed.
--
-- Note: CMVC databases are not maintained, updated, or consulted.
--       If any CMVC subsystems reference any of the destroyed objects, it
--       will be necessary to use Cmvc_Maintenance.Check_Consistency on them
--       afterwards.
--
-- To use this procedure:
--
--  a) First, use Repair_Directory on all directories in the Objects list and
--     on any immediately containing directory, eg. Objects & ".$".
--
--  b) Second, try a normal destroy, eg. Library.Delete on the objects.
--
--  c) Third, if this fails then use Low_Level_Destroy.
--
--  d) Fourth, if any of the Objects still exist, go back to step (a).
------------------------------------------------------------------------------
```

```
procedure Monitor_Performance (Max_Samples     : Natural := 500;
                               Sample_Interval : Duration := 60.0 * 12;
                               Output_File     : String := ">>filename<<";
                               Append_To_File  : Boolean := False;
                               Header          : String := "Performance data");

-- Monitor system performance at a global level.  Take Max_Samples, one
-- every Sample_Interval time, and write the output to Output_File.
-- If Append_To_File is true, append the data to the end of the file, else
-- overwrite it.  The Header string is output to the file before
-- anything else.
--
-- Output file contains one entry per line with each entry having the
-- following meaning:
--
--      Time        Time that the entry is written
--      Cpu         Cpu % used during the sample interval.  If the load is
--                  high, this number will be very inflated as it must
--                  be computed.  Values >100% are common under high loads,
--                  so this value must be considered approximate.
--      DiskW       Number of disk waits in this sample.  Roughly, the number
--                  of disk transfers which is a function of the number of
--                  page faults.
--      Users       Number of users logged on at the end of the interval.
--      Jobs        Number of jobs running at the end of the interval.
--      Jobs_T      Number of jobs that terminated during the interval.
--                  Indicates the level of command activity.
--      Load15      CPU run load for last 15 min.
--      Disk15      Disk load for last 15 min.
--      Whld15      Withheld task load for last 15 min.
--
```





```
procedure Refresh_Terminal_Information;

-- Refreshes systemwide terminal information from
--    !Machine.Editor_Data.[Terminal_Recognition,
--                          Terminal_Types,
--                          @_Keys,
--                          @_Commands]
-- Information from these objects is automatically read during system
-- boot, and it is therefore not necessary to call Refresh_Terminal_
-- Information from Machine.Initialize.  However, running this proce-
-- dure makes the system refresh its internal cache of the information
-- from these files without requiring the system to be rebooted.  This
-- should be done when any of these files are changed.  Users who login
-- subsequent to running this procedure will get the changed information.
```

```
procedure Release_Tape_User (Drive : Natural);

-- Resets the environment state of the drive to "Free" so that subsequent
-- mount request will not encounter "Drive is in use" error.  Also attempts
-- to rewind the tape.
--
-- This routine DOES NOT issue a hardware reset to the tape drive.
```

```
procedure Repair_Cg_Attrs (Verbose     : Boolean := False;
                           Effort_Only : Boolean := True);

-- Clean up persistent garbage generated by an RCG bug.
--
-- RCG (prior to D_12_4_8) had a bug whereby attributes associated
-- with Ada objects would accidently get created twice, with the first
-- collection of attributes becoming persistent garbage.  Machines have
-- been found with more than 200 megabytes of persistent (i.e., undeletable)
-- garbage.
--
-- This utility will locate orphan attributes (known as attrs), and
-- optionally remove them.  It can safely be run while others are using
-- the system, although it does require the caller to be a member of the
-- privileged group.
--
-- It typically takes 1 or 2 hours on an unloaded system to identify all the
-- ophan attrs, although on a heavily loaded system (or a system with many
-- Ada objects), it may take considerably longer.
--
-- The amount of disk space which can be reclaimed is summarized
-- by the utility; the reclamation will actually occur after the next
-- disk collection.
--
-- In VERBOSE mode, the locating of an attribute (good or bad) is
-- noted; this mode is not very meaningful for anyone other than a
-- low-level implementor.
--
-- In EFFORT_ONLY mode, bad attrs are not deleted.
--
-- Special notes:
--
--   1. A message of "KEY_NOT_FOUND" is not fatal, and merely an indication
--      that some other job has started/completed on the system.
--   2. A message of "NONEXISTENT_PAGE_ERROR" suggests the disk has
--      a corrupt object on it.  This may or may not be fatal, and has
--      no known fix other than restoring from a 'good' backup.
--
-- In theory, this routine need only be run once after moving from a
-- pre-D_12_4_8 release to a post-D_12_4_7 release.
--
```





```
procedure Repair_Directory (Directory : String := "<IMAGE>";
                            Response  : String := "<PROFILE>");

-- Previous releases of the directory system code contained bugs that
-- could corrupt the internal structures used to represent directories in
-- the system.  This procedure will traverse the representations of the
-- directories named by the Directory parameter, checking for any
-- anomalies.  If an anomaly is found, the procedure will attempt to
-- repair it.

-- The only anomalies detected by Repair_Directory are objects that are
-- no longer accessible by name or are not deletable by normal means.
-- The mode of repair is to destroy these objects and remove from the
-- internal representation any history of their existence.  Once the
-- repair is complete, the directory will be useable again as if nothing
-- ever happened; the objects that were damanged, however, will no longer
-- be accessible.

-- REMOVE VALUABLE DATA FROM THE DIRECTORY BEFORE RUNNING THIS COMMAND.
-- (You can put it back afterwards.)  While we believe that the analysis
-- performed by this procedure will not cause good data to be destroyed,
-- we cannot account for all possible ways a previous release (or this
-- one) may fail.

-- USE THIS PROCEDURE AS A LAST RESORT; IT CAN ONLY DESTROY DATA THAT
-- CANNOT BE DESTROYED BY OTHER MEANS.
```

```
procedure Save_Performance_Data
             (Reason   : String   := "";
              Filename : String   := "!Machine.Error_Logs.Performance_Data";
              Samples  : Natural  := 12;
              Interval : Duration := 20.0);

-- Append current job and scheduler information in the specified file.
-- If provided, the Reason string will also be included at the head of
-- the entry.  Multiple samples will be taken and logged at the
-- specified interval.  This can be used to capture scheduler
-- information at critical times during system operation for later
-- analysis.
```

```
procedure Setup_Machine (Response : String := "<PROFILE>");

-- For use by Rational Tech Reps only.  This procedure is used to convert
-- existing machines to Delta 2 per-session pricing, or to initialize new
-- machines on the manufacturing floor or on delivery.
--
-- Requests interactive input of the site name for the machine followed
-- by token limites for each of the products installed on the machine.
-- The site name and token counts are then set for the machine.
--
-- Once run, the machine cannot be converted back pre-Delta 2 pricing.
-- Equivalently, this command can be run only once.
--
-- Requires that the machine have no previous Site_ID
--
-- To change the Site Name of the machine, use the Set_Site command.
-- Accept_Tokens and Donate_Tokens can be used to alter the number of
-- tokens for each product individually.
```

```
procedure Set_Site (Site          : String := ">>SITE ID<<";
                    Code          : String := ">>AUTHORIZATION CODE<<";
                    Authorization : String := "";
                    Response      : String := "<PROFILE>");

-- Set the Site ID for this machine to Site using Code provided by
-- Rational.  This is used for changing the site of a machine should
-- it be transfered within or between support contracts.
--
-- This procedure is for Rational technical representatives use only.
```





```
procedure Set_Universe_Acls (Level               : Natural := 0; -- none
                             Implementation_Okay : Boolean := True;
                             Network_Read_Okay   : Boolean := True;
                             Network_Write_Okay  : Boolean := True;
                             Trace_Only          : Boolean := False;
                             Produce_Tables      : Boolean := False;
                             Tables_Output_File  : String  := "acl_tables");

-- Level = 0 => none   : anyone can do anything.
--       = 1 => Open   : anyone can do anything, but they may have to change
--                       acls to do it.
--       = 2 => Safe   : System and users are protected.  The operator must
--                       change acls to create new areas and allow others to
--                       things that users can do under level=1.
--       = 3 => Secure : Like safe, but more limited network access and less
--                       read access.

-- Set acls for the standard universe to be as described above.
-- Level 3 is about the most restrictive the system can be and still
-- run.  Level 3 will prevent most users other than Operator from
-- successfully executing operator commands even if they have operator
-- capability via write access to !Machine.Operator_Capability.

-- Implementation_Okay => access is given to !Implementation and
-- !Compiler_Interface.  Actually, !Compiler_Interface needs to be
-- readable anyway because it contains the switch file for the
-- standard universe.

-- Network_Read_Okay => Network_Public is granted read to most things, except
-- when Secure (level=3) is specified.

-- Network_Write_Okay is analogous to Network_Read but for Write access.

-- Be sure to update !machine.[user_acl_suffix,user_default_acl_siffix]
-- so that new users will get the acls you wish.
-- Don't forget about !machine.operator_capability, either.

-- Produce_Tables true causes previous parameters to be ignored, and a
-- file (specified by Tables_Output_File) to be written showing all
-- combinations of acls for documentation purposes.
```

```
procedure Show_Compiler_Version (Units    : String := "<SELECTION>";
                                 Response : String := "<PROFILE> USE_ERROR");

-- Displays the version of the compiler (code generator)
-- used to compile the specified units (which should be
-- in the coded state).
```

```
procedure Show_Configuration;

-- Display information about the R1000 configuration.  This includes
-- the machine id, release, whether the system is a coprocessor,
-- number of memory boards, whether there is an Exabyte cartridge
-- tape present, information about the last backup, and information
-- about each disk volume.
```

```
procedure Show_Directory_Information (Objects  : String  := "<IMAGE>";
                                      Levels   : Integer := -1;
                                      Versions : Boolean := True);

-- Display internal directory information associated with the
-- specified Objects.

-- Levels is the number of levels to descend in the traversal of objects.
-- -1 => no limit.  Note that this does not include crossing directory
-- boundaries, only subunit boundaries.
```





```
procedure Show_Elaborated_Configuration;

--
-- Display the configuration currently elaborated on this R1000.  The
-- configuration is the versions on the operating system subsystems that
-- make up an environment release.  This procedure displays the name of
-- the current configuration, then the subsystems and their versions
-- that the configuration contains, and finally the Microcode version
-- that is running in this configuration.
--
-- This information may be helpful to Rational personnel when they are
-- investigating a problem on the machine.  Running this command will
-- not cause any harm to the R1000, but the output will have very little
-- meaning to the end users.
```

```
procedure Show_Environment_Vpids;

-------------------------------------------------------------------------------
-- Generate a list of all EEDB configurations on a machine.  (All configurations
-- that reside in the DFS file system.)  For each configuration, lists the
-- subsystems (with name, time, and user).  For each subytem, lists all
-- segments (with VPID, kind, and segment number).
-------------------------------------------------------------------------------
```

```
procedure Show_Groups (Show_Next_Available_Id : Boolean := False);

-- Display the free group map and group name map.
-- If Show_Next_Available_Id is true, the next group id to be allocated
-- is displayed.  Getting this information CONSUMES THAT GROUP ID so
-- the display costs one id.  The group id is recovered as part of ACL
-- compaction.
```

```
with Machine;
with Default;

procedure Show_Identity (Job : Machine.Job_Id := Default.Process);

-- Display the access control identity of the specified job.
-- This is the list of groups that the job is a member of.
```

```
procedure Show_Image_Options (Unit : String := "<IMAGE>");

-- Display the pretty printer options in effect for one or more Ada units.
-- Pretty printer options are set for a unit when it is created, based on the
-- appropriate library switch file.  Thereafter, the settings are only changed
-- by running Library.Reformat_Image.  This procedure can be used to diagnose
-- cases where the current image and switch file options differ.
```

```
procedure Show_Iop_Kernel;
```





```
procedure Show_Jobs (Job              : Natural  := 0;
                     Active_Jobs_Only : Boolean  := True;
                     Repeat_Display   : Duration := 0.0);

-- Show information about jobs.  Includes lots of resource use information.
-- Active_Jobs_Only restricts the display to jobs that are active
-- or have otherwise consumed a lot of resources.  If false, all
-- jobs are displayed.

-- Use Show_Job_Names to get their names and additional information.

-- If Job /= 0, that specific job is displayed.  Otherwise all jobs are
-- displayed subject to the active filter.

-- If Repeat_Display is nonzero, then the display is repeated every
-- time that amount of time elapses.  The job will run until killed,
-- or 5000 displays have been output.

-- The values displayed are as follows:
--
-- Job    The job number as used in other commands.  This is the VP number.
-- Pri      Machine priority.  User jobs start at 6.  Larger => lower priority.
-- Stat     Status: R(unning), I(dle), W(ait), Q(ueued), AT(tached),
--                  DT (detached), CE (Core editor), OE (Obj editor).
-- CPU      % of CPU being used.
-- ModCt    Number of packages and tasks in the job (roughly).
-- Cache    Number of main memory pages occupied by the job (excluding heaps).
-- Disk     Number of disk pages occupied by the job (excluding heaps).
-- PgLim    Page limit.  The job page limit.
-- DskWts   Disk waits.  Roughly, the number of page faults taken by the job.
-- D/S      Disk waits/second.  Number of faults/second.
-- JSegSz   Number of pages in the job's job garbage heap.
-- WsSiz    Working set size as defined by the scheduler.
-- WsLim    Working set limit.
```

```
procedure Show_Job_Names (Job              : Natural := 0;
                          Active_Jobs_Only : Boolean := True);

-- Show job names and other name-related information.  If Active_Jobs_Only
-- is false show all jobs, else only those consuming some amount of
-- resources.  Job=0 => all jobs else the specified job.

-- Information shown is:

-- Job      The job number (as used in other commands).
-- CPU%     % of CPU being consumed by the job
-- Root     The root task id of the job.  If zero, it's probably a system
--          job.
-- Job Seg  The name of the job segment.  This is the segment name of the
--          job garbage heap.
-- Name     The string name of the job.  Job 4 is the "SYSTEM".
```





```
procedure Show_Load_Proc_Unit_Names (Load_Proc : String := "<SELECTION>";
                                     Response  : String := "<PROFILE>");

------------------------------------------------------------------------------
--  Load_Proc   - Specifies the 'c(Load_Proc) or 'c(Func_Proc) to query.
--  Response    - Specifies what messages to report.
--
-- Used to obtain the Ada object instance ID numbers for the Ada units that
-- make up the source code for this load-proc.  The printout includes lines
-- of this form:
--
--      Library.Resolve ("<[ADA,  12345,1]>");
--
-- These lines will sometimes be followed by an environment path name.
-- These lines indicate the Ada object instance ID of one or more of the
-- original Ada source units that make up this load-proc.  If those Ada source
-- units still exist, and if they exist on the current machine, then the
-- environment paths of those units will also be printed.  If the units do
-- not exist (perhaps they have been edited, moved, renamed, or deleted), or if
-- the load-proc came from another machine, then no path will be printed.
--
-- In order to attempt to locate the source units for this load-proc, take the
-- Library.Resolve lines that were printed and log into each R1000 where the
-- source units might still exist.  Execute the Library.Resolve commands.
-- With luck, eventually one of the source units will be found.
------------------------------------------------------------------------------
```

```
procedure Show_Locks (Show_Job_Action_Summary : Boolean := True;
                      Show_All_Actions        : Boolean := False;
                      Job_Id_Filter           : Natural := 0;
                      Action_Id_Filter        : Natural := 0;
                      Show_Raw_Info           : Boolean := False);

-- Display information about actions in progress.  The Job_Action_Summary
-- shows, for each active job, how many actions are in-progress.
-- The All_Actions display shows each in-progress action and the
-- objects it has locks on.

-- If Job_Id_Filter is nonzero, the displays are limited to only actions
-- and information for the specified job.

-- If the Action_Id_Filter is nonzero, the displays are limited to only
-- the specified action.

-- Show_Raw_Info causes display of the raw form of info from the Action
-- manager.
```

```
procedure Show_Log_Throttle_State;

-- Display the current state of the error log throttle.  Includes the
-- current condition and configuration parameters.  If the condition is
-- abnormal, individual client and status information is included.
```

```
procedure Show_Machine_Id (Boards : Boolean := False);

-- Display the machine's id number.
--
-- If Boards is true, also display maintenance information about each
-- PC board in the main R1000.
```





```
procedure Show_Manager_State (Sorted_By_Field : Integer := 1);

-- Print the current size, VPID, and volume of each of the managers.
-- Sorted_By_Field controls the order of printing:
--       1 => by Name
--       2 => by Size (decreasing)
--       3 => by VPID
--       4 => by Volume
```

```
procedure Show_Memory_Hogs (Vp             : Natural := 0;
                            Volume         : Natural := 1;
                            Size_Threshold : Natural := 250);

-- Show large segments/jobs
--
-- If vp is in the range 1..1023, then only that Vp is searched; Volume is
-- ignored.
--
-- If Vp is 0, then all Vps on the specified Volume are searched.
--
-- The search looks for segments bigger than Size_Threshold pages,
-- and reports the names of the objects as best as it can.
--
-- The output includes:
--
--  Pages       - the number of pages in the object.
--  Snap        - Snapshot number in which the object is committed.
--  Space_Mark  - The identifier that indicates the owner of the space.
--  Object_Name - The environment object name or object id.  May be the
--                image of an associated ada object: check the space_mark.
--  Space_Name  - The virtuam memory segment address.  Can be given to
--                the Kernel command Show_Space_Info to get more information.
```

```
procedure Show_Mts_Profiles (Job : Natural := 0);

-- Show profile information about jobs.  Reports number of times MTS
-- changed the state of a job, including run, wait, and idle states.
-- The wait state counts are further divided into CPU, disk, and memory
-- waits.
```

```
procedure Show_Mts_Stats;

-- Show job duration statistics for foreground jobs, aged foreground
-- jobs, and background jobs.
```





```
procedure Show_R1000_Configurations
             (Machine_List_File : String := "Machine_List";
              Owner_File        : String := "!Machine.Transfer.Global";
              Commands          : String :=
                 """!Commands.Abbreviations"".Show_Configuration;");

-- This command produces a report about a network of R1000s providing
-- information about each in table form.  It gathers all information
-- before producing any output.  It is somewhat experimental, but
-- may be useful to system managers and network administrators.
--
-- The Machine_List_File is a file containing the names of the machines
-- which are to appear in the report.  The names must appear one per
-- line.  If the name is preceeded with a "*", then the name appears in
-- the report, but no attempt is made to get information about it.  This
-- is useful if there is a machine that is known to be down.  Example:
-- (note: the leading "-- " are not part of the file:
--
-- capitol
-- clem
-- cookie
-- *crud
-- duke
--
-- The Owner_File is a mail distribution list which is searched to find
-- machine owner/managers.  For each machine M, the Owner_File is searched
-- for a line containing "M_mgr: " and the first token on that line is used
-- as the machine owner.
--
-- Commands is the command that is run on each machine to get its information.
-- It must be runnable under identity Network_Public.  The intended command
-- is Show_Configuration from this subsystem.  For maximum reliability,
-- it is recommended that a Load_Proc form of Show_Configuration be created
-- in !Commands.Abbreviations.  The default value for the Commands parameter
-- assumes this has been done.
--
-- Sample report output:
--
-- Machine Configuration List.  Generated May 6, 1990 at 10:51:40 PM
--
--   Name      Cluster Id    Release     Type  Exabyte   Owner     Last Backup
-- =========  ==========  ============  ====  =======  ========  =============
-- capitol    983758      D10_20_0_MAI  ?              ray       06-MAY-90 Full
-- clem       932367      D_11_4_0B     200   Yes      bill      05-MAY-90 Full
-- cookie     943752      D_11_4_1      200   Yes      sara      05-MAY-90 Full
-- crud                                                mike
-- duke       973551      D_10_20_2A_U  200            jenny     18-APR-90 Full
```

```
with Machine;
with Default;

procedure Show_Session_Of_Job (Job : Machine.Job_Id := Default.Process);

--
-- Display the Session Id of the specified job.  If no values is
-- specified for the "job" parameter, then the job that is calling this
-- procedure is used.  This procedure displays the job number and name,
-- then the user identity that the job is running under, then the base
-- session name that the job is running under, and finally the full
-- session name that the job is running under.
--
-- Sessions are the basic element of the Rational Environment that jobs
-- and tokens are associated with.  This procedure is useful in
-- determining who started a given job, or under what identity a given
-- job is executing.
```

```
procedure Show_Site;
```





```
procedure Show_Snapshot_Times (Log_File        : String  := "<IMAGE>";
                               Not_Before      : Integer := 7;
                               Not_After       : Integer := 19;
                               Report_Per_File : Boolean := False;
                               Response        : String  := "<PROFILE>");

-- Takes a log file as would appear in !Machine.Error_Logs and analyses the
-- time spent in snapshots.  Filters the snapshots to exclude those occurring
-- before Not_Before or after Not_After.  Not_Before => 7, Not_After => 19
-- would only process snapshots that occurred between 7:00 in the morning and
-- 7:00 in the evening.  Report_Per_File will cause a separate analysis of each
-- file.  This can be useful for comparing a series of files.
-- All times are reported in seconds.
-- Snapshots requiring more than 2 minutes are displayed as the time of the
-- snapshot along with the length.
-- RMS is Sqrt (Sum (Times**2)).
-- A sample output might be:
--
-- 03:11:11 90/01/10 => 131
-- Value          Attribute
-- =====  ==========================
--    42  Snapshots
--    41  Average length
--    49  RMS
--     1  Over 2 minutes
--
--     8  Snapshots lasting 0.. 14
--     8  Snapshots lasting 15.. 29
--     8  Snapshots lasting 30.. 44
--     9  Snapshots lasting 45.. 59
--     5  Snapshots lasting 60.. 74
--     2  Snapshots lasting 75.. 89
--     1  Snapshots lasting 90.. 104
--     1  Snapshots lasting 120.. 134
```

```
procedure Show_Stats;

-- Display job resource usage to current output.
--
-- Information includes:
--
--   Elapsed     Elapsed job time
--   Cpu         Cpu time used
--   Disk_Waits  Total number of disk waits
--   Job_Heap    Number of pages in job heap
```

```
procedure Show_Tape_Users;
```





```
procedure Show_Tasks (Task_Or_Job_Id              : Integer;
                      Show_Stack                  : Boolean := True;
                      Show_Full_Stack             : Boolean := False;
                      Show_Hex_Stack              : Boolean := False;
                      Show_Packages               : Boolean := True;
                      Name_Filter                 : String  := "";
                      Max_Tasks                   : Natural := 50;
                      Names_File                  : String  :=
                         "!Commands.System_Maintenance.@_Mload";
                      Debug_File                  : String  :=
                         "!Commands.System_Maintenance.@_Rdebug";
                      Structural_Expansion_Level : Natural := 4;
                      Pointer_Expansion_Level    : Natural := 3;
                      Max_Elements                : Natural := 20);

-- Scan the specified Task_Or_Job_Id.  If less than 256, it is
-- interpreted as a job id.  As many tasks as possible for the
-- job are located.

-- Show_Stack => stacks call trace is displayed

-- Show_Full_Stack => a full detailed display of the stack is provided,
--                    with each word shown and decoded.  The parameters
--                    Structural_Expansion_Level, Pointer_Expansion_Level,
--                    and Max_Elements control the amount of decoding word
--                    done.

-- Show_Hex_Stack => control stack words are displayed in hex.

-- Show_Packages => inactive package modules are listed.

-- Name_Filter is a string against which names of modules are matched.
-- If the Name_Filter is contained in the module name, it is listed, else
-- not.  The null string is in all names.

-- Max_Tasks = max number of modules to be displayed/traversed.  The search
-- stops once this count is reached.  Running time is partially proportional
-- to the square of this number.

-- In MV location names, the package is listed along with the subprogram
-- node number and statement number.  The node number is the diana node
-- id in the MV tree.

-- The Names_File is a pattern for file names.  These files are Mload
-- format files listing package names for each code segment.
```

```
procedure Show_Tokens (User_Filter        : String  := "";
                       Product_Filter     : String  := "";
                       Include_Sessions   : Boolean := True;
                       Include_Debugging  : Boolean := False;
                       Include_Statistics : Boolean := False);
```

```
procedure Smooth_Snapshots (Load      : Float   := 0.01;
                            Disk_Load : Float   := 0.1;
                            Trace     : Boolean := False);

-- Load is the target maximum load that this scan will generate.
-- Disk_Load is the target maximum disk load that this scan will generate.
-- Load is generated during the last part of the interval between snapshots.
-- Trace => True generates log messages (---) show what is being done.
```

```
procedure Token_Information (Product_Name :     String;
                             Limit        : out Natural;
                             Buy_Out      : out Natural;
                             Current      : out Natural);

-- Return the token limit, buy_out level and current use for the named
-- product.
```





```
procedure Training_Authorization
            (Code          : String  := ">>AUTHORIZATION CODE<<";
             Authorization : String  := "";
             Enable        : Boolean := True;
             Response      : String  := "<PROFILE>");

--
-- Suspend token and product checking during training.  Temporary until the
-- machine is re-booted.  Code is ignored if Enable is False.
--
```

```
procedure Uncode (Units                     : String  := "<SELECTION>";
                  Rebuild_Program_Libraries : Boolean := True;
                  Response                  : String  := "<PROFILE>");

-- Forces units to be demoted from the coded to installed state without
-- regard for any dependencies (Thus, it is possible to have coded units
-- that have suppliers that are not coded ... so this command should be
-- used with extreme caution).  An attempt is made to destroy all listing
-- files, code segments and attribute spaces; but failure to destroy any
-- of these associated objects will not cause the demotion to fail (any
-- undestroyed objects will become permanent garbage if nothing further
-- is done to get rid of them).  The program libraries for the enclosing
-- worlds are not updated as part of the demotion and so they must be
-- rebuilt in order to make the world consistent again; any load procs
-- in the enclosing world that are not Delta-1 compatibile must be reloaded
-- as a result of rebuilding the program library.  Units that are not in
-- the coded state are ignored.

-- This command is intended to be used to demote units when the compiler
-- cannot demote a unit due to a bug in the system.  Typical problems
-- include inability to destroy an associated object during demotion,
-- problems with the program library for the associated world and
-- unhandled exceptions in the compiler itself.  The compiler need not
-- be running in order to use this command to demote units.
--
-- The user must have privileged mode enabled to use this command.
```

```
procedure Uninstall (Units                    : String  := "<SELECTION>";
                     Rebuild_Program_Libraries : Boolean := True;
                     Response                 : String  := "<PROFILE>");

-- Forces units to be demoted to the source  state without regard for any
-- dependencies (Thus, it is possible to have installed units that have
-- suppliers that are not installed ... so this command should be used
-- with extreme caution).  An attempt is made to destroy all listing
-- files, code segments and attribute spaces; but failure to destroy
-- any of these associated objects will not cause the demotion to fail
-- (any undestroyed objects will become permanent garbage if nothing
-- further is done to get rid of them).  The program libraries for the
-- enclosing worlds are not updated as part of the demotion and so they
-- must be rebuilt in order to make the world consistent again; any load
-- procs in the enclosing world that are not Delta-1 compatible must be
-- reloaded as a result of rebuilding the program library.  Units that
-- are not in the installed or coded state are ignored.

-- This command is intended to be used to demote units when the compiler
-- cannot demote a unit due to a bug in the system.  Typical problems
-- include inability to destroy an associated object during demotion,
-- problems with the program library for the associated world and
-- unhandled exceptions in the compiler itself.  This command requires
-- the user have privileges enabled.  The compiler need not be running
-- in order to use this command to demote units.


procedure Verify_Backup (Wait_Until : String := "");

-- Verify that a tape backup is complete and recoverable.
-- This program mimics the backup recovery procedure, reading all
-- appropriate data, but does not actually restore the data.
--
-- Verify backup is appropriate for any R1000 backup (full, primary
-- or secondary) on either 9-track or 8mm tapes.  Like recovery, tape
-- mount requests are generate on the system console.
--
-- For a 9-track tape backup, tapes should be loaded in the following
-- order: Backup Index (Blue) tape first, then data tapes in sequence.
--
-- The 1st tape is mounted immediately; then the procedure pauses until
-- the time specified by Wait_Until.
```





```
package Tape is

    procedure Rewind (Drive : Natural := 0);
    procedure Unload (Drive : Natural := 0);

    procedure Read_Mt  (Drive : Natural := 0);
    procedure Write_Mt (File     : String  := "<SELECTION>";
                        Indirect : Boolean := True;
                        Drive    : Natural := 0);

    procedure Read (Volume      : String;
                    Directory   : String := "$";
                    Options     : String := "R1000 Add_New_Line";
                    To_Operator : String := "Thank You";
                    Response    : String := "<PROFILE>");

    -- The specified volume is mounted and all files are read into the
    -- given directory.

    -- Options are:

    --   FORMAT = R1000 | MV | VAX/VMS

    --   ADD_NEW_LINE
    --       Add a line terminator following each record read from tape.
    --       Without this option, bytes are copied from tape without
    --       interpretation or modification.

    -- Notes on mapping of tape names to R1000 file names
    --
    -- The file name from the tape is processed by replacing strings
    -- of non-alpha-numeric characters with a single '_'.  Then,
    -- if the name ends with an '_', the character 'B' is appended
    -- to the name.  If the name contains no alpha-numeric
    -- characters, a name derived from the user name and time is generated.

    procedure Write (Files       : String := "$@";
                     Volume      : String := "";
                     Options     : String := "R1000 Text_Files";
                     To_Operator : String := "Thank You";
                     Response    : String := "<PROFILE>");

    -- The specified Volume is mounted and the specified files are
    -- written to the volume.

    -- The To_Operator string is displayed to the operator when the
    -- request to mount the tape is made.
```

```
    -- Options are:

    --    Text_Files          If Text_Files is specified, the file is assumed
    --                        to contain only characters, line_terminators,
    --                        page_terminators, etc.  Each line of the file
    --                        is written to a record on the tape. Lines are
    --                        read according to the same rules as
    --                        Text_Io.Get_Line.

    --    Label               an optional part of the label written
    --                        to the volume header.
    --    Format              Target system (no abbreviations):
    --                           R1000, MV, or VAX/VMS
    --                           [Default: R1000]
    --    Record_Format       Ansi record format:
    --                           FIXED_LENGTH, VARIABLE_LENGTH or SPANNED
    --                           [Default: VARIABLE_LENGTH]
    --    Record_Length       A positive integer. [Default: 512]
    --    Block_Length        A positive integer. [Default: 2048]

    -- The file name that goes on the tape is generated as follows:
    --
    -- First, if the object is an Ada Unit, then "V_" or "B_" are prepended
    -- to the name if the unit is an Ada spec or body, respectively.
    -- Then, '_' characters in the name are removed.  One exception
    -- to this is that if the name ends in "_xyz", that underscore is
    -- replaced with '.', yielding a filename that will end in ".xyz".
    -- VAX/VMS bound file names are shortened to 9 characters;  others
    -- are shortened to 17 characters.  If, after removing '_' characters,
    -- the name is too long, vowels are removed starting at the right end
    -- of the name (excluding the suffix).  Then, if the name
    -- is still too long, it is truncated (again, excluding the ".xyz"
    -- suffix, if any).
    --
    -- Finally, to produce a unique name (with respect to others going on to
    -- the tape), 'A' characters are inserted in front of the suffix, if any,
    -- (preserving the ".xyz" suffix) and then these characters are
    -- incremented alphabetically until the name is unique.

    -- Thus, "An_Interesting_Name_Txt" becomes (if not VAX/VMS bound),
    --       "AnInterstngNm.Txt"

    Error : exception;

    procedure Examine_Labels (Vol_Id             : String  := "";
                              Vol_Set_Name       : String  := "";
                              To_Operator        : String  := "Thank you";
                              Volume_Labels_Only : Boolean := True);
```





```
    procedure Format_Tape (Drive : Natural := 0; Vol_Id : String := "");

    procedure Display_Tape (Drive             : Natural := 0;
                            Marks_To_Skip     : Integer := 0;
                            Records_To_Skip   : Integer := 0;
                            Blocks_To_Display : Natural := 10);

end Tape;
```

```
with System_Utilities;
with Telnet_Profile;

package Telnet is

    subtype User_Name is String;
    -- As used here, a User_Name is the name of a local user
    -- and session, joined by a '.', for example "CAROL.S_1".

    subtype Machine_Name is String;
    -- The name of a remote machine, to be used by Transport_Name_Map.

    subtype Session_Number is Positive;
    -- A single user may have several Telnet sessions with one remote
    -- machine: they are distinguished by different Session_Numbers.

    procedure Connect (Remote_Machine  : Machine_Name :=
                          Telnet_Profile.Remote_Machine;
                       Session         : Session_Number := 1;
                       Escape          : String := Telnet_Profile.Escape;
                       Escape_On_Break : Boolean :=
                          Telnet_Profile.Escape_On_Break;
                       Terminal        : System_Utilities.Port :=
                          System_Utilities.Terminal);

    -- Start or resume a session with the specified Remote_Machine
    -- and Session number.  If such a session already exists, it is
    -- resumed, if not, a new session is started.

    -- If Escape is non-null, and the Escape string is received
    -- from the terminal, then the session will be suspended
    -- and the terminal will be reconnected to the Environment.
    -- If Escape_On_Break is true, then a BREAK signal from the
    -- terminal will likewise escape from the session.

    -- Terminal specifies the local terminal from which you want
    -- to interact with the Remote_Machine.  The default is the
    -- same terminal you're currently logged in on.

    function My_User_Name return User_Name;

    procedure Disconnect (Remote_Machine : Machine_Name :=
                             Telnet_Profile.Remote_Machine;
                          Session        : Session_Number := 1;
                          User           : User_Name := Telnet.My_User_Name);
```





```
    -- Disconnect the Telnet session with the specified Remote_Machine
    -- and Session number which was started by the specified User.
    -- If no such session exists, do nothing.

    procedure Show_Sessions (User : User_Name := Telnet.My_User_Name);

    -- Show a table of existing sessions for the specified User.
    -- If User => "?", show existing sessions for all users.

    procedure Send (Data           : String         := Telnet_Profile.Escape;
                    Remote_Machine : Machine_Name   :=
                       Telnet_Profile.Remote_Machine;
                    Session        : Session_Number := 1);

    -- If a session to the specified Remote_Machine and Session
    -- exists, send the specified Data on it.  To the remote
    -- machine, it looks as though the data came from the terminal.
    -- If no such session exists, do nothing.

    procedure Send_Break (Remote_Machine : Machine_Name   :=
                             Telnet_Profile.Remote_Machine;
                          Session        : Session_Number := 1);

    -- If a session to the specified Remote_Machine and Session
    -- exists, send a break signal on it.  To the remote machine,
    -- it looks as though the break signal came from the terminal.
    -- If no such session exists, do nothing.

end Telnet;
```

```
with Default;
with Machine;
with System;
with System_Utilities;

package Terminal is

    subtype Port is Natural range 0 .. 4 * 16 * 16;

    -- valid terminal types
    -- Rational, VT100, Facit

    -- valid terminal rates
    --   DISABLE,      50,      75,     110,
    --   134_5,       150,     200,     300,
    --   600,        1200,    1800,    2400,
    --   4800,       9600,   19200,    EXT_REC_CLK

    subtype Stop_Bits_Range      is System_Utilities.Stop_Bits_Range;
    subtype Character_Bits_Range is System_Utilities.Character_Bits_Range;
    subtype Parity_Kind          is System_Utilities.Parity_Kind;
    -- None, Even, Odd

    function Current (S : Machine.Session_Id := Default.Session) return Port
        renames System_Utilities.Terminal;

    procedure Settings (Line : Port := Terminal.Current);
    -- print summary of current terminal

    procedure Set_Terminal_Type
                (Line  : Port   := Terminal.Current;
                 To_Be : String := System_Utilities.Terminal_Type);

    procedure Set_Input_Rate (Line  : Port   := Terminal.Current;
                              To_Be : String := System_Utilities.Input_Rate);

    procedure Set_Output_Rate (Line  : Port   := Terminal.Current;
                               To_Be : String := System_Utilities.Output_Rate);

    procedure Set_Parity (Line  : Port        := Terminal.Current;
                          To_Be : Parity_Kind := System_Utilities.Parity);

    procedure Set_Stop_Bits (Line  : Port            := Terminal.Current;
                             To_Be : Stop_Bits_Range :=
                                System_Utilities.Stop_Bits);
```





```
    procedure Set_Character_Size (Line  : Port := Terminal.Current;
                                  To_Be : Character_Bits_Range :=
                                     System_Utilities.Character_Size);

    procedure Set_Xon_Xoff_Characters
                (Line     : Port   := Terminal.Current;
                 Xon_Xoff : String := System_Utilities.Xon_Xoff_Characters);
    -- takes a 2-element string consisting of Xon followed by Xoff

    procedure Set_Xon_Xoff_Bytes (Line     : Port := Terminal.Current;
                                  Xon_Xoff : System.Byte_String :=
                                     System_Utilities.Xon_Xoff_Bytes);

    procedure Set_Flow_Control
                (Line  : Port   := Terminal.Current;
                 To_Be : String := System_Utilities.Flow_Control);

    procedure Set_Receive_Xon_Xoff_Characters
                (Line     : Port   := Terminal.Current;
                 Xon_Xoff : String := System_Utilities.
                                         Receive_Xon_Xoff_Characters);

    procedure Set_Receive_Xon_Xoff_Bytes
                (Line     : Port               := Terminal.Current;
                 Xon_Xoff : System.Byte_String :=
                    System_Utilities.Receive_Xon_Xoff_Bytes);

    procedure Set_Receive_Flow_Control
                (Line  : Port   := Terminal.Current;
                 To_Be : String := System_Utilities.Receive_Flow_Control);

    procedure Set_Disconnect_On_Disconnect
                (Line    : Port    := Terminal.Current;
                 Enabled : Boolean := System_Utilities.
                                         Disconnect_On_Disconnect);

    procedure Set_Logoff_On_Disconnect
                (Line    : Port    := Terminal.Current;
                 Enabled : Boolean := System_Utilities.Logoff_On_Disconnect);

    procedure Set_Disconnect_On_Logoff
                (Line    : Port    := Terminal.Current;
                 Enabled : Boolean := System_Utilities.Disconnect_On_Logoff);

    procedure Set_Disconnect_On_Failed_Login
                (Line    : Port    := Terminal.Current;
                 Enabled : Boolean := System_Utilities.
                                         Disconnect_On_Failed_Login);
```

```
    procedure Set_Log_Failed_Logins
                 (Line    : Port    := Terminal.Current;
                  Enabled : Boolean := System_Utilities.Log_Failed_Logins);

    procedure Set_Login_Disabled
                 (Line     : Port    := Terminal.Current;
                  Disabled : Boolean := System_Utilities.Login_Disabled);

    procedure Set_Detach_On_Disconnect
                 (Line    : Port    := Terminal.Current;
                  Enabled : Boolean := System_Utilities.Detach_On_Disconnect);

end Terminal;
```





```
package Text is

    type Image_Kind is (File, Input_Output);


    procedure Create (Image_Name : String     := ">>IMAGE_NAME<<";
                      Kind       : Image_Kind := Text.File);

    -- Create a text image.
    -- Image_Kind = File a text file with the given name
    -- Image_Kind = Input_Output creates an input_output image of that name
    -- Commands run from an input_output image will have that image as the
    -- default destination for Current_Output


    procedure Block (All_Windows : Boolean := False);


    procedure Continue (Page_Mode   : Boolean := False;
                        All_Windows : Boolean := False);


    procedure End_Of_Input;


    procedure Redirect (To : String := ">>File Name<<");

    -- Redirect the output associated with the current output
    -- window to the named file.

end Text;
```

```
with Directory;

package Transfer is

    type Name_Map_Kind is (Personal, Local, Global);

    procedure Check_Name_Map (File     : Directory.Naming.Name := "<CURSOR>";
                              Kind     : Name_Map_Kind := Transfer.Personal;
                              Response : String := "<PROFILE>");
    -- Check the File to see if it would be a valid name map file
    -- of the specified Kind.  Log errors to current_output.

    procedure Load_Name_Map (Response : String := "<PROFILE>");
    -- Read alias definitions from !Machine.Transfer.[Global,Local].
    -- Log errors to current_output.  It's not neccessary to call
    -- this procedure after changing a Personal name map.

    procedure Load_Carrier_Map (Response : String := "<PROFILE>");
    -- Read host-to-carrier assignments from !Machine.Transfer.Carrier_Map.
    -- Log errors to current_output.

    procedure Load_Time_Zone (Response : String := "<PROFILE>");
    -- Read the local time zone from the text file !Machine.Time_Zone.
    -- The first line of the file is the value that, when added to
    -- local standard time, gives the equivalent Universal Time (UT).
    -- This value is expressed in HHMM notation, and may be followed by
    -- comments, introduced Ada-style by "--".  The second line contains
    -- option values; the Boolean option Daylight_Savings, if true,
    -- indicates that the machine clock has been set forward one hour
    -- from standard time.  For example, in the North American Pacific
    -- time zone during the summer, the Time_Zone file would contain:
    --    +0800 -- Pacific Time
    --    Daylight_Savings => True
    -- During the winter in the same time zone, it would contain:
    --    +0800 -- Pacific Time
    --    Daylight_Savings => False

    procedure Show (Options : String := ""; Response : String := "<PROFILE>");
    procedure Display (Options : String := ""; Response : String := "<PROFILE>")
        renames Show;
    -- Write the current status of the Transfer system to current_output.
    -- Display all available status by default, or specific components of
    -- the status, as specified by the Options.
```





```
    procedure Set (Options : String := ""; Response : String := "<PROFILE>");
    -- Set the curent values of the specified Options:
    --     Started : Boolean; -- if True, start the carrier(s) running
    --     Tracing : Boolean; -- write detailed information to the log
    --     SMTP  : Boolean; -- Set option values for SMTP
    --     Local : Boolean; -- Set option values for Local delivery
    -- For example, Set ("SMTP, start") starts the SMTP carrier running.

end Transfer;
```

```
with Transport_Defs;

package Transport_Route is

    -- The system maintains a table used for routing Transport
    -- packets.  At this time the routing table is used only
    -- for IP packets, including TCP/IP and UDP/IP packets.
    -- When sending a packet to a machine on some other network
    -- (e.g. Ether), it must be routed first to a gateway, not the
    -- destination machine.  Some gateways do not respond to ARP
    -- queries for destination machines whose traffic they carry,
    -- so this machine must know the address of the gateway in
    -- order to transmit packets to it.

    -- The routing table contains a list of entries.  Each entry
    -- contains a route (i.e. the Internet address of a gateway)
    -- with a destination that can be reached by way of it.  The
    -- destination may be a specific Host_Id (Internet address), or
    -- the network number of a network (signifying all hosts in that
    -- network), or the Null_Host_ID (signifying any remote host).
    -- There may be multiple entries for each route, identifying
    -- multiple hosts or networks accessible by way of that route.
    -- The table entries are searched in order when deciding where
    -- to send an outgoing packet.  The table is kept ordered with
    -- all host-specific entries followed by all network-specific
    -- entries, followed by the default entry.  Within each group,
    -- entries are maintained in order they were defined.

    procedure Show (Route       : String                        := "";
                    Destination : String                        := "";
                    Network     : Transport_Defs.Network_Name := "";
                    Response    : String                        := "<PROFILE>");
    -- Create a text listing of entries matching the given values,
    -- and write it to the current output file.  "" is a wildcard
    -- for each parameter.  That is, if Route = "", show entries
    -- for all routes; if Destination = "", show entries for all
    -- destinations; and if Network = "", show entries for all
    -- networks.  Route and/or Destination may be a Host_Id in
    -- decimal dotted notation (e.g. "89.32") or a machine name
    -- (e.g. "Fred").  In the latter case, the name is resolved
    -- to a Host_Id using Transport_Name.Host_to_Host_Id.

    procedure Load (Table    : String := "!machine.transport_routes";
                    Form     : String := "";
                    Response : String := "<PROFILE>");
    -- Read the object named by Table, using package Text_Io.
    -- Pass the specified Form to Text_Io.Open.  In the resulting text,
```





```
    -- each text line should contain the Host_ID or name of a route,
    -- followed by the Host_ID or name of a destination, followed by
    -- a Network_Name.  If the Network_Name is omitted, "IP" is assumed.
    -- If the destination Host_ID is omitted, the Null_Host_ID is assumed.
    -- For each line in the Table, call the Define procedure (below)
    -- with parameter values parsed from the line.  The overall
    -- effect is to copy the information from the object named by
    -- Table into the system's routing table.

    procedure Define (Route       : String;
                      Destination : String                        := "";
                      Network     : Transport_Defs.Network_Name := "IP";
                      Response    : String                        := "<PROFILE>");
    -- Add one entry to the routing table, with the given values.
    -- If there is already such an entry in the table, do nothing.
    -- Route and/or Destination can be a Host_Id, in decimal dotted
    -- notation (e.g. "89.32"), or else a host name (e.g. "Fred").
    -- If Route or Destination is a symbolic name, resolve it to
    -- a Host_ID using Transport_Name.Host_to_Host_Id.  If Route
    -- or Destination = "", this means Transport_Defs.Null_Host_Id.

    procedure Undefine (Route       : String;
                        Destination : String := "";
                        Network     : Transport_Defs.Network_Name := "IP";
                        Response    : String := "<PROFILE>");
    -- Delete the entry with the given values from the routing table.
    -- If there is no such entry in the table, do nothing.
    -- Route and/or Destination can be a Host_Id, in decimal dotted
    -- notation (e.g. "89.32"), or else a host name (e.g. "Fred").
    -- If Route or Destination is a symbolic name, resolve it to
    -- a Host_ID using Transport_Name.Host_to_Host_Id.  If Route
    -- or Destination = "", this means Transport_Defs.Null_Host_Id.

end Transport_Route;
```

```
package What is

    procedure Does     (Name : String := "");
    procedure Command  (Clue : String := "");
    procedure Line;
    procedure Tabs;
    procedure Message  (File : String := "Daily_Message");
    procedure Time;
    procedure Load     (Verbose : Boolean := True);
    procedure Version;
    procedure Users    (All_Users : Boolean := True);
    procedure Jobs     (Interval          : Positive := 10;
                        User_Jobs_Only    : Boolean  := False;
                        My_Jobs_Only      : Boolean  := False;
                        Running_Jobs_Only : Boolean  := True);
    procedure Home_Library;
    procedure Object   (Name : String := "<IMAGE>");
    procedure Locks    (Name : String := "<IMAGE>");

    procedure Search_List_Resolution (Name : String := "<CURSOR>");
    -- Determines the object the given name will resolve to using the
    -- current search list.  The resolution and the search list entry that
    -- provided the resolution are displayed in the message window.

end What;
```





```
package Work_Order is

    procedure Set_Default_Venture (To_Venture : String := "<CURSOR>";
                                   For_User   : String := "<CURRENT_USER>";
                                   Response   : String := "<PROFILE>");

    function Default_Venture (For_User       : String  := "<CURRENT_USER>";
                              Ignore_Garbage : Boolean := True) return String;

    procedure Set_Notes_Venture (To_Value     : String := ">>New Notes<<";
                                 Venture_Name : String := "<VENTURE>";
                                 Response     : String := "<PROFILE>");
    --
    -- The "" Venture_Name is interpreted as "<CURSOR>".
    -- "<VENTURE>" are interpreted as the default venture.

    function Notes_Venture (Venture_Name : String := "<VENTURE>") return String;
    --
    -- The "" Venture_Name is interpreted as "<CURSOR>".

    procedure Display_Venture (Venture_Name : String := "<VENTURE>";
                               Options      : String := "";
                               Response     : String := "<PROFILE>");
    --
    -- Display the object by formatting and printing it.  The "" argument
    -- is interpreted as "<CURSOR>"
    -- Valid Options are all of the session switches, plus "<DEFAULT>"
    -- (which is the current session switch values), "<TERSE>" (the default),
    -- and "<VERBOSE>".

    procedure Edit_Venture (Venture_Name : String := "<VENTURE>");
    --
    -- Invoke the appropriate object_editor.  The "" Argument is
    -- interpreted as "<CURSOR>"

    procedure Create_Venture (Venture_Name : String := ">>OBJECT NAME<<";
                              Notes : String := "";
                              Make_Default_Venture : Boolean := True;
                              Response : String := "<PROFILE>");
    --
    -- Intended to be called from the command line
```

```
type Venture_Policy_Switch is (Require_Current_Work_Order,
                               Require_Comment_At_Check_In,
                               Require_Comment_Lines,
                               Journal_Comment_Lines,
                               Allow_Edit_Of_Work_Orders);

procedure Set_Venture_Policy (The_Switch   : Venture_Policy_Switch;
                              To_Value     : Boolean;
                              Venture_Name : String  := "<VENTURE>";
                              Effort_Only  : Boolean := False;
                              Response     : String  := "<PROFILE>");
--
-- Change a venture's policy switches.
-- The "" Venture_Name argument is interpreted as "<CURSOR>".

procedure Set_Default (To_Work_Order : String := "<CURSOR>";
                       For_Venture   : String := "<VENTURE>";
                       For_User      : String := "<CURRENT_USER>";
                       Response      : String := "<PROFILE>");

function Default (For_Venture    : String  := "<VENTURE>";
                  For_User       : String  := "<CURRENT_USER>";
                  Ignore_Garbage : Boolean := True) return String;

procedure Set_Notes (To_Value   : String := ">>New Notes<<";
                     Order_Name : String := "<ORDER>";
                     Response   : String := "<PROFILE>");
--
-- The "" Order_Name argument is interpreted as "<CURSOR>".

function Notes (Order_Name : String := "<ORDER>") return String;
--
-- The "" Order_Name argument is interpreted as "<CURSOR>".

procedure Close (Order_Name : String := "<ORDER>";
                 Response   : String := "<PROFILE>");
--
-- The "" Order_Name argument is interpreted as "<CURSOR>".

procedure Display (Order_Name : String := "<ORDER>";
                   Options    : String := "";
                   Response   : String := "<PROFILE>");
```





```
--
-- Display the object by formatting and printing it.  The "" argument
-- is interpreted as "<CURSOR>".
-- Valid Options are all of the session switches, plus "<DEFAULT>"
-- (which is the current session switch values), "<TERSE>" (the default),
-- and "<VERBOSE>".

procedure Edit (Order_Name : String := "<ORDER>");
--
-- Invoke the appropriate object_editor.  The "" Argument is
-- interpreted as "<CURSOR>"

procedure Create (Order_Name              : String  := ">>OBJECT NAME<<";
                  Notes                   : String  := "";
                  On_List                 : String  := "<WORK_LIST>";
                  On_Venture              : String  := "<VENTURE>";
                  Make_Default_Work_Order : Boolean := True;
                  Response                : String  := "<PROFILE>");
--
-- Command line interface
-- "" for list is interpreted as Nil (Added to no list)

procedure Create_Field (Field_Name : String := ">>FIELD NAME<<";
                        Field_Type : String := ">>BOOLEAN|STRING|INTEGER<<";
                        Is_Vector : Boolean := False;
                        Is_Controlled : Boolean := False;
                        Default : String := "";
                        Display_Position : Natural := Natural'Last;
                        On_Venture : String := "<VENTURE>";
                        Propagate : Boolean := True;
                        Renumber_Fields : Boolean := True;
                        Response : String := "<PROFILE>");
--
-- Create a new user-defined field in a Venture.
-- Field_Name is the name given to the field.
-- Field_Type can be "Boolean", "String", or "Integer".
-- If Is_Vector is true, the field is declared equivalent to
-- Field_Name : array (Positive) of Field_Type.
-- If Is_Controlled is true, whether or not the field is modifiable
-- using the object editor is controlled by a policy switch.
-- Display_Position specifies the relative position of this field
-- in the object editor display as compared to all of the other
-- user defined fields.  0 means don't display.
-- If Renumber_Fields is True, the display position is treated as
-- an ordinal number, i.e. a value of N will cause fields to be
-- renumbered so that the new one is the Nth in the sort order.
```

```
-- Default is the image of the default value (all elements of
-- a vector have the same default).  If no default is supplied,
-- False, "", or 0 will be assumed.
-- If Propagate is True, all existing work orders will be updated.


procedure Delete_Field (Field_Name           : String  := ">>FIELD NAME<<";
                        Venture_Name         : String  := "<VENTURE>";
                        Even_If_Data_Present : Boolean := False;
                        Response             : String  := "<PROFILE>");
--
-- Delete the named field from the venture.
-- If work orders exist that have data in the field, the
-- operation fails unless Even_If_Data_Present is true.


procedure Add_To_List (Order_Names : String := "<IMAGE>";
                       List_Name   : String := "<WORK_LIST>";
                       Response    : String := "<PROFILE>");


procedure Remove_From_List (Order_Names : String := "<IMAGE>";
                            List_Name   : String := "<WORK_LIST>";
                            Response    : String := "<PROFILE>");


procedure Set_Default_List (To_List     : String := "<CURSOR>";
                            For_Venture : String := "<VENTURE>";
                            For_User    : String := "<CURRENT_USER>";
                            Response    : String := "<PROFILE>");

function Default_List (For_Venture    : String  := "<VENTURE>";
                       For_User       : String  := "<CURRENT_USER>";
                       Ignore_Garbage : Boolean := True) return String;


procedure Set_Notes_List (To_Value  : String := ">>New Notes<<";
                          List_Name : String := "<WORK_LIST>";
                          Response  : String := "<PROFILE>");
--
-- The "" List_Name argument is interpreted as "<CURSOR>".


function Notes_List (List_Name : String := "<WORK_LIST>") return String;
--
-- The "" List_Name argument is interpreted as "<CURSOR>".
```





```
procedure Display_List (List_Name : String := "<WORK_LIST>";
                        Options   : String := "";
                        Response  : String := "<PROFILE>");
--
-- Display the object by formatting and printing it.  The "" argument
-- is interpreted as "<CURSOR>".
-- "<WORK_LIST>" is the default list for the current user.
-- Valid Options are all of the session switches, plus "<DEFAULT>"
-- (which is the current session switch values), "<TERSE>" (the default),
-- and "<VERBOSE>".


procedure Edit_List (List_Name : String := "<WORK_LIST>");
--
-- Invoke the appropriate object_editor.  The "" Argument is interpreted
-- as "<CURSOR>"


procedure Create_List (List_Name         : String  := ">>OBJECT NAME<<";
                       Notes             : String  := "";
                       On_Venture        : String  := "<VENTURE>";
                       Make_Default_List : Boolean := True;
                       Response          : String  := "<PROFILE>");


package Venture_Editor is

    procedure Set_Notes (Notes : String := ">>New Notes<<");
    procedure Set_Policy (To_Value   : Boolean := False;
                          The_Switch : Venture_Policy_Switch);
    procedure Spread_Fields (Interval : Natural := 10);
    procedure Set_Field_Info (Is_Controlled : Boolean := False;
                              Display_Position : Natural := 1;
                              The_Field : String := ">>Field Name<<");
    procedure Set_Default_Order (New_Default : String := "<SELECTION>";
                                 For_User    : String := "<CURRENT_USER>");
    procedure Set_Default_List (New_Default : String := "<SELECTION>";
                                For_User    : String := "<CURRENT_USER>");
    --
    -- Command line procedures for modifying a Venture.

end Venture_Editor;


package Editor is

    procedure Set_Notes (Notes : String := ">>New Notes<<");
    --
    -- A command line procedure to change the Notes.
```

```
        procedure Set_Field (To_Value  : String  := ">>Field Value<<";
                             The_Index : Natural := 0;
                             The_Field : String  := ">>Field Name<<");
        procedure Set_Field (To_Value  : Integer := 0;
                             The_Index : Natural := 0;
                             The_Field : String  := ">>Field Name<<");
        procedure Set_Field (To_Value  : Boolean := False;
                             The_Index : Natural := 0;
                             The_Field : String  := ">>Field Name<<");
        --
        -- A command line procedure for changing a field in a Work_Order.
        -- The_Index is ignored for scalar fields.

        procedure Add_User (The_User : String := "<CURRENT_USER>");
        procedure Add_Version (The_Configuration : String :=
                                  ">>Configuration Name<<";
                               The_Element       : String := ">>Element Name<<";
                               The_Generation    : Natural := 0);
        procedure Add_Configuration
                     (The_Configuration : String := ">>Configuration Name<<");
        procedure Add_Comment (The_Comment : String := ">>Comment<<";
                               The_Element : String := ">>Element Name<<";
                               The_User    : String := "<CURRENT_USER>");
        --
        -- Command line procedures for augmenting a Work_Order.

    end Editor;


    package List_Editor is

        procedure Set_Notes (Notes : String := ">>New Notes<<");
        --
        -- A command line procedure to change the Notes.

        procedure Add (Work_Orders : String := ">>Work Order Names<<");
        --
        -- A command line procedure for adding to a Work_Order_List.

    end List_Editor;

end Work_Order;
```

## !IO

This tabbed section contains copies of the specifications for the Ada packages defined in the world !Io. With the exception of minor formatting differences, these specifications are exact copies of those online.

The specifications in this section are organized alphabetically, as you would find them listed online. Units within a package are grouped together under the name of their enclosing package.

The standard contents of world !Io are listed below.

To find other locations where a package is documented, see the Map of the Rational Environment Library System (behind the Environment Specifications tab) or see the Master Index.

```
!Io : Library (World):
  Device_Independent_Io              : Ada (Pack_Spec);
  Direct_Io                          : Ada (Gen_Pack);
  Io                                 : Ada (Pack_Spec);
  Io_Exceptions                      : Ada (Pack_Spec);
  Object_Set                         : Ada (Pack_Spec);
  Pipe                               : Ada (Pack_Spec);
  Polymorphic_Io                     : Ada (Pack_Spec);
  Polymorphic_Sequential_Io          : Ada (Pack_Spec);
  Sequential_Io                      : Ada (Gen_Pack);
  Tape_Specific                      : Ada (Pack_Spec);
  Terminal_Specific                  : Ada (Pack_Spec);
  Text_Io                            : Ada (Pack_Spec);
  Window_Io                          : Ada (Pack_Spec);
```





```
with System;
with Action;
with Directory;
with Io_Exceptions;

package Device_Independent_Io is

    -- Device_Independent_IO is designed to provide a uniform method of
    -- accessing sequential devices, including files, terminals, windows,
    -- printers, and tape drives.  Its clients are expected to be Text_IO and
    -- Sequential_IO, though there may be others.

    -- The assumption is made that devices deal in bytes or characters rather
    -- than elemental types.

    type File_Type is private;
    type File_Mode is (In_File, Out_File);

    subtype Class       is Directory.Class;
    subtype Subclass    is Directory.Subclass;
    subtype Version     is Directory.Version;
    subtype Byte        is System.Byte;
    subtype Byte_String is System.Byte_String;

    procedure Open (File       : in out File_Type;
                    Mode       :        File_Mode;
                    Name       :        String;
                    Form       :        String          := "";
                    With_Class :        Directory.Class := Directory.Nil;
                    Action_Id  :        Action.Id       := Action.Null_Id);

    procedure Open (File       : in out File_Type;
                    Mode       :        File_Mode;
                    Object     :        Version;
                    Form       :        String          := "";
                    With_Class :        Directory.Class := Directory.Nil;
                    Action_Id  :        Action.Id       := Action.Null_Id);

    procedure Append (File       : in out File_Type;
                      Name       :        String;
                      Form       :        String          := "";
                      With_Class :        Directory.Class := Directory.Nil;
                      Action_Id  :        Action.Id       := Action.Null_Id);
```

```
procedure Append (File        : in out File_Type;
                  Object      :        Version;
                  Form        :        String          := "";
                  With_Class  :        Directory.Class := Directory.Nil;
                  Action_Id   :        Action.Id       := Action.Null_Id);
-- open the object for output and position at end of file

procedure Create (File          : in out File_Type;
                  Mode          :        File_Mode := Out_File;
                  Name          :        String    := "";
                  Form          :        String    := "";
                  With_Class    :        Class     := Directory.Nil;
                  With_Subclass :        Subclass  := Directory.Nil;
                  Action_Id     :        Action.Id := Action.Null_Id);
-- creates the named object, if it currently does not exist
-- declaration of a new version is dependent on the class of the object
-- if object does not exists, and class is nil, file is assumed.

procedure Close  (File : in out File_Type);
procedure Delete (File : in out File_Type);
procedure Reset  (File : in out File_Type; Mode : File_Mode);
procedure Reset  (File : in out File_Type);
procedure Save   (File : File_Type; Immediate_Effect : Boolean := True);
-- Save the current contents of the file, but leave it open
-- Immediate_Effect => don't wait until the action is committed

function Mode         (File : File_Type) return File_Mode;
function Name         (File : File_Type) return String;
function Form         (File : File_Type) return String;
function Get_Class    (File : File_Type) return Class;
function Get_Subclass (File : File_Type) return Subclass;
function Get_Version  (File : File_Type) return Version;
function Get_Action   (File : File_Type) return Action.Id;

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

procedure Read (File  :     File_Type;
                Item  : out Byte_String;
                Count : out Natural);
procedure Read (File : File_Type; Item : out Byte);
procedure Read (File : File_Type; Item : out String; Count : out Natural);
procedure Read (File : File_Type; Item : out Character);

procedure Write (File : File_Type; Item : Byte_String);
procedure Write (File : File_Type; Item : Byte);
procedure Write (File : File_Type; Item : String);
procedure Write (File : File_Type; Item : Character);
```





```
function Is_Interactive (File : File_Type) return Boolean;
-- The user-visible function that determines whether or not a file
-- is interactive.

function Is_Empty (File : File_Type) return Boolean;
-- Determine if the file has any contents


generic
    type Derived_File_Type is limited private;
    -- Only works for types derived from Device_Independent_IO.File_Type
    pragma Must_Be_Constrained (Derived_File_Type);
package File_Type_Conversions is
    function From_Standard (File : File_Type) return Derived_File_Type;
    function To_Standard   (File : Derived_File_Type) return File_Type;
end File_Type_Conversions;

-- Ability to convert between Device_Independent_IO File_Type and those
-- used by its clients.
-- Provided principally to allow setting of options for device_specific
-- packages that may be used in conjunction with Text_IO, etc.
-- Specific clients may buffer data in ways not visible to
-- Device_Independent_IO, so this is a generally dangerous operation for
-- Input/Output operations.


generic
    type Element_Type is private;
package Type_Specific_Operations is
    function  Read  (File : File_Type) return Element_Type;
    procedure Read  (File : File_Type; Item : out Element_Type);
    procedure Write (File : File_Type; Item : Element_Type);
end Type_Specific_Operations;
-- Type_Specific_Operations make it possible to implement the equivalent
-- of Sequential_IO or Polymorphic_Sequential_IO.


-- Exceptions

Status_Error : exception renames Io_Exceptions.Status_Error;
Mode_Error   : exception renames Io_Exceptions.Mode_Error;
Name_Error   : exception renames Io_Exceptions.Name_Error;
Use_Error    : exception renames Io_Exceptions.Use_Error;
Device_Error : exception renames Io_Exceptions.Device_Error;
End_Error    : exception renames Io_Exceptions.End_Error;
Data_Error   : exception renames Io_Exceptions.Data_Error;

end Device_Independent_Io;
```

```
with Io_Exceptions;
with Device_Independent_Io;

generic
    type Element_Type is private;
    pragma Must_Be_Constrained (Element_Type);

package Direct_Io is

    type File_Type is limited private;

    type File_Mode is (In_File, Inout_File, Out_File);

    type Count is new Integer range 0 .. Integer'Last / Element_Type'Size;

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


    -- File management

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

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

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

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

    function Is_Open (File : File_Type) return Boolean;


    -- Input and output operations

    procedure Read (File :     File_Type;
                    Item : out Element_Type;
                    From :     Positive_Count);
```





```
    procedure Read (File : File_Type; Item : out Element_Type);

    procedure Write (File : File_Type;
                     Item : Element_Type;
                     To   : Positive_Count);
    procedure Write (File : File_Type; Item : Element_Type);

    procedure Set_Index (File : File_Type; To : Positive_Count);

    function Index (File : File_Type) return Positive_Count;
    function Size  (File : File_Type) return Count;

    function End_Of_File (File : File_Type) return Boolean;


    -- Exceptions

    Status_Error : exception renames Io_Exceptions.Status_Error;
    Mode_Error   : exception renames Io_Exceptions.Mode_Error;
    Name_Error   : exception renames Io_Exceptions.Name_Error;
    Use_Error    : exception renames Io_Exceptions.Use_Error;
    Device_Error : exception renames Io_Exceptions.Device_Error;
    End_Error    : exception renames Io_Exceptions.End_Error;
    Data_Error   : exception renames Io_Exceptions.Data_Error;

private
    -- implementation dependent
    type File_Descriptor;
    type File_Type is access File_Descriptor;
    pragma Segmented_Heap (File_Type);

end Direct_Io;
```

```
with Action;
with Device_Independent_Io;
with Directory;
with Io_Exceptions;
with Text_Io;

package Io is

    type File_Type is private;

    subtype File_Mode is Text_Io.File_Mode;
    In_File  : constant File_Mode := Text_Io.In_File;
    Out_File : constant File_Mode := Text_Io.Out_File;

    subtype Count          is Text_Io.Count;
    subtype Positive_Count is Count range 1 .. Count'Last;
    Unbounded : constant Count := Text_Io.Unbounded;

    subtype Field       is Integer range 0 .. Integer'Last;
    subtype Number_Base is Integer range 2 .. 16;

    subtype Type_Set is Text_Io.Type_Set;
    Upper_Case : constant Type_Set := Text_Io.Upper_Case;
    Lower_Case : constant Type_Set := Text_Io.Lower_Case;

    -- File Management

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

    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 := "");
    -- Open a particular directory version; not Text_IO

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





```
    procedure Append (File   : in out File_Type;
                      Object :        Directory.Version;
                      Form   :        String := "");
    -- Open existing file for output, positioned at end of file; not Text_IO
    -- Output starting after an Append is on a new line, but on the same page
    -- as the previous end of the file

    procedure Flush (File : File_Type);
    -- Force any buffer characters out to file

    procedure Save (File : File_Type);
    -- Save the current contents of the file, but leave it open; calls Flush

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

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

    function Is_Open (File : File_Type) return Boolean;


    -- Control of default input, output and error files;  error not Text_IO

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

    -- Equivalent of an Open/Create followed by above; not in Text_IO
    procedure Set_Input  (Name : String := "<SELECTION>");
    procedure Set_Output (Name : String := ">>FILE NAME<<");
    procedure Set_Error  (Name : String := ">>FILE NAME<<");

    function Standard_Input  return File_Type;
    function Standard_Output return File_Type;
    function Standard_Error  return File_Type;
    function Standard_Error  return Text_Io.File_Type;

    function Current_Input  return File_Type;
    function Current_Output return File_Type;
    function Current_Error  return File_Type;
    function Current_Error  return Text_Io.File_Type;

    -- For each default files, f, Set_f pushes that File_Type entry on a stack
    -- for the job.  Pop_f removes the top of the stack.  Reset is equivalent
```

```
-- to a Close and a Pop.
--
-- All open files in the default file stack at job termination are closed.
procedure Reset_Error;
procedure Reset_Input;
procedure Reset_Output;

procedure Pop_Error;
procedure Pop_Input;
procedure Pop_Output;


-- Specification of line and page lengths

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

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

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

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


-- Column, Line and Page Control

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

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

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

procedure New_Page (File : File_Type);
procedure New_Page;

procedure Skip_Page (File : File_Type);
procedure Skip_Page;

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

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





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

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

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

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

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


-- Character Input-Output

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

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


-- String Input-Output

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

procedure Put  (File : File_Type; Item : String);
procedure Put  (Item : String);
procedure Echo (Item : String := "");

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

procedure Put_Line  (File : File_Type; Item : String);
procedure Put_Line  (Item : String);
procedure Echo_Line (Item : String := "");


-- String Input-Output not in Text_IO

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

```
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);
-- Get all or part of a line.
-- End_Of_Line iff Item contains the end of line (possibly null)
-- End_Of_Page iff End_Of_Line and this is the last line of the page
-- End_Of_File iff End_Of_Page and this is the last page of the file
--
-- The intent if for each call to return as many characters from the
-- line as fit, but Last /= Item'Last doesn't implies End_Of_Line.


-- equivalents for instantiations of the type-specific generics

-- Integer Input-Output; equivalent to an instantiation of Integer_IO

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

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

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);

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


-- Float Input-Output; equivalent to an instantiation of Float_IO

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

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);

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


-- Boolean Input-Output; equivalent to an instantiation of Enumeration_IO

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

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

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

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


-- Generic package for Input-Output of Integer Types

generic
    type Num is range <>;
package Integer_Io is

    Default_Width : Field       := Num'Width;
    Default_Base  : Number_Base := 10;

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

    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);

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

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

```
-- Generic package for Input-Output of Floating Point Types

generic
    type Num is digits <>;
package Float_Io is

    Default_Fore : Field := 2;
    Default_Aft  : Field := Num'Digits - 1;
    Default_Exp  : Field := 3;


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

    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);

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

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

-- Generic package for Input-Output of Fixed Point Types

generic
    type Num is delta <>;
package Fixed_Io is

    Default_Fore : Field := Num'Fore;
    Default_Aft  : Field := Num'Aft;
    Default_Exp  : Field := 0;

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

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





```
    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);

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

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

-- Generic package for Input-Output of Enumeration Types

generic
    type Enum is (<>);
package Enumeration_Io is

    Default_Width   : Field    := 0;
    Default_Setting : Type_Set := Upper_Case;

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

    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);

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

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

```
    -- Interchange with other system file_types
    --
    -- Compatibility with Device_Independendent_IO is solely for the purpose
    -- of allowing access to device-specific options at open.
    --
    -- Interchange of Get/Put and Read/Write operations between IO and
    -- Device_Independent_IO is undefined due to internal buffering in IO,
    -- though Flush can be used on output to clear the buffer.
    --
    -- Free interchange of operations with Text_IO is supported.

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

    function Convert (File : File_Type) return Device_Independent_Io.File_Type;
    function Convert (File : Device_Independent_Io.File_Type) return File_Type;

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

    -- Operate on multiple input files matching a wildcard


    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);
    -- Calls Process once with an open File corresponding to each of Names
    -- Name errors and unhandled expections are reported through Note_Error.
    -- Closes File after each call to Process, if not already closed.


    procedure Convert (From :        Device_Independent_Io.File_Type;
                       To   : in out Text_Io.File_Type);
    -- Conversion form that allows changing limited private file_type


    -- File management logical overloads.  Each procedure duplicates one
    -- above, but with direct control over the Action.ID used.

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





```
    procedure Open (File      : in out File_Type;
                    Mode      :        File_Mode := Out_File;
                    Name      :        String;
                    Form      :        String    := "";
                    Action_Id :        Action.Id);

    procedure Open (File      : in out File_Type;
                    Mode      :        File_Mode;
                    Object    :        Directory.Version;
                    Form      :        String := "";
                    Action_Id :        Action.Id);

    procedure Append (File      : in out File_Type;
                      Name      :        String;
                      Form      :        String := "";
                      Action_Id :        Action.Id);

    procedure Append (File      : in out File_Type;
                      Object    :        Directory.Version;
                      Form      :        String := "";
                      Action_Id :        Action.Id);

    function Get_Action (File : File_Type) return Action.Id;

    -- Exceptions

    Status_Error : exception renames Io_Exceptions.Status_Error;
    Mode_Error   : exception renames Io_Exceptions.Mode_Error;
    Name_Error   : exception renames Io_Exceptions.Name_Error;
    Use_Error    : exception renames Io_Exceptions.Use_Error;
    Device_Error : exception renames Io_Exceptions.Device_Error;
    End_Error    : exception renames Io_Exceptions.End_Error;
    Data_Error   : exception renames Io_Exceptions.Data_Error;
    Layout_Error : exception renames Io_Exceptions.Layout_Error;

end Io;
```

```
package Io_Exceptions is

    Status_Error : exception;
    Mode_Error   : exception;
    Name_Error   : exception;
    Use_Error    : exception;
    Device_Error : exception;
    End_Error    : exception;
    Data_Error   : exception;
    Layout_Error : exception;

    pragma Exception_Name (Status_Error, 271);
    pragma Exception_Name (Status_Error, 256);  -- 256..271

    pragma Exception_Name (Mode_Error, 287);
    pragma Exception_Name (Mode_Error, 272);  -- 272..287

    pragma Exception_Name (Name_Error, 303);
    pragma Exception_Name (Name_Error, 288);  -- 288..303

    pragma Exception_Name (Use_Error, 319);
    pragma Exception_Name (Use_Error, 304);  -- 304..319

    pragma Exception_Name (Device_Error, 335);
    pragma Exception_Name (Device_Error, 320);  -- 320..335

    pragma Exception_Name (End_Error, 351);
    pragma Exception_Name (End_Error, 336);  -- 336..351

    pragma Exception_Name (Data_Error, 367);
    pragma Exception_Name (Data_Error, 352);  -- 352..367

    pragma Exception_Name (Layout_Error, 383);
    pragma Exception_Name (Layout_Error, 368);  -- 368..383

end Io_Exceptions;
```





```
with Action;
with Directory;
with Polymorphic_Io;

package Object_Set is

    function Is_Set (Object : Directory.Object) return Boolean;

    type Set is private;

    procedure Create (Set_Name  :     String;
                      Set_Id    : out Directory.Object;
                      Status    : out Directory.Error_Status;
                      Action_Id :     Action.Id := Action.Null_Id);

    procedure Open (Set_Id         :     Directory.Object;
                    The_Set        : out Set;
                    Status         : out Directory.Error_Status;
                    Action_Id      :     Action.Id := Action.Null_Id;
                    For_Update     :     Boolean   := False;
                    Prevent_Create :     Boolean   := False);

    procedure Close (The_Set : Set; Status : out Directory.Error_Status);

    function  Is_Empty   (The_Set : Set) return Boolean;
    procedure Make_Empty (The_Set : in out Set);

    -- The Set must be open for update.  (all sets are initially empty).

    function Is_Member (The_Set : Set; Id : Directory.Object) return Boolean;

    procedure Add (The_Set : in out Set; Id : Directory.Object);

    procedure Remove (The_Set : in out Set; Id : Directory.Object);

    type Iterator is limited private;

    procedure Init  (Iter : out Iterator; The_Set : Set);
    procedure Next  (Iter : in out Iterator);
    function  Value (Iter : Iterator) return Directory.Object;
    function  Done  (Iter : Iterator) return Boolean;

    function Handle_Of (H : Set) return Polymorphic_Io.Handle;

    -- returns the Polymorphic Io handle on the open set.

end Object_Set;
```

```
with Action;
with Directory;
with Io_Exceptions;
with System;

package Pipe is

    subtype Action_Id is Action.Id;
    Null_Action_Id : constant Action_Id := Action.Null_Id;
    subtype Byte        is System.Byte;
    subtype Byte_String is System.Byte_String;
    subtype Object_Id   is Directory.Version;

    subtype Operate_Status is Integer;

    Status_Error : exception renames Io_Exceptions.Status_Error;
    Mode_Error   : exception renames Io_Exceptions.Mode_Error;
    Name_Error   : exception renames Io_Exceptions.Name_Error;
    Use_Error    : exception renames Io_Exceptions.Use_Error;
    Device_Error : exception renames Io_Exceptions.Device_Error;
    End_Error    : exception renames Io_Exceptions.End_Error;
    Data_Error   : exception renames Io_Exceptions.Data_Error;
    Layout_Error : exception renames Io_Exceptions.Layout_Error;

    -- Exceptions are raised iff the the comments following the operation
    -- indicate that the exception is possible.  All other exception
    -- propagation is considered a bug in the underlying implementation.

    -- A pipe is an object which contains a queue of messages, possibly empty.
    -- By opening the object for write, one can do "Write" operations which
    -- append messages to the end of the queue.  By opening the object for
    -- read, one can do "Read" operations which consume messages from the
    -- beginning of the queue.  Each message is read by exactly one Read
    -- operation; thus, in the face of concurrent Reads, each client may see
    -- just a subset of the messages that were written to the pipe.

    -- It is ok to make concurrent calls to this package.  BUT, this does NOT
    -- include calls which supply the same Handle; this is considered
    -- erroneous.  The implementation does not prevent such erroneous behavior;
    -- this behavior might cause your Handle to be left inconsistent, but the
    -- internal representation of the pipe itself is protected and will remain
    -- consistent; thus, other Handle's (including those in other jobs) should
    -- still work properly.

    function Pipe_Class return Directory.Class;

    type Handle is private;
```





```
    Null_Handle : constant Handle;

    -- Contains control information which is pertinent to a particular "open"
    -- of a particular pipe.  Other control information (about pipes) is kept
    -- internally.  Logically, a Handle is limited private.  Use of multiple
    -- copies is considered erroneous.  The implementation does not prevent
    -- such erroneous behavior.  At worst, an erroneous program will be able
    -- to Read/Write a pipe which is still open elsewhere, even though other
    -- copies of the Handle have been closed; the internal representation of
    -- the pipe itself is protected from such erroneous behavior and will
    -- remain consistent.

    function Max_Buffer_Size return Positive;

    -- Measured in bytes.  Currently about half the maximum size of a heap.

    function Default_Buffer_Size return Positive;

    -- Measured in bytes.  Currently about 20K bytes.

    function Message_Overhead return Natural;

    -- Measured in bytes.  Currently about 8 bytes.  Clients can compute the
    -- value of B = n * (c + M), where "c" is the result of this function, "M"
    -- is the fixed size of messages supplied to the Write operation, "n" is
    -- the desired capacity of the buffer (in messages), and "B" is the
    -- resulting requirement for buffer size, in bytes.  This function may
    -- change between releases of the system.

    type Pipe_Open_Mode is (Exclusive_Read, Shared_Read,
                            Exclusive_Write, Shared_Write, Exclusive);

    -- The read modes allow one to use Read operations to consume messages from
    -- the beginning of the queue.  The write modes allow one to use Write
    -- operations to append messages to the end of the queue.  The same client
    -- can use both Read and Write by opening the pipe multiple times.  The
    -- compatibility matrix is as follows:

    -- Other action compatibility matrix:
    --                   Current Mode
    --                  (other actions)
    --                  ER  SR  EW  SW  E
    --                 ---------------------
    --              ER |         X   X
    --                 |
    --              SR |     X   X   X
    --                 |
    --    Desired   EW | X   X
    --     Mode        |
```

```
--             SW | X    X        X
--                |
--             E  |
--                ---------------------
-- Absence of an "X" indicates that the desired access will not be granted
-- if any OTHER action (not including requesting action) has the indicated
-- current access.  Via Max_Wait, queueing is available when access is
-- denied for this reason.

-- Assuming access is not denied by the above rules, the following matrix
-- indicates the upgrade compatibility rules:

-- Upgrade matrix:
--                     Current Access
--                    (by same action)
--                   ER  SR  EW  SW  E
--                ---------------------
--             ER | ER  ER           E
--                |
--             SR | ER  SR           E
--    Desired     |
--    Access   EW |         EW  EW   E
--                |
--             SW |         EW  SW   E
--                |
--             E  |  E   E   E   E   E
--                ---------------------
-- Absence of a mode indicates that the desired access will not be granted
-- if the requesting action already has the indicated current access.
-- Queueing is not available in this case.  Presence of a mode indicates
-- that the access will be granted, and indicates the new lock mode in
-- which the action holds object.

-- Note that the upgrade rules imply that a single action cannot be used to
-- both read and write the same pipe.  Of course, a single task can read
-- and write the same pipe by using 2 actions.

-- RESTRICTION: In Delta, the Create operation only supports Exclusive,
-- and the Open operation only supports Shared_Read, Shared_Write, and
-- Exclusive.

procedure Create (Pipe                : in out Handle;
                  Mode                : Pipe_Open_Mode;
                  Name                : String;
                  Action              : Action_Id := Null_Action_Id;
                  Max_Wait            : Duration := Directory.Default_Wait;
                  Permanent_Contents  : Boolean := False;
                  Buffer_Size         : Positive := Default_Buffer_Size;
                  Reader_Buffer_Size  : Natural := 0);
```





```
-- Since it's an object, a pipe lives in the directory system, as specified
-- by the Name parameter.  Naming of pipes is the same as for vanilla
-- files.  Multiple versions of a pipe are allowed.

-- With respect to disk space, the system reserves the right to allocate
-- disk space for the entire buffer, at any time, including the first
-- open. Thus, one should not use excessively large buffer sizes. With
-- respect to working set, under certain circumstances the buffer is used
-- in a cylic fashion.  Thus, a large buffer size may cause a large
-- working set.  All in all, its a good idea to use reasonable buffer
-- sizes.  One rule of thumb is to use a buffer of size 2 * N * E, where
-- "N" is the number of servers (readers), and "E" is the expected message
-- size; this tends to leave just enough room for writers to be "double
-- bufferred".  (Of course, the buffer must be large enough to hold the
-- biggest message.)

-- Given variable length messages, it is not possible for clients to
-- accurately predict the number of messages that can be stored by a buffer
-- of a particular size.

-- A pipe is made empty when it is last closed, or first opened if the
-- system crashes while the pipe is open.  The only difference from the
-- user's point of view is disk space consumption.

-- The create operation leaves the pipe "open" in the specified mode.

-- Rules for Action are the same as for Open.

-- Abandoning the action may cause the object to dissappear from the
-- directory system.

-- Abandoning/Committing the action causes all open handles (using the
-- action) to become closed.

-- If Null_Action_Id is supplied, a new action is created.  If the Create
-- is successful, the new id is stored in the Handle, and committed when
-- the Handle is closed.  If the Create fails, the new action is
-- abandoned.

-- Reader_Buffer_Size controls the operation of the Read function.  If 0,
-- then it defaults to the result of calling Max_Buffer_Size.

-- KNOWN BUGS IN DELTA: (1) Abandoning the action which created the pipe
-- does NOT cause all open handles to become closed immediately.  (2) The
-- implementation has a window in which concurrent opens may acquire the
-- object; this will cause the Create to return any of the exceptions that
-- can be returned by Open.
```

```
-- EXCEPTIONS:
--     Status_Error:  The given Handle is already open
--     Mode_Error:    Mode must be Exclusive
--     Name_Error:    Directory wont create the object
--     Use_Error:     Illegal buffer size, or lock error, or
--                      access control violation
--     Device_Error:  Obj Mgr can't set/get the buffer size;
--                    and other internal errors

procedure Open (Pipe               : in out Handle;
                Mode               : Pipe_Open_Mode;
                Name               : String;
                Reader_Buffer_Size : Natural := 0;
                Action             : Action_Id := Null_Action_Id;
                Max_Wait           : Duration := Directory.Default_Wait);

-- Open an already existing pipe.

-- If the action is abandoned, the Handle may become implicitly closed.

-- Committing the action has no effect on the state of the pipe buffer.

-- If Null_Action_Id is supplied, a new action is created, its id stored
-- in the Handle, and the action is committed when the Handle is closed.

-- Reader_Buffer_Size controls the operation of the Read function.  If 0,
-- then it defaults to the result of calling Max_Buffer_Size.

-- KNOWN BUG IN DELTA: Exclusive access shows up in the action_manager's
-- lock information as "Update"; all other access modes show up as "Read".

-- EXCEPTIONS:
--     Status_error : The given Handle is already open.
--     Mode_Error:    Mode must be Shared_Read, Shared_Write, or Exclusive
--     Name_Error:    Directory can't find the object
--     Use_Error:     Lock error, or access control violation
--     Device_Error:  Obj mgr can't get the buffer size;
--                    and other internal errors

procedure Open (Pipe               : in out Handle;
                Mode               : Pipe_Open_Mode;
                Object             : Object_Id;
                Reader_Buffer_Size : Natural := 0;
                Action             : Action_Id := Null_Action_Id;
                Max_Wait           : Duration := Directory.Default_Wait);

-- Same as above, but assumes that the caller has already resolved the
-- string name into an object id.
```





```
procedure Close (Pipe     : in out Handle;
                 Max_Wait :        Duration := Directory.Default_Wait);

-- If the pipe is open for writing, causes an implicit call to
-- Write_End_Of_File (throwing away a Use_Error caused by Max_Wait
-- induced timeout);

-- If the corresponding Create/Open supplied Null_Action_Id, then the
-- implicit action is either committed (when the Close is successful) or
-- abandoned (when the Close is unsuccessful).
-- The handle becomes closed.

-- EXCEPTIONS:
--     Status_error:  The given Handle is not open.
--     Device_Error:  internal errors

procedure Delete (Pipe     : in out Handle;
                  Max_Wait :        Duration := Directory.Default_Wait);

-- Like all objects, causes it to be deleted.  Must have the object open
-- for Exclusive access.  Assuming a reasonable value for retention count,
-- the object can be "undeleted" using other environment operations.

-- If the corresponding Create/Open supplied Null_Action_Id, then the
-- implicit action is either committed (when the Delete is successful) or
-- abandoned (when the Delete is unsuccessful).
-- The handle becomes closed.

-- EXCEPTIONS:
--     Status_Error:  The given Handle is not open
--     Name_Error:    Directory returned an error other than Lock_Error or
--                    access control error
--     Use_Error:     Directory returned Lock_Error, which probably means
--                    that Handle was not open for Exclusive access;
--                    or could be an access control violation
--     Device_Error:  internal errors

Dont_Wait : constant Duration := 0.0;
Forever   : constant Duration := Duration'Last;

procedure Write (Pipe     : in out Handle;
                 Message  :        Byte_String;
                 Max_Wait :        Duration := Forever);

procedure Read (Pipe     : in out Handle;
                Message  : out    Byte_String;
                Length   : out    Integer;
                Max_Wait :        Duration := Forever);
```

```
function Read (Pipe : Handle; Max_Wait : Duration := Forever)
         return Byte_String;

-- These operations are "record (message) oriented".  That is, the write
-- operation puts one record into the pipe which remembers the record and
-- its length.  When successful, the read operation reads exactly one
-- record (when unsuccessful, it reads 0 records), the Length out parameter
-- indicates the actual length of the record (as given by the corresponding
-- Write operation).

-- This is in contrast with the Device_Independent_Io (Dio) Byte_String
-- operations which are "stream oriented".  That is, the read operation
-- returns exactly the number of bytes that are requested, unless
-- end-of-file is reached, in which case fewer bytes are returned, as
-- indicated by the Length out parameter.

-- Given that pipes are record oriented, it is possible to write a program
-- which reads messages from a pipe, and copies them or sends them
-- somewhere else, without regard for the actual type of the data, and
-- preserving message boundaries.

-- The Read function is the same as the Read procedure except that it
-- internally allocates a Byte_String (of the length specified by the
-- Reader_Buffer_Size parameter of Create/Open) in which to read the
-- message, and then returns the first Length bytes.  For variable length
-- messages, this frees the client (of this package) from needing to know
-- the maximum message size.  In the current implementation, this
-- convenience is not free: the function makes an additional copy of the
-- message (as compared to the procedure), and it allocates
-- Reader_Buffer_Size - Length extra bytes in its stack frame. Of course,
-- if the function call site simply assigns the result into some variable,
-- there is an additional copy (as compared to the procedure).

-- Read and Write operations are atomic with respect to each other.  BUT,
-- This DOES NOT include multiple tasks reading/writing with the same
-- Handle.

-- Messages are passed by value.  That is, once Write completes, the entire
-- message is stored within the pipe.  Termination of the client (which
-- performed the Write) does not effect the state of the pipe.

-- Recall that a pipe has finite internal buffer capability.  A Write
-- operation which would exceed the maximum buffer size (defined at pipe
-- creation time) always raises Use_Error (and extended status
-- Item_Too_Big).  A Write operation which would exceed the remaining
-- buffer capacity is handled as follows: If Max_Wait time expires before
-- sufficient room becomes available in the buffer (this is immediately
-- true if Max_Wait = Dont_Wait), then raises Use_Error (and extended
```





```
-- status No_Room_In_Buffer).  When Use_Error is raised, the pipe is left
-- unmodified (except for overrun notification, as discussed below).  The
-- client can distinguish between these flavors of Use_Error via the
-- Get_Extended_Status operation, below.

-- In the event that there are multiple clients waiting to do Write, they
-- are typically serviced FIFO in order to avoid starvation.

-- Similarly, a Read operation specifies the maximum amount of time to
-- wait for the buffer to become non-empty.  A time of 0 indicates that the
-- client does not want to wait at all.  If the wait time expires before a
-- message is received by the client, then the client gets Use_Error, and
-- the pipe is left unchanged.

-- The Read operation returns a single message.  The Length parameter
-- indicates the actual number of bytes written into the Message parameter.
-- In the event that the actual message (supplied by the corresponding
-- Write operation) was longer than the Message parameter supplied to Read,
-- the client will receive Data_Error (and extended status of
-- Item_Too_Big), and the contents of the pipe are left unchanged.  In
-- future implementations, negative values of Length may be defined.

-- Recall that each message is read by exactly one Read operation; thus, in
-- the face of concurrent Reads, each client may see just a subset of the
-- messages that were written to the pipe.

-- In the event that there are multiple clients waiting to do Read, they
-- are typically serviced LIFO.  We assume that the application considers
-- all readers to be equivalent.  In this context, LIFO is better than FIFO
-- because it minimizes the working set of the readers.  (LIFO causes the
-- reader which most recently finished working to be the next to receive a
-- message).  This simplifies applications which need to choose the number
-- of readers; they can simply pick the maximum number of readers which can
-- operate in parallel.

-- The implementation of Read and Write waiting can handle aborts of
-- clients.

-- Specifying infinite wait times allows one to use the finite buffer
-- capacity as a flow control mechanism.

-- "end of file" (eof) messages are written into a pipe via the
-- Write_End_Of_File operation, and implicitly via Close (which itself may
-- be implicit via action abandon, which itself may be implicit ...).  When
-- a Read operation encounters an eof message, it is consumed, and
-- End_Error is raised.  Unlike other sequential media, one can read an
-- eof only once.

-- "Overrun" refers to a situation in which the writer does not wait
```

```
    -- forever for buffer space to become available and drops the unsent
    -- messages on the floor.  Pipes include the following mechanism for
    -- detecting overrun:

    -- In addition to messages of type data and eof, there are messages of type
    -- overrun.  A Write operation which raises Use_Error (because there is
    -- insufficient room in the buffer) appends a message of type overrun.
    -- Adjacent overrun messages are coalesced into a single overrun message.
    -- The Read operation consumes the overrun message (when encountered) and
    -- raises Use_Error.  Like eof, an overrun message can only be read once.

    -- Death of a client that has the pipe open for update may sometimes cause
    -- an overrun to be placed in the pipe.

    -- Observations: (1) The writer should probably not "poll" the pipe by
    -- using a short Max_Wait, since each unsuccessful attempt will append an
    -- overrun message, causing the reader to get a Use_Error.  (2) The reader
    -- can distinguish between timeout and overrun (both raise Use_Error) by
    -- using the Extended_Status function, below.

    -- EXCEPTIONS:
    --     Status_Error  : The given Handle is not open
    --     Mode_Error    : Write: Handle was Open'd for Exclusive_Read
    --                       or Shared_Read
    --                     Read: Handle was Open'd for Exclusive_Write
    --                       or Shared_Write
    --     Use_Error     : Write: Max_Wait expired,
    --                       or Message'length is larger than buffer size;
    --                     Read: Max_Wait expired,
    --                       or just consumed an overrun message
    --     Data_Error    : Read: Message'length is shorter than next message
    --                   : Read/Write: touching Message caused
    --                       Nonexistant_Page_Error
    --                   : Read: storing into Message caused
    --                       Write_To_Read_Only_Page
    --     End_Error     : Read: just consumed an end-of-file message
    --     Device_Error  : internal errors

    pragma Consume_Offset (4);

    generic
        type Element_Type is private;

    package Type_Specific_Operations is

        procedure Write (Pipe     : in out Handle;
                         Message  :        Element_Type;
                         Max_Wait :        Duration := Forever);
```





```
        procedure Read (Pipe     : in out Handle;
                        Message  : out    Element_Type;
                        Max_Wait :        Duration := Forever);

        function Read (Pipe : Handle; Max_Wait : Duration := Forever)
                 return Element_Type;

        pragma Consume_Offset;

    end Type_Specific_Operations;

    -- The usual "legal type for IO" rules apply to Element_Type. In
    -- particular, Element_Type cannot be (or contain) pointers or tasks.
    -- Both ends of the pipe should instantiate this package with the same
    -- type, else one will get implicit unchecked conversions, and might
    -- get Data_Error.

    -- The generic Write operation first normalizes the Message, converts the
    -- bits (of the Message) into a Byte_String (adding up to 7 bits of
    -- padding, as necessary), and then calls the non-generic Write.

    -- By normalize, we mean the following.  For record types, if the object is
    -- not constrained, allocate a constained instance of the object and copy
    -- the Message into the constrained copy.  Note that this is expensive,
    -- since it involves declaring collections and doing copies.  For array
    -- types, if the "bounds with object"ness of the Message is not the same as
    -- that of the Element_Type (argument to the generic), then a copy is made
    -- to convert the Message to the same boundedness as the Element_Type.

    -- The generic Read procedure calls the non-generic Read procedure to fetch
    -- the padded Byte_String, does an implicit unchecked conversion to
    -- Element_Type, and assigns it to the out parameter.

    -- The conversion may cause Data_Error to be raised when Element_Type is
    -- not "compatable" with the actual bits in the message; this might happen
    -- if the Write generic was instantiated with a different type than the
    -- Read generic, for example.  Some conditions that may cause
    -- incompatibility: The 'size of the result of the unchecked conversion
    -- (rounded to a byte) is not the same as the actual byte length of the
    -- received message. The Element_Type is unconstrained and the message is
    -- garbage (when interpreted according to Element_Type).

    -- The assignment follows Ada semantics, and may therefore fail for a
    -- variety of reasons, causing Data_Error. Some conditions that may cause
    -- the assignment to fail: Element_Type is an unconstrained array type
    -- (such as String), and the 'length of the string value in the buffer is
    -- not the same as the 'length of the Message out parameter.  The
    -- Element_Type is unconstrained and the message is garbage (when
```

```
-- interpreted according to Element_Type).

-- The generic Read function calls the non-generic Read function, does an
-- implicit unchecked conversion to Element_Type, and returns the result.
-- Data_Error may be raised when Element_Type is not "compatible" with the
-- actual bits in the message, as for the Read procedure.

-- EXCEPTIONS (in addition to those raised by the non-generic forms):
--     Data_Error    : Read: bits in the actual message are not
--                         "compatible" with Element_Type, or := failed.
--                     Pkg instantiation: raised when Element_Type has
--                         task or access/heap_access components.

function End_Of_File (Pipe : Handle) return Boolean;

-- Returns true iff a read operation would have caused End_Error to be
-- raised.

-- EXCEPTIONS:
--     Status_Error  : The given Handle is not open
--     Device_Error  : internal errors

procedure Write_End_Of_File (Pipe     : in out Handle;
                             Max_Wait :        Duration := Forever);

-- Puts an end-of-file message into the pipe.  Note that Close (of a pipe
-- open for writing) may implicitly call this procedure.  Abandoning
-- an action (of a writer) may implicitly call this procedure.  With
-- respect to overruns, this call follows rules given for Write.

-- EXCEPTIONS:
--     Status_Error  : The given Handle is not open
--     Use_Error     : Max_Wait expired,
--                         or Message'length is larger than buffer size;
--     Device_Error  : internal errors

function Current_Message_Count (Pipe : Handle) return Natural;

-- Can be used to "poll" a pipe to see how many messages are queued up,
-- waiting to be read.

-- EXCEPTIONS:
--     Status_Error  : The given Handle is not open

function Max_Buffer_Size (Pipe : Handle) return Positive;

-- Return the buffer size of an open pipe.
```





```
-- EXCEPTIONS:
--     Status_Error  : The given Handle is not open

function Open_Action (Pipe : Handle) return Action_Id;

-- Returns the action by which the Handle has the pipe open.

-- EXCEPTIONS:
--     Status_Error  : The given Handle is not open

pragma Consume_Offset (3);

type Full_Status_Kinds is
   (Pipe_Status, Directory_Error_Status,
    Directory_Name_Status, Manager_Status, U4, U5, U6, U7);

function Get_Full_Status_Kind (Pipe : Handle) return Full_Status_Kinds;

-- Defined iff the Handle is currently open and the last PROCEDURE call on
-- the Handle raised an exception and the following table indicates that
-- additional status is available.
--      Status_Error no additional status
--      Mode_Error no additional status
--      Name_Error more status available
--      Use_Error more status available
--      Device_Error more status available
--      End_Error more status available
--      Data_Error more status available
--      Layout_Error no additional status In this case, indicates which
-- kind of additional status information is available about the exception.

type Extended_Status is (Internal_Pipe_Error, Item_Too_Big,
                         No_Room_In_Buffer, Buffer_Is_Empty,
                         Behind_Other_Readers, Read_An_Eof, Read_An_Overrun,
                         Missing_Page, Read_Only_Page, U09,
                         U10, U11, U12, U13, U14, U15, U16);

function Get_Extended_Status        (Pipe : Handle) return Extended_Status;
function Get_Directory_Error_Status (Pipe : Handle) return Integer;
function Get_Directory_Name_Status  (Pipe : Handle) return Integer;
function Get_Manager_Status         (Pipe : Handle) return Operate_Status;

-- The above are defined iff Get_Full_Status_Kind is defined and returns
-- the corresponding value of Full_Status_Kinds. Rational reserves the
-- right to add additional Extended_Status values. Otherwise, it's ok
-- to program against Extended_Status values. The integer values returned
-- by the last 3 functions are for debugging only, and may change between
-- between releases of this software.
```

```
    function Status_Explanation (Pipe : Handle) return String;

    -- Returns, in string form, the best explanation of the status that is
    -- currently available.  This explanation may include additional internal
    -- state information.  The returned string may differ between releases of
    -- this software.

    generic
        with procedure Put_Line (S : String);
    procedure Put_Pipe_Internal_State (Pipe      : Handle;
                                       Depth     : Natural := 25;
                                       Get_Locks : Boolean := False);
    generic
        with procedure Put_Line (S : String);
    procedure Put_Internal_State (Depth     : Natural := 25;
                                  Get_Locks : Boolean := False);

    -- These operations are primarily intended for use as debugging
    -- aids by Rational personnel.  However, it is also possible for customers
    -- to use this information to debug their applications.  The format of the
    -- of the information fed through Put_Line may change in future releases.

    -- The first operation gives you more information if the Handle is for an
    -- open pipe!  Depth is used to keep various algorithms from going into an
    -- infinite loop when the internal data structures for the pipe are
    -- inconsistent.  Get_Locks indicates whether or not the internal data
    -- structures should be viewed from within the appropriate critical
    -- regions; in the current implementation, only the default is supported.

    function Debug_Image (Pipe            : Handle;
                          Level           : Natural;
                          Prefix          : String;
                          Expand_Pointers : Boolean) return String;
    --
    -- Daemon control.  The interval specifies how often the pipe daemon
    -- runs.  It defaults to every 30 seconds at low CPU priority.
    --
    -- Run_Daemon will cause the daemon to run at the priority of the
    -- calling task.  Note that this might actually cause the daemon
    -- to run twice if it is currently scheduled and blocked, since it
    -- has to finish (at the low priority) before this call can run it.

    function Get_Daemon_Interval return Duration;
    procedure Set_Daemon_Interval (Interval : Duration);
    procedure Run_Daemon;

end Pipe;
```





```
with Action;
with Default;
with Directory;
with Io_Exceptions;
with System;

package Polymorphic_Io is

    -- Provides the basic file abstraction on top of the package directory
    -- and file object manager abstractions.  Understanding actions is not
    -- necessary to use this level;  parameters are always defaulted to be
    -- single, queued actions.  Intended users are the Ada LRM Chapter 14
    -- IO packages, as well as sophisticated users that require more facilities
    -- than those provided in Chapter 14.

    subtype File is Directory.Object;

    subtype Version is Directory.Version;

    function Get_Class return Directory.Class;

    subtype Error_Status is Directory.Error_Status;

    type Handle is limited private;
    -- Handle that is needed to do anything to a file.

    function Nil                     return Handle;
    function Is_Nil (File : Handle) return Boolean;

    type File_Mode is (Read_Only, Write_Only, Read_Write, None);

    type File_Position is private;
    -- Logical file pointer that is needed to input and output operations.

    function Nil return File_Position;
    function First return File_Position;
    function Is_Nil   (Position : File_Position) return Boolean;
    function Is_First (Position : File_Position) return Boolean;

    function "<"  (Left, Right : File_Position) return Boolean;
    function "<=" (Left, Right : File_Position) return Boolean;
    function ">"  (Left, Right : File_Position) return Boolean;
    function ">=" (Left, Right : File_Position) return Boolean;

    package Naming renames Directory.Naming;

    subtype Context is Directory.Naming.Context;
```

```
function Default_Context
            (For_Job : Default.Process_Id := Default.Process) return Context
    renames Directory.Naming.Default_Context;

procedure Open (The_Handle     : in out Handle;
                Mode           : File_Mode;
                File_Name      : Naming.Name;
                Status         : out Error_Status;
                The_Version    : Directory.Version_Name :=
                   Directory.Default_Version;
                The_Context    : Context := Polymorphic_Io.Default_Context;
                Action_Id      : Action.Id := Action.Null_Id;
                Max_Wait       : Duration := Directory.Default_Wait;
                Prevent_Backup : Boolean := False);

procedure Open (The_Handle     : in out Handle;
                Mode           :        File_Mode;
                The_Object     :        File;
                Status         : out    Error_Status;
                The_Version    :        Directory.Version_Name :=
                   Directory.Default_Version;
                Action_Id      :        Action.Id := Action.Null_Id;
                Max_Wait       :        Duration := Directory.Default_Wait;
                Prevent_Backup :        Boolean := False);

procedure Open (The_Handle     : in out Handle;
                Mode           :        File_Mode;
                The_Version    : in out Version;
                Status         : out    Error_Status;
                Action_Id      :        Action.Id := Action.Null_Id;
                Max_Wait       :        Duration  := Directory.Default_Wait;
                Prevent_Backup :        Boolean   := False);

procedure Close (File : in out Handle; Status : out Error_Status);
-- Close a previously opened File.

procedure Delete (File : in out Handle; Status : out Error_Status);
-- Delete a previously opened File.  File cannot have been opened for Read.
-- Commit any action opened on behalf of the user

function Is_Open   (File : Handle) return Boolean;
function Mode      (File : Handle) return File_Mode;
function Name      (File : Handle) return Naming.Simple_Name;
function Full_Name (File : Handle) return Naming.Name;
-- Extract information about an open File.

function End_Of_File
            (File : Handle; Position : File_Position) return Boolean;
-- TRUE => Position is past end_of_File
```





```
function First_Free_Position (File : Handle) return File_Position;
-- Determine the first free (ie. non-existent) position within File.

function Size (File : Handle) return Long_Integer;
-- size of file in bits

generic
    type Element is private;
    -- Element must be constrained and "safe".

package Direct_Operations is

    function Compute (In_File : Handle;
                      Index   : Positive;
                      Base    : File_Position := Polymorphic_Io.First)
                     return File_Position;
    -- Determine the File_Position of the Index'th Element past Base.

    function Read (From_File : Handle; At_Position : File_Position)
                 return Element;
    -- Yield the Element at the specified position in From_File.
    -- If At_Position >= Free (From_File), END_ERROR is raised.
    -- If the system can detect that no element has ever been
    -- written At_Position, HOLE_ERROR is raised.

    procedure Write (To_File     : Handle;
                     At_Position : File_Position;
                     Value       : Element);
    -- Store the Value at the specified position in To_File.
    -- If At_Position + Value'Size >= Free (To_File), To_File is
    -- extended so that Free (To_File) = At_Position + Value'Size.

end Direct_Operations;

generic
    type Element is private;
    -- must be "safe"
package Sequential_Operations is

    function Next (In_File : Handle; After : File_Position)
                 return File_Position;
    -- Move to the next Element in the specified file beyond After.
    -- If After >= Free (In_File), END_ERROR is raised.

    function Read (From_File : Handle; At_Position : File_Position)
                 return Element;
    -- Yield the Element at the specified position in From_File.
    -- If At_Position >= Free (From_File), END_ERROR is raised.
    -- If the system can detect that no element has ever been
```

```
    -- written At_Position, HOLE_ERROR is raised.

    procedure Write (To_File     : Handle;
                     At_Position : File_Position;
                     Value       : Element);
    -- Store the Value at the specified position in To_File.
    -- If At_Position + Value'Size >= Free (To_File), To_File is
    -- extended so that Free (To_File) = At_Position + Value'Size.

end Sequential_Operations;

generic
    type Element         is private;
    type Element_Pointer is access Element;
    pragma Segmented_Heap (Element_Pointer);
package Access_Operations is

    function Reference (From_File : Handle; At_Position : File_Position)
                       return Element_Pointer;
    -- return a reference to the element "at_position"

    function Position (From_File : Handle; Pointer : Element_Pointer)
                      return File_Position;

    -- return position of element referenced by Pointer.
end Access_Operations;


Status_Error : exception renames Io_Exceptions.Status_Error;
Mode_Error   : exception renames Io_Exceptions.Mode_Error;
End_Error    : exception renames Io_Exceptions.End_Error;
Data_Error   : exception renames Io_Exceptions.Data_Error;


----------------------------------------------
-- CONVERSION OPERATIONS for FILE_POSITIONS --
----------------------------------------------
function Convert (Pos : File_Position) return Long_Integer;
function Convert (Pos : Long_Integer)  return File_Position;

procedure Save (File             : in out Handle;
                Status           : out    Error_Status;
                Immediate_Effect :        Boolean := False);

function Get_Action (File : Handle) return Action.Id;

package String_Operations is
    subtype Byte        is System.Byte;
    subtype Byte_String is System.Byte_String;
```





```
    procedure Read (File  :        Handle;
                    Pos   : in out File_Position;
                    Item  : out    Byte_String;
                    Count : out    Natural);
    procedure Read (File :        Handle;
                    Pos  : in out File_Position;
                    Item : out    Byte);
    procedure Read (File  :        Handle;
                    Pos   : in out File_Position;
                    Item  : out    String;
                    Count : out    Natural);
    procedure Read (File :        Handle;
                    Pos  : in out File_Position;
                    Item : out    Character);

    procedure Write (File :        Handle;
                     Pos  : in out File_Position;
                     Item :        Byte_String);
    procedure Write
               (File : Handle; Pos : in out File_Position; Item : Byte);
    procedure Write
               (File : Handle; Pos : in out File_Position; Item : String);
    procedure Write (File :        Handle;
                     Pos  : in out File_Position;
                     Item :        Character);

end String_Operations;

procedure Truncate (File : Handle;
                    Pos  : File_Position := Polymorphic_Io.First);
-- Shortens the file so that Pos is the first position outside the file.
-- Will not make the file bigger if Pos is larger than the current size of
-- the file.

end Polymorphic_Io;
```

```
with Io_Exceptions;
with Device_Independent_Io;

package Polymorphic_Sequential_Io is

    type File_Type is limited private;

    type File_Mode is (In_File, Out_File);


    -- File management

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

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

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

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

    function Is_Open (File : File_Type) return Boolean;


    -- Input and output operations

    generic
        type Element_Type is private;
    package Operations is
        procedure Read  (File : File_Type; Item : out Element_Type);
        procedure Write (File : File_Type; Item : Element_Type);
    end Operations;

    function End_Of_File (File : File_Type) return Boolean;

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





```
    -- Exceptions

    Status_Error : exception renames Io_Exceptions.Status_Error;
    Mode_Error   : exception renames Io_Exceptions.Mode_Error;
    Name_Error   : exception renames Io_Exceptions.Name_Error;
    Use_Error    : exception renames Io_Exceptions.Use_Error;
    Device_Error : exception renames Io_Exceptions.Device_Error;
    End_Error    : exception renames Io_Exceptions.End_Error;
    Data_Error   : exception renames Io_Exceptions.Data_Error;

end Polymorphic_Sequential_Io;
```

```
with Io_Exceptions;
with Device_Independent_Io;

generic
    type Element_Type is private;

package Sequential_Io is

    type File_Type is limited private;

    type File_Mode is (In_File, Out_File);


    -- File management

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

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

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

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

    function Is_Open (File : File_Type) return Boolean;


    -- Input and output operations

    procedure Read  (File : File_Type; Item : out Element_Type);
    procedure Write (File : File_Type; Item : Element_Type);

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





```
    -- Exceptions

    Status_Error : exception renames Io_Exceptions.Status_Error;
    Mode_Error   : exception renames Io_Exceptions.Mode_Error;
    Name_Error   : exception renames Io_Exceptions.Name_Error;
    Use_Error    : exception renames Io_Exceptions.Use_Error;
    Device_Error : exception renames Io_Exceptions.Device_Error;
    End_Error    : exception renames Io_Exceptions.End_Error;
    Data_Error   : exception renames Io_Exceptions.Data_Error;

private
    type File_Type is new Device_Independent_Io.File_Type;

end Sequential_Io;
```

```
with Device_Independent_Io;
with System;

package Tape_Specific is

    subtype File_Type   is Device_Independent_Io.File_Type;
    subtype Byte_Range  is Natural range 0 .. 4096;
    subtype Pipe_Range  is Natural range 0 .. 8;
    subtype Byte_String is System.Byte_String;

    type On_Off is (On, Off);

    procedure Set_Block_Size (File : File_Type; Size : Byte_Range);
    -- default is Recommended_Max_Block_Length

    procedure Set_Streaming_Mode (File : File_Type; Mode : On_Off);
    -- on = true turns streaming mode on
    -- on = false turns streaming mode off
    -- default is off

    procedure Set_Pipeline_Size (File : File_Type; Size : Pipe_Range);
    -- pipeline size to use if in streaming mode
    -- default is Recommended_Pipeline_Size

    procedure Unload (File : File_Type);
    -- the "file" is closed
    -- the tape drive unloads the tape

    procedure Rewind (File : File_Type);
    -- the tape is put at beginning of tape

    type Skip_Records_Obstacles is
       (None,                      -- No obstacle encountered
        Tape_Mark,                 -- Tape mark encountered,
        Bot                        -- Beginning of tape was encountered while
                                   --   while skipping backwards
        );

    type Skip_Marks_Obstacles is
       (None,                    -- No obstacle encountered
        Double_Tape_Mark,        -- 2 consecutive tape marks were encountered
        --   while skipping forward
        Bot                      -- Beginning of tape was encountered while
                                 --   while skipping backwards
        );
```





```
    type Error_Status is
       (Success,                -- No error encountered
        Record_Length_Long,     -- Record on tape was longer than parameter
        Not_On_Line,            -- Drive was offline
        Retry_Count_Exhausted,  -- Record/tape mark can't be read/written
        Unexpected_Tape_Error,  -- Tape position lost, rewind or unload it
        Unit_Is_Bad);           -- Call Field Service

    -- The following procedures that do not return an error status will have
    -- the exception DATA_ERROR raised if RECORD_LENGTH_LONG would have been
    -- returned.  DEVICE_ERROR is raised for all other non-SUCCESS statuses.

    procedure Unload (File : File_Type; Status : out Error_Status);
    -- the "file" is closed
    -- the tape drive unloads the tape

    procedure Rewind (File : File_Type; Status : out Error_Status);
    -- the tape is put at beginning of tape

    -- The following should NOT be intermingled with the Read and Write
    -- procedures in Device_Independent_IO for the same file.

    -- The READ procedures return the next record of data on the tape.
    -- COUNT returns the actual size of the physical tape record in bytes.
    -- Only the first COUNT elements of RECRD are valid.
    -- If RECORD_LENGTH_LONG is returned as the error status, then RECRD
    -- contains the first RECRD'LENGTH bytes of the physical tape record and
    -- COUNT = RECRD'LENGTH.  If a tape mark was read, then COUNT = 0.

    procedure Read (File  :     File_Type;
                    Recrd : out Byte_String;
                    Count : out Natural);

    procedure Read (File   :     File_Type;
                    Recrd  : out Byte_String;
                    Count  : out Natural;
                    Status : out Error_Status);

    procedure Read (File : File_Type; Recrd : out String; Count : out Natural);

    procedure Read (File   :     File_Type;
                    Recrd  : out String;
                    Count  : out Natural;
                    Status : out Error_Status);

    -- The two IS_MARK subprograms return whether the next tape record to be
    -- read is a tape mark.  These subprograms should only be used while in
    -- streaming mode otherwise they will raise USE_ERROR.  They will raise
    -- MODE_ERROR if the file is not open for input.
```

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

procedure Is_Mark (File   :     File_Type;
                   Result : out Boolean;
                   Status : out Error_Status);

-- The WRITE procedures write the contents of RECRD on the tape as a
-- physical tape record.  RECRD'LENGTH must be greater than or equal to 18
-- and less than or equal to the ABSOLUTE_MAX_BLOCK_LENGTH (currently
-- 4096) otherwise USE_ERROR will be raised.

-- PAST_EOT_MARKER indicates that the area beyond the reflective EOT marker
-- on the tape is now being written.  Users are cautioned that tape
-- standards specify that there is at least 25 ft. of tape from the marker
-- to the end of the reel, but only the first 10 ft. are useable.  It
-- is OK to write in this 10 ft. area but writing beyond that runs the
-- risk of running the tape off its reel.

procedure Write (File            :     File_Type;
                 Recrd           :     Byte_String;
                 Past_Eot_Marker : out Boolean);

procedure Write (File            :     File_Type;
                 Recrd           :     Byte_String;
                 Past_Eot_Marker : out Boolean;
                 Status          : out Error_Status);

procedure Write (File            :     File_Type;
                 Recrd           :     String;
                 Past_Eot_Marker : out Boolean);

procedure Write (File            :     File_Type;
                 Recrd           :     String;
                 Past_Eot_Marker : out Boolean;
                 Status          : out Error_Status);

-- The WRITE_MARK procedures cause a tape mark to be written to the tape.

procedure Write_Mark (File : File_Type; Past_Eot_Marker : out Boolean);

procedure Write_Mark (File            :     File_Type;
                      Past_Eot_Marker : out Boolean;
                      Status          : out Error_Status);

-- The SKIP_RECORDS procedures position the tape either forward
-- or backward until ABS (NUM_RECORDS_TO_SKIP) have been skipped or
-- an obstacle has been encountered.  A positive NUM_RECORDS_TO_SKIP
-- implies skipping forward; negative implies skipping backward; zero
-- implies no movement.  If a tape mark is encountered as an obstacle,
```





```
-- the position of the tape is on the "other side" of the tape mark; i.e.,
-- when skipping backward, the next item read would be that same tape mark
-- or when skipping forward, the next item read would be the record or
-- tape mark beyond the obstacle tape mark.  The RECORDS_SKIPPED does
-- not include the tape mark.  If the Beginning-Of-Tape reflective marker
-- is encountered while skipping backward, the position of the tape will
-- be at the beginning of the tape, ready to read the first record.
-- MODE_ERROR is raised if the file is not open for reading.  USE_ERROR
-- is raised if the file is in streaming mode.

procedure Skip_Records (File                :     File_Type;
                        Num_Records_To_Skip :     Integer;
                        Obstacle            : out Skip_Records_Obstacles;
                        Records_Skipped     : out Natural);

procedure Skip_Records (File                :     File_Type;
                        Num_Records_To_Skip :     Integer;
                        Obstacle            : out Skip_Records_Obstacles;
                        Records_Skipped     : out Natural;
                        Status              : out Error_Status);

-- The SKIP_TAPE_MARKS procedures position the tape either forward or
-- backward until ABS (NUM_MARKS_TO_SKIP) have been skipped or
-- an obstacle has been encountered.  A positive NUM_MARKS_TO_SKIP
-- implies skipping forward; negative implies skipping backward; zero
-- implies no movement.  Two consecutive tape marks (a double tape mark)
-- is only an obstacle while skipping forward; in which case neither of
-- the tape marks is counted in MARKS_SKIPPED.  If two consecutive tape
-- marks are encountered while skipping backward, it is not an obstacle
-- and they are treated as individual tape marks in MARKS_SKIPPED.  If
-- the Beginning-Of-Tape reflective marker is encountered while skipping
-- backward, the position of the tape will be at the beginning of the
-- tape, ready to read the first record.  If no obstacle was encountered,
-- the position of the tape is on the "other side" of the last tape mark.
-- MODE_ERROR is raised if the file is not open for reading.  USE_ERROR
-- is raised if the file is in streaming mode.

procedure Skip_Tape_Marks (File              :     File_Type;
                           Num_Marks_To_Skip :     Integer;
                           Obstacle          : out Skip_Marks_Obstacles;
                           Marks_Skipped     : out Natural);

procedure Skip_Tape_Marks (File              :     File_Type;
                           Num_Marks_To_Skip :     Integer;
                           Obstacle          : out Skip_Marks_Obstacles;
                           Marks_Skipped     : out Natural;
                           Status            : out Error_Status);
end Tape_Specific;
```

```
with Device_Independent_Io;
with System;

package Terminal_Specific is

    --  This package supports operations that are specific to
    --  "terminals".  For this purpose, a terminal is an object of
    --  type Terminal.  These objects are in !Machine.Devices.
    --
    --  Normal Text_IO-style IO to the terminal is done through:
    --      !USERS.user.session     Standard_Output
    --      !MACHINE.USERS.user     Standard_Error
    --
    --  Window_IO provides quarter-plane, addressable display and key input
    --
    --  Access to the terminal for Standard_Output, Standard_Error and
    --  Window_IO is handled by the job controlling the session, so there may
    --  be multiple, simultaneously-active windows.
    --
    --  Opening a terminal directly provides the application complete control
    --  of the terminal.  In this case, the terminal is controlled by the job
    --  that opens it, NOT the session job.
    --
    --  More than one job can have a terminal open at the same time, but
    --  only one of them will actually receive input or transmit output.
    --  Any others will be blocked on both input and output.  A job that
    --  references the terminal directly and simultaneously uses one of the
    --  session-controlled forms of terminal interaction will not work well and
    --  may deadlock.
    --
    --  Attempts to open an enabled terminal other than the one for
    --  current session will fail.  Control over enabled/disabled status
    --  of terminals is available in the Operator package.
    --
    --  The determination of which of the various jobs dealing with the
    --  terminal actually have the right to transmit/receive is done on
    --  the basis of which job is "connected".  There is at most one
    --  connected job at any time.  If a job that has the terminal open
    --  is currently connected, it has the terminal.  If it disconnects
    --  or is terminated, control of the terminal reverts to the session.
    --  The user can return control of the terminal to the application
    --  by doing a Job.Connect with the appropriate job number.
    --
    --  Transfers of terminal ownership are detectable as part of the
    --  status of the Read and Write operations.  This allows
    --  applications that support disconnect to detect when to redraw
    --  their version of the screen.
```





```
    --
    --  The following device-specific Form options are supported:
    --
    --      Option      Explanation                 Default
    --
    --      Echo        whether to echo input       True
    --      Edit        Line editing or None        Line
    --      CRLF        map LF to CRLF              True
    --
    --  Note: the CRLF option is ignored by the Write procedures in this
    --  package to reduce confusion over whether the CR was transferred for a
    --  particular count.  CRLF is honored by device_independent write/put
    --  operations.
    --

    subtype File_Type   is Device_Independent_Io.File_Type;
    subtype Byte_String is Device_Independent_Io.Byte_String;

    package Status is
        type Code is new Integer;
        function Image (Value : Code) return String;
        --
        --      Status.Code                 Standard reaction
        --
        --      Normal                      none
        --      Break                       raise End_Error
        --      Disconnect                  raise End_Error
        --      Timed_Out                   0 data bytes transferred
        --      Data_Error                  raise Data_Error
        --      Data_Overrun                raise Device_Error
        --      Lost_Ownership              ignored
        --      Gained_Ownership            ignored
        --      Too_Many_Clients            raise Device_Error
        --
        --  Standard Read/Write routines in Device_Independent_IO use:
        --      Duration'Last for Wait parameters
        --      "Standard Reaction" for Result parameters
        --
        Normal           : constant Code := 0;
        Break            : constant Code := 1;
        Disconnect       : constant Code := 2;
        Not_Open         : constant Code := 3;
        Timed_Out        : constant Code := 4;
        Data_Error       : constant Code := 5;
        Data_Overrun     : constant Code := 6;
        Lost_Ownership   : constant Code := 7;
        Gained_Ownership : constant Code := 8;
        Too_Many_Clients : constant Code := 9;
    end Status;
```

```
package Output is

    procedure Map_Lf_To_Crlf (File : File_Type; Value : Boolean := True);
    -- Equivalent of CRLF Form option; default True

    procedure Transmit_Break (File : File_Type);

    procedure Transmit_Break (File   :     File_Type;
                              Wait   :     Duration;
                              Result : out Status.Code);

    procedure Disconnect (File : File_Type);

    procedure Disconnect (File   :     File_Type;
                          Wait   :     Duration;
                          Result : out Status.Code);

    procedure Wait_For_Transmission (File : File_Type);
    -- Wait for all previously written data to be transmitted.

    procedure Set_Rts (File : File_Type; On : Boolean);
    -- Set the current state of the RTS (pin 4) RS-232
    -- modem control output.  True => ON, False => OFF.

    procedure Set_Dtr (File : File_Type; On : Boolean);
    -- Set the current state of the DTR (pin 20) RS-232
    -- modem control output.  True => ON, False => OFF.

end Output;

package Input is

    procedure Flush (File : File_Type);

    procedure Set_Echo (File : File_Type; Echo : Boolean := True);
    -- Equivalent to Echo Form option; default True

    function Get_Echo (File : File_Type) return Boolean;

    procedure Set_Editing (File : File_Type; Mode : String := "Line");
    -- Equivalent to the Edit Form option; default Edit => Line
    -- Disabled with value None.

    function Get_Editing (File : File_Type) return String;

end Input;
```





```
    procedure Read (File  :     File_Type;
                    Item  : out Byte_String;
                    Count : out Natural;
                    Wait  :     Duration);

    procedure Read (File  :     File_Type;
                    Item  : out String;
                    Count : out Natural;
                    Wait  :     Duration);

    procedure Read (File   :     File_Type;
                    Item   : out Byte_String;
                    Count  : out Natural;
                    Wait   :     Duration;
                    Result : out Status.Code);

    procedure Read  (File   :     File_Type;
                     Item   : out String;
                     Count  : out Natural;
                     Wait   :     Duration;
                     Result : out Status.Code);
    procedure Write (File  :     File_Type;
                     Item  :     Byte_String;
                     Count : out Natural;
                     Wait  :     Duration);

    procedure Write (File  :     File_Type;
                     Item  :     String;
                     Count : out Natural;
                     Wait  :     Duration);
    procedure Write (File   :     File_Type;
                     Item   :     Byte_String;
                     Count  : out Natural;
                     Wait   :     Duration;
                     Result : out Status.Code);

    procedure Write (File   :     File_Type;
                     Item   :     String;
                     Count  : out Natural;
                     Wait   :     Duration;
                     Result : out Status.Code);

end Terminal_Specific;
```

```
with Io_Exceptions;

package Text_Io is

    type File_Type is limited private;

    type File_Mode is (In_File, Out_File);

    type    Count           is range 0 .. 1_000_000_000;
    subtype Positive_Count is Count range 1 .. Count'Last;
    Unbounded : constant Count := 0;  -- line and page length

    subtype Field       is Integer range 0 .. Integer'Last;
    subtype Number_Base is Integer range 2 .. 16;

    type Type_Set is (Lower_Case, Upper_Case);


    -- File Management

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

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

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

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

    function Is_Open (File : File_Type) return Boolean;


    -- Control of default input and output files

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





```
    function Standard_Input  return File_Type;
    function Standard_Output return File_Type;

    function Current_Input  return File_Type;
    function Current_Output return File_Type;


    -- Specification of line and page lengths

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

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

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

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


    -- Column, Line and Page Control

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

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

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

    procedure New_Page (File : File_Type);
    procedure New_Page;

    procedure Skip_Page (File : File_Type);
    procedure Skip_Page;

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

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


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

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


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

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

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


    -- Character Input-Output

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


    -- String Input-Output

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

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

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


    -- Generic package for Input-Output of Integer Types

    generic
        type Num is range <>;
    package Integer_Io is

        Default_Width : Field       := Num'Width;
        Default_Base  : Number_Base := 10;

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

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





```
        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);

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

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


    -- Generic package for Input-Output of Floating Point Types

    generic
        type Num is digits <>;
    package Float_Io is

        Default_Fore : Field := 2;
        Default_Aft  : Field := Num'Digits - 1;
        Default_Exp  : Field := 3;


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

        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);

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

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

```
    -- Generic package for Input-Output of Fixed Point Types

    generic
        type Num is delta <>;
    package Fixed_Io is

        Default_Fore : Field := Num'Fore;
        Default_Aft  : Field := Num'Aft;
        Default_Exp  : Field := 0;

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

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

        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);

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

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

    -- Generic package for Input-Output of Enumeration Types

    generic
        type Enum is (<>);
    package Enumeration_Io is

        Default_Width   : Field    := 0;
        Default_Setting : Type_Set := Upper_Case;

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

        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);

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

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


    -- Exceptions

    Status_Error : exception renames Io_Exceptions.Status_Error;
    Mode_Error   : exception renames Io_Exceptions.Mode_Error;
    Name_Error   : exception renames Io_Exceptions.Name_Error;
    Use_Error    : exception renames Io_Exceptions.Use_Error;
    Device_Error : exception renames Io_Exceptions.Device_Error;
    End_Error    : exception renames Io_Exceptions.End_Error;
    Data_Error   : exception renames Io_Exceptions.Data_Error;
    Layout_Error : exception renames Io_Exceptions.Layout_Error;

private
    type File_Type is new Device_Independent_Io.File_Type;

end Text_Io;
```

```
with Io_Exceptions;

package Window_Io is

    -- package for providing raw IO facilities to an image

    type File_Type is private;

    type File_Mode is (In_File, Out_File);
    -- the mode of the handle.  Each image can be opened twice - once
    -- for input and once for output.

    -- Create an image for IO.
    -- Normally, a new empty image is created on this call.
    -- If an image is already open for this job with the given name,
    -- and the mode given /= the mode the image is open for, that
    -- image will be opened for the new mode.
    procedure Create (File : in out File_Type;
                      Mode :        File_Mode := Out_File;
                      Name :        String;
                      Form :        String    := "");

    -- Open a previously closed image.  The same rules apply for create
    -- in the case one job opens the same image twice; once for input and
    -- once for output
    procedure Open (File : in out File_Type;
                    Mode :        File_Mode := Out_File;
                    Name :        String;
                    Form :        String    := "");

    -- Terminate operations on this image.
    procedure Close (File : in out File_Type);

    -- Delete the image.  Any other handles on this image are implicitly
    -- closed.
    procedure Delete (File : in out File_Type);

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

    function Is_Open (File : File_Type) return Boolean;

    package Raw is
        -- gain access to the keyboard for "raw" input.
        -- one channel may be opened per job.
        -- no echoing or local editing is performed.
```





```
        type Stream_Type is private;

        procedure Open  (Stream : in out Stream_Type);
        procedure Close (Stream              : in out Stream_Type;
                         Flush_Pending_Input :        Boolean := False);

        procedure Disconnect (Stream : in out Stream_Type);
        -- free users keyboard

        type Key        is new Natural range 0 .. 1023;
        type Key_String is array (Positive range <>) of Key;
        -- a key is the basic bit of input.

        subtype Simple_Key is Key range 0 .. 127;
        -- a simple key represents the ascii characters

        procedure Get (Stream : Stream_Type; Item : out Key);
        procedure Get (Stream : Stream_Type; Item : out Key_String);

        -- converting keys to characters
        -- the ascii characters map directly to the first 128 keys
        function Convert (C : Character)  return Simple_Key;
        function Convert (K : Simple_Key) return Character;

        -- keys are mapped to logical names
        -- these names correspond to the 'image attribute of the
        -- enumerations in machine.editor_data.visible_key_names

        subtype Terminal is String;
        -- supported terminal types are Cit500R, Vt100, Rational

        function Image (For_Key : Key; On_Terminal : Terminal) return String;
        -- image is "", if For_Key is not defined for this terminal type

        procedure Value (For_Key_Name :     String;
                         On_Terminal  :     Terminal;
                         Result       : out Key;
                         Found        : out Boolean);
        -- Found is false => For_Key_Name does not name a key on this terminal

        function Value
                  (For_Key_Name : String; On_Terminal : Terminal) return Key;

        Unknown_Key : exception;
        -- raised by functional form of value

    end Raw;
```

```
subtype Column_Number is Positive;
subtype Line_Number   is Positive;
-- a file_type is initialized to column 1, line 1

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

-- output
-- characters are displayed at the current cursor position
-- control characters are displayed in reverse-video

type Designation is (Text, Prompt, Protected);

type Attribute is
    record
        Bold        : Boolean;
        Faint       : Boolean;
        Underscore  : Boolean;
        Inverse     : Boolean;
        Slow_Blink  : Boolean;
        Rapid_Blink : Boolean;
        Unused_0    : Boolean;
        Unused_1    : Boolean;
    end record;

Vanilla : constant Attribute := (others => False);

type Character_Set is new Natural range 0 .. 15;

Plain    : constant Character_Set := 0;
Graphics : constant Character_Set := 1;

type Font is
    record
        Kind : Character_Set;
        Look : Attribute;
    end record;

Normal : constant Font := Font'(Plain, Vanilla);

function Default_Font (For_Type : Designation) return Font;
-- the fonts normally used by the environment for these designations
-- are returned

procedure Position_Cursor (File   : File_Type;
                           Line   : Line_Number   := Line_Number'First;
                           Column : Column_Number := Column_Number'First;
                           Offset : Natural       := 0);
```





```
-- Position the cursor on the image.
-- Offset is used to position the cursor relative to the top of the window.
-- With an offset of 0, the cursor is made visible in the window using
-- the normal editor defaults.
-- With a positive offset, the image is scrolled in the window so the
-- cursor is the offset line in the window.

procedure Move_Cursor (File          : File_Type;
                       Delta_Lines   : Integer;
                       Delta_Columns : Integer;
                       Offset        : Natural := 0);

procedure Report_Cursor (File   :     File_Type;
                         Line   : out Line_Number;
                         Column : out Column_Number);

procedure Overwrite (File  : File_Type;
                     Item  : Character;
                     Image : Font        := Normal;
                     Kind  : Designation := Text);
-- writes ITEM at the current cursor position and advances column by 1

procedure Overwrite (File  : File_Type;
                     Item  : String;
                     Image : Font        := Normal;
                     Kind  : Designation := Text);
-- writes ITEM at the current cursor position and advances column by
-- ITEM'LENGTH

procedure Insert (File  : File_Type;
                  Item  : Character;
                  Image : Font        := Normal;
                  Kind  : Designation := Text);
-- writes ITEM at the current cursor position and advances column by 1

procedure Insert (File  : File_Type;
                  Item  : String;
                  Image : Font        := Normal;
                  Kind  : Designation := Text);
-- writes ITEM at the current cursor position and advances column by
-- ITEM'LENGTH

procedure New_Line (File : File_Type; Lines : Count := 1);
-- insert lines after the current line
-- advances line by Lines, and sets column to 1.

procedure Delete (File : File_Type; Characters : Count);
-- deletes Count characters at current position.  Position is unchanged
```

```
procedure Delete_Lines (File : File_Type; Lines : Count := 1);
-- deletes Lines including the current line. Position is unchanged

-- input with editing
-- an input prompt with contents PROMPT will be displayed at the current
-- cursor position.  Control of the keyboard will be returned to the
-- core editor for user input at the prompt.

procedure Get      (File   :     File_Type;
                    Prompt :     String := "[input]";
                    Item   : out Character);
procedure Get      (File   :     File_Type;
                    Prompt :     String := "[input]";
                    Item   : out String);
procedure Get_Line (File   :     File_Type;
                    Prompt :     String := "[input]";
                    Item   : out String;
                    Last   : out Natural);
function  Get_Line
            (File : File_Type; Prompt : String := "[input]") return String;

-- banner operations

-- The value will be displayed in the banner for this image
-- fields are defined from left to right.  The first few fields
-- are reserved for the editor.  Users may specify field_names
-- of the form "FIELD_0" .. "FIELD_9". Currently 0 .. 2 are used
-- for job_number, start_time and blocked indication, but may be
-- reused by the user.
-- Calling set_banner with other values will be a noop.

procedure Set_Banner
             (File : File_Type; Field_Name : String; Value : String);

function Read_Banner (File : File_Type; Field_Name : String) return String;

-- predefined field_names, may be passed to Set_Banner
function Job_Number return String;
function Job_Time   return String;


-- sound the terminal bell
procedure Bell (File : File_Type);

-- information about the current image

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





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

    function Char_At (File : File_Type) return Character;
    function Font_At (File : File_Type) return Font;

    function Last_Line (File : File_Type) return Line_Number;

    -- information about the current window

    -- the origin is the line and column number of the point of the image
    -- located in the upper right corner of the window
    procedure Report_Origin (File   :     File_Type;
                             Line   : out Line_Number;
                             Column : out Column_Number);

    -- the size of the window in characters
    procedure Report_Size (File    :     File_Type;
                           Lines   : out Positive_Count;
                           Columns : out Positive_Count);

    -- the location of the window on the screen
    -- the upper right corner of the screen is line 1, column 1
    procedure Report_Location (File   :     File_Type;
                               Line   : out Line_Number;
                               Column : out Column_Number);

end Window_Io;
```

## !LRM

This tabbed section contains copies of the specifications for the Ada packages and subprograms defined in the world !Lrm. With the exception of minor formatting differences, these specifications are exact copies of those online.

The specifications in this section are organized alphabetically, as you would find them listed online. Units within a package are grouped together under the name of their enclosing package.

The standard contents of world !Lrm are listed below.

To find other locations where a package or subprogram is documented, see the Map of the Rational Environment Library System (behind the Environment Specifications tab) or see the Master Index.

```
!Lrm : Library (World):
  Calendar                          : Ada (Pack_Spec);
  System                            : Ada (Pack_Spec);
  Unchecked_Conversion              : Ada (Gen_Func);
  Unchecked_Deallocation            : Ada (Gen_Proc);
```





```
package Calendar is

    type Time is private;

    subtype Year_Number  is Integer range 1901 .. 2099;
    subtype Month_Number is Integer range 1 .. 12;
    subtype Day_Number   is Integer range 1 .. 31;
    subtype Day_Duration is Duration range 0.0 .. 86_400.0;

    function Clock return Time;

    function Year    (Date : Time) return Year_Number;
    function Month   (Date : Time) return Month_Number;
    function Day     (Date : Time) return Day_Number;
    function Seconds (Date : Time) return Day_Duration;

    procedure Split (Date    :     Time;
                     Year    : out Year_Number;
                     Month   : out Month_Number;
                     Day     : out Day_Number;
                     Seconds : out Day_Duration);

    function Time_Of (Year    : Year_Number;
                      Month   : Month_Number;
                      Day     : Day_Number;
                      Seconds : Day_Duration := 0.0) return Time;

    function "+" (Left : Time; Right : Duration) return Time;
    function "+" (Left : Duration; Right : Time) return Time;
    function "-" (Left : Time; Right : Duration) return Time;
    function "-" (Left : Time; Right : Time)     return Duration;

    function "<"  (Left, Right : Time) return Boolean;
    function "<=" (Left, Right : Time) return Boolean;
    function ">"  (Left, Right : Time) return Boolean;
    function ">=" (Left, Right : Time) return Boolean;

    Time_Error : exception;  -- can be raised by TIME_OF, "+" and "-"

end Calendar;
```

```
package System is

    pragma Read_Only;
    pragma Open_Private_Part;
    pragma Subsystem (Ada_Base);

    type Address is private;

    Null_Address : constant Address;

    type Name is (R1000);

    System_Name : constant Name := R1000;

    Bit          : constant := 1;
    Storage_Unit : constant := 1 * Bit;

    Word_Size   : constant := 128 * Bit;
    Byte_Size   : constant := 8 * Bit;
    Megabyte    : constant := (2 ** 20) * Byte_Size;
    Memory_Size : constant := 32 * Megabyte;


    -- System-Dependent Named Numbers

    Min_Int : constant := Long_Integer'Pos (Long_Integer'First);
    Max_Int : constant := Long_Integer'Pos (Long_Integer'Last);

    Max_Digits   : constant := 15;
    Max_Mantissa : constant := 63;
    Fine_Delta   : constant := 1.0 / (2.0 ** 63);
    Tick         : constant := 200.0E-9;

    subtype Priority is Integer range 0 .. 5;


    type Byte is new Natural range 0 .. 255;

    type Byte_String is array (Natural range <>) of Byte;
    -- Basic units of transmission/reception to/from IO devices.

    type    Virtual_Processor_Number is new Long_Integer range 0 .. 2 ** 10 - 1;
    type    Module_Number            is new Long_Integer range 0 .. 2 ** 22 - 1;
    type    Module_Name              is new Long_Integer range 0 .. 2 ** 32 - 1;
    subtype Code_Segment_Name        is Module_Name range 0 .. 2 ** 24 - 1;
    type    Bit_Offset               is new Long_Integer range 0 .. 2 ** 32 - 1;
```





```
    Null_Module : constant Module_Name := 0;

    function Convert (The_Address : Address) return Long_Integer;
    pragma Suppress (Elaboration_Check, Convert);

    function Extract_Vp (From_Address : Address)
                        return Virtual_Processor_Number;
    pragma Suppress (Elaboration_Check, Extract_Vp);

    function Extract_Number (From_Address : Address) return Module_Number;
    pragma Suppress (Elaboration_Check, Extract_Number);

    function Extract_Name (From_Address : Address) return Module_Name;
    pragma Suppress (Elaboration_Check, Extract_Name);

    function Extract_Offset (From_Address : Address) return Bit_Offset;
    pragma Suppress (Elaboration_Check, Extract_Offset);


    function Get_Vp (From_Name : Module_Name) return Virtual_Processor_Number;
    pragma Suppress (Elaboration_Check, Get_Vp);

    function Get_Number (From_Name : Module_Name) return Module_Number;
    pragma Suppress (Elaboration_Check, Get_Number);

    function Compose_Name (With_Vp     : Virtual_Processor_Number;
                           With_Number : Module_Number) return Module_Name;
    pragma Suppress (Elaboration_Check, Compose_Name);


    function Current_Name return Module_Name;
    pragma Suppress (Elaboration_Check, Current_Name);

    function Current_Vp return Virtual_Processor_Number;
    pragma Suppress (Elaboration_Check, Current_Vp);

    function Current_Number return Module_Number;
    pragma Suppress (Elaboration_Check, Current_Number);

    type Segment is private;

    Null_Segment : constant Segment;

    type Package_Type is private;
    pragma Enable_Runtime_Privacy (Package_Type);

    Null_Package : constant Package_Type;

    Invalid_Package_Value : exception;
```

```
    type Exception_Number is new Long_Integer range 0 .. 2 ** 48 - 1;

    Operand_Class_Error : exception;
    pragma Exception_Name (Operand_Class_Error, 96);

    Type_Error : exception;
    pragma Exception_Name (Type_Error, 97);

    Visibility_Error : exception;
    pragma Exception_Name (Visibility_Error, 98);

    Capability_Error : exception;
    pragma Exception_Name (Capability_Error, 99);

    Machine_Restriction : exception;
    pragma Exception_Name (Machine_Restriction, 100);

    Illegal_Instruction : exception;
    pragma Exception_Name (Illegal_Instruction, 101);

    Illegal_Reference : exception;
    pragma Exception_Name (Illegal_Reference, 102);

    Illegal_Frame_Exit : exception;
    pragma Exception_Name (Illegal_Frame_Exit, 103);

    Record_Field_Error : exception;
    pragma Exception_Name (Record_Field_Error, 104);

    Utility_Error : exception;
    pragma Exception_Name (Utility_Error, 105);

    Unsupported_Feature : exception;
    pragma Exception_Name (Unsupported_Feature, 106);

    Illegal_Heap_Access : exception;
    pragma Exception_Name (Illegal_Heap_Access, 107);

    Select_Use_Error : exception;
    pragma Exception_Name (Select_Use_Error, 108);


    Frame_Establish_Error : exception;
    pragma Exception_Name (Frame_Establish_Error, 129);

    Nonexistent_Space_Error : exception;
    pragma Exception_Name (Nonexistent_Space_Error, 131);

    Nonexistent_Page_Error : exception;
```





```
    pragma Exception_Name (Nonexistent_Page_Error, 132);

    Write_To_Read_Only_Page : exception;
    pragma Exception_Name (Write_To_Read_Only_Page, 133);

    Heap_Pointer_Copy_Error : exception;
    pragma Exception_Name (Heap_Pointer_Copy_Error, 134);

    Assertion_Error : exception;
    pragma Exception_Name (Assertion_Error, 135);

    Microcode_Assist_Error : exception;
    pragma Exception_Name (Microcode_Assist_Error, 136);

private

    type Address is new Long_Integer;

    Null_Address : constant Address := 0;

    type Segment is access Boolean;
    pragma Segmented_Heap (Segment);

    Null_Segment : constant Segment := null;

    type Package_Type is new Long_Integer;

    Null_Package : constant Package_Type := 0;

end System;
```

```
generic
    type Source is limited private;
    type Target is limited private;

function Unchecked_Conversion (S : Source) return Target;
```





```
generic
    type Object is limited private;
    type Name   is access Object;

procedure Unchecked_Deallocation (X : in out Name);
```

## !TOOLS

This tabbed section contains copies of the specifications for the Ada packages and subprograms defined in the world !Tools and its sublibraries. With the exception of minor formatting differences, these specifications are exact copies of those online.

The specifications in this section are organized alphabetically, as you would find them listed online. Units within a package or sublibrary are grouped together under the name of their enclosing package or sublibrary.

The standard contents of world !Tools are listed below. Those objects that are *not* documented in this section are noted.

To find other locations where a package or subprogram is documented, see the Map of the Rational Environment Library System (behind the Environment Specifications tab) or see the Master Index.

```
!Tools : Library (World);
  Access_List_Tools                    : Ada (Pack_Spec);
  Ada_Object_Editor                    : Ada (Pack_Spec);
  Ada_Text                             : Ada (Pack_Spec);
  Allows_Deallocation                  : Ada (Gen_Func);
  Bit_Operations                       : Ada (Pack_Spec);
  Bounded_String                       : Ada (Pack_Spec);
  Ci'Spec_View.Units                   : Library (Directory);
    .Ci                                : Ada (Pack_Spec);
    .Commands                          : Library (Directory);
      .Run                             : Ada (Pack_Spec);
      .Show                            : Ada (Pack_Spec);
      .Typ                             : Ada (Proc_Spec);
  Code_Segment_Object_Editor           : Ada (Pack_Spec);
  Compatibility'Spec_View.Units        : Library (Directory);
    .Cdb_Maintenance                   : Ada (Pack_Spec);
    .Check                             : Ada (Pack_Spec);
  Concurrent_Map_Generic               : Ada (Gen_Pack);
  Debug_Tools                          : Ada (Pack_Spec);
  Diana_Object_Editor                  : Ada (Pack_Spec);
  Directory_Tools                      : Ada (Pack_Spec);
  Disk_Daemon                          : Ada (Pack_Spec);
  Dtia_Rpc_Mechanisms'Spec_View.Units : Library (Directory); -- not documented
    .Remote_Shell                      : Library (Directory); -- not documented
    .Sun_Rpc                           : Library (Directory); -- not documented
    .Target_Interface                  : Library (Directory); -- not documented
    .Transfer_Utilities                : Library (Directory); -- not documented
    .Utilities                         : Library (Directory); -- not documented
  Hash                                 : Ada (Pack_Spec);
  Library_Object_Editor                : Ada (Pack_Spec);
  Link_Tools                           : Ada (Pack_Spec);
```





```
  List_Generic                         : Ada (Gen_Pack);
  Lrm                                  : Library (Directory);
    .Ada_Program                       : Ada (Pack_Spec);
    .Associations                      : Ada (Pack_Spec);
    .Compilation_Units                 : Ada (Pack_Spec);
    .Declarations                      : Ada (Pack_Spec);
    .Names_And_Expressions             : Ada (Pack_Spec);
    .Pragmas                           : Ada (Pack_Spec);
    .Representation_Clauses            : Ada (Pack_Spec);
    .Statements                        : Ada (Pack_Spec);
    .Type_Information                  : Ada (Pack_Spec);
  Map_Generic                          : Ada (Gen_Pack);
  Math_Support                         : Library (Subsystem); -- not documented
  Networking                           : Library (Directory);
    .Byte_Defs                         : Ada (Pack_Spec);
    .Byte_String_Io                    : Ada (Pack_Spec);
    .File_Transfer                     : Ada (Pack_Spec);
    .Ftp_Defs                          : Ada (Pack_Spec);
    .Ftp_Name_Map                      : Ada (Pack_Spec);
    .Ftp_Product                       : Ada (Pack_Spec);
    .Ftp_Profile                       : Ada (Pack_Spec);
    .Ftp_Server                        : Ada (Pack_Spec);
    .Host_Id_Io                        : Ada (Pack_Spec);
    .Interchange                       : Ada (Pack_Spec);
    .Interchange_Defs                  : Ada (Pack_Spec);
    .Network_Product                   : Ada (Pack_Spec);
    .Rpc                               : Ada (Pack_Spec);
    .Rpc_Access_Utilities              : Ada (Pack_Spec);
    .Rpc_Client                        : Ada (Pack_Spec);
    .Rpc_Product                       : Ada (Pack_Spec);
    .Rpc_Server                        : Ada (Pack_Spec);
    .Show_Trace                        : Ada (Load_Proc);
    .Tcp_Ip_Boot                       : Ada (Proc_Spec);
    .Tcp_Ip_Dump                       : Ada (Proc_Spec);
    .Telnet_Product                    : Ada (Pack_Spec);
    .Telnet_Profile                    : Ada (Pack_Spec);
    .Telnet_Tools'Spec_View.Units      : Library (Directory);
      .Telnet_Port                     : Ada (Pack_Spec);
      .Telnet_Protocol                 : Ada (Pack_Spec);
      .Telnet_Tools                    : Ada (Pack_Spec);
    .Transfer_Generic                  : Ada (Gen_Pack);
    .Transport                         : Ada (Pack_Spec);
    .Transport_Defs                    : Ada (Pack_Spec);
    .Transport_Interchange             : Ada (Pack_Inst);
    .Transport_Name                    : Ada (Pack_Spec);
    .Transport_Server                  : Ada (Gen_Pack);
    .Transport_Server_Job              : Ada (Pack_Spec);
    .Transport_Stream                  : Ada (Pack_Spec);
  Object_Editor                        : Ada (Pack_Spec);
  Parameter_Parser                     : Ada (Gen_Pack);
```

```
Profile                              : Ada (Pack_Spec);
Program_Library'Spec_View.Units      : Library (Directory);
  .Program_Library_Maintenance       : Ada (Pack_Spec);
Program_Library_Object_Editor        : Ada (Pack_Spec);
Queue_Generic                        : Ada (Gen_Pack);
Random                               : Ada (Pack_Spec);
Rpc'Spec_View.Units                  : Library (Directory);
  .Interchange                       : Ada (Pack_Spec);     -- see networking
  .Interchange_Defs                  : Ada (Pack_Spec);     -- see networking
  .Rpc                               : Ada (Pack_Spec);     -- see networking
  .Rpc_Client                        : Ada (Pack_Spec);     -- see networking
  .Rpc_Product                       : Ada (Pack_Spec);     -- see networking
  .Rpc_Server                        : Ada (Pack_Spec);     -- see networking
  .Transport_Interchange             : Ada (Pack_Inst);     -- see networking
  .Transport_Server                  : Ada (Gen_Pack);      -- see networking
  .Transport_Server_Job              : Ada (Pack_Spec);     -- see networking
  .Transport_Server_Proc             : Ada (Gen_Proc);
  .Transport_Stream                  : Ada (Pack_Spec);     -- see networking
Rpc_Servers'Spec_View.Units          : Library (Directory);
  .Queue_Service                     : Ada (Pack_Spec);
Script                               : Ada (Pack_Spec);
Set_Generic                          : Ada (Gen_Pack);
Simple_Status                        : Ada (Pack_Spec);
Stack_Generic                        : Ada (Gen_Pack);
String_Map_Generic                   : Ada (Gen_Pack);
String_Table                         : Ada (Pack_Spec);
String_Utilities                     : Ada (Pack_Spec);
System_Availability'Spec_View.Units  : Library (Directory);
  .Accounting_Information            : Ada (Pack_Spec);
  .Accounting_Report                 : Ada (Proc_Spec);
  .Data                              : Library (Directory);
  .Log_Reader                        : Ada (Pack_Spec);
  .Outage_Information                : Ada (Pack_Spec);
  .Reports                           : Library (Directory);
    .Availability_Report             : Ada (Proc_Spec);
    .Display_Large_Error_Logs        : Ada (Proc_Spec);
    .Error_Summary                   : Ada (Proc_Spec);
    .Error_Summary_Server            : Ada (Proc_Spec);
    .Pm_Report                       : Ada (Proc_Spec);
  .Show_Error_Log                    : Ada (Proc_Spec);
  .System_Information                : Ada (Pack_Spec);
  .System_Report                     : Ada (Pack_Spec);
  .Utilities                         : Library (Directory);
System_Utilities                     : Ada (Pack_Spec);
Table_Formatter                      : Ada (Gen_Pack);
Table_Sort_Generic                   : Ada (Gen_Proc);
Tape_Tools                           : Ada (Pack_Spec);
Time_Utilities                       : Ada (Pack_Spec);
Unbounded_String                     : Ada (Gen_Pack);
Unchecked_Conversions                : Ada (Pack_Spec);
```





```
Xref_Utility'Spec_View.Units         : Library (Directory);
  .Commands.Xref                     : Ada (Pack_Spec);
```

```
with Action;
with Simple_Status;
with Bounded_String;
with Directory;

with Machine;

package Access_List_Tools is

    subtype Name is String;  -- an object name

    subtype Access_Class is String;  -- of only the following characters:
    Read   : constant Character := 'R';  -- objects and worlds
    Write  : constant Character := 'W';  -- objects only
    Delete : constant Character := 'D';  -- worlds only; same bit as W
    Create : constant Character := 'C';  -- worlds only
    Owner  : constant Character := 'O';  -- worlds only

    -- An object string name is as defined by the directory
    -- package.  No wilcards are accepted;  each operation in this
    -- package operates on one object.

    subtype Acl is String;
    Max_Acl_Length : constant := 512;  -- max length for access list string
    -- The max size will not be exceeded when an Acl is returned.

    -- String representations of access lists have the following syntax:
    --  Acl       ::= Acl_Entry [',' Acl_Entry]*
    --  Acl_Entry ::= Group '=>' Access
    --  Group     ::= Identifier
    --  Access    ::= Acc_Type+
    --  Acc_Type  ::= 'R' | 'W' | 'D' | 'C' | 'O' |
    --                'r' | 'w' | 'd' | 'c' | 'o'
    --  Examples:  "Phil => R , TRW => rw",  "Public=>RCOD"

    Access_Tools_Error : exception;  -- Raised by functions

    function  Get (For_Object : Name)              return Acl;
    function  Get (For_Object : Directory.Version) return Acl;
    procedure Get (For_Object :        Name;
                   List       : out    Bounded_String.Variable_String;
                   Status     : in out Simple_Status.Condition);
    procedure Get (For_Object :        Directory.Version;
                   List       : out    Bounded_String.Variable_String;
                   Status     : in out Simple_Status.Condition);
```





```
    procedure Set (For_Object :        Name;
                   To_List    :        Acl;
                   Status     : in out Simple_Status.Condition);
    procedure Set (For_Object :        Directory.Version;
                   To_List    :        Acl;
                   Status     : in out Simple_Status.Condition);

    -- Get or Set the access list for the specified object.
    -- Setting the access list requires "Owner" access.
    -- function Get raises Access_Tools_Error if an error occurs.
    -- The procedure version should be called in that case to get the
    -- actual error information.
    -- ACL for world must be contain only R, C, O, or D access.  Others
    -- must be only R or W access.

    function Check (User_Name : String := "";
                    Object_Id : Directory.Version;
                    Desired   : Access_Class)   return Boolean;
    function Check (User_Name   : String := "";
                    Object_Name : String;
                    Desired     : Access_Class) return Boolean;
    function Check (User_Id   : Directory.Version;
                    Object_Id : Directory.Version;
                    Desired   : Access_Class)   return Boolean;
    function Check (Job       : Machine.Job_Id;
                    Object_Id : Directory.Version;
                    Desired   : Access_Class)   return Boolean;

    -- Check if the specified user has the indicated access to the
    -- specified object.  Only meaningful for Ada objects, Files, and Worlds.
    -- The null string for the User_Name parameter means the identity of
    -- the calling job.  If a user name is specified, the access control
    -- identity of that user (its member groups) is used for the test.
    -- If an error is detected during the test, the value false is returned.
    -- The most common errors are illegal values for Desired and references
    -- to objects that do not exist.  If an object that does not have an
    -- access list is referenced, the value true is returned.

    function  Get_Default (For_World : Name) return Acl;
    procedure Get_Default (For_World :        Name;
                           List      : out    Bounded_String.Variable_String;
                           Status    : in out Simple_Status.Condition);
    procedure Set_Default (For_World :        Name;
                           To_List   :        Acl;
                           Status    : in out Simple_Status.Condition);
    -- Get or set the default ACL for new objects created in the specified
    -- world.  The function raises the exception Access_Tools_Error if
    -- an error is detected.  The procedure version returns a status
    -- that indicates the cause of the error.
```

```
procedure Check_Validity (For_List :        Acl;
                          Status   : in out Simple_Status.Condition);
-- Check the validity of the specified access list.  Return status
-- indicating that it is okay, or the error, if any.

function Has_Operator_Capability return Boolean;
-- Return true if the calling job has operator capability.  This is
-- true if the job has an identity that includes the group
-- "operator", is on the access list for "!machine.operator_capability",
-- or is priviledged.

function Normalize (Initial_Acl : Acl) return Acl;
-- Scan the acl and eliminate any entries for groups that do
-- not currently exist.  Return the revised acl.  If the
-- acl is otherwise illegal, raise Access_Tools_Error.

function Amend (Initial_Acl : Acl; New_Group : Name; Desired : Access_Class)
               return Acl;
-- Amend Initial_Acl so that New_Group is granted Desired access.  If
-- necessary, the right-most acl entry is removed to do this.
-- Raise Access_Tools_Error if any parameter is illegal.

function Grants (For_List  : Acl;
                 Desired   : Access_Class;
                 User_Name : String := "") return Boolean;

-- Raise Access_Tools_Error if For_List, Desired, or User_Name are
-- illegal. Return true or false depending on whether For_List grants
-- User_Name Desired access.  User_Name ="" (the default) performs the
-- check for the identity of the calling job.

procedure Set (For_Object :        Directory.Version;
               To_List    :        Acl;
               Status     : in out Simple_Status.Condition;
               Action_Id  :        Action.Id);

procedure Set_Default (For_World :        Name;
                       To_List   :        Acl;
                       Status    : in out Simple_Status.Condition;
                       Action_Id :        Action.Id);
-- same as above with an action_id parameter.

procedure Remove (Group       :     String;
                  Initial_Acl :     Bounded_String.Variable_String;
                  New_Acl     : out Bounded_String.Variable_String;
                  Group_Found : out Boolean);
--
-- Removes the groups's entry from the specified access list
```





```
function Is_Valid_Group (Name : String) return Boolean;

function Is_In_Group (User_Name  : String;
                      Group_Name : String) return Boolean;
-- Return true if the User_Name is a member of Group_Name

end Access_List_Tools;
```

```
with Directory;
with Diana;

package Ada_Object_Editor is
    Lock_Error, Undefined : exception;
    -- Lock_Error will be raised if the designated tree is open for
    -- update by the editor.
    -- Undefined exception will be raised if the designated object
    -- does not exist.

    function Current_Image     return Diana.Tree;
    function Current_Selection return Diana.Tree;
    -- Both functions return the appropriate tree with access mode none.
    -- Lock_Error and Undefined may be raised.

    procedure Display (Tree : Diana.Tree);
    -- Create a window displaying the given tree. If this tree is already
    -- displayed on an existing window this window will become the
    -- current focus. If a new window is created it will be designated
    -- read-only.

    type Window_State      is (Normal, Promoted, Demoted);
    type Selection_Request is (Must_Select, Dont_Select, Try_Select);
    type Display_Status    is (Successful, Locked_Out, Illegal_Access,
                               Cannot_Select, Nonexistent_Tree, Unknown_Error);

    procedure Display (Tree              :     Diana.Tree;
                       Status            : out Display_Status;
                       Selection_Command :     Selection_Request := Try_Select;
                       State_Of_Window   :     Window_State      := Normal);
    -- Displays the tree.  State_Of_Window indicates if the window should
    -- appear in its current state, or be promoted, or be demoted.  The
    -- Selection_Command parameter controls as follows:
    --    Must_Select:  Fails if selection is not possible
    --    Dont_Select:  Does no selection
    --    Try_Select :  Tries to select; on failure brings up window anyway

    function Image_Name return String;
    function Selection_Name
                (From_Current_Image_Only : Boolean := True) return String;
    -- Functions to return a directory name for the image or the selection.
    -- Both functions return "###Unknown###" if unable to get a name from
    -- from the directory.  Selection_Name returns "###No_Selection###"
    -- if no selection is found on the current or any image (as designated
    -- by the parameter).  No exceptions come out of these functions.
end Ada_Object_Editor;
```





```
with Diana;
with Directory;
with Directory_Tools;

package Ada_Text is

    -- Ada objects have two components: a Diana tree and a textual image.
    -- This package exports a mapping between the tree and the image, and
    -- provides read access to the image.

    type Handle is private;
    Nil_Handle : constant Handle;


    procedure Open (Ada_Object :     Directory_Tools.Object.Handle;
                    Unit       : out Handle;
                    Status     : out Directory_Tools.Object.Error_Code);

    procedure Open (Ada_Object :     Directory.Version;
                    Unit       : out Handle;
                    Status     : out Directory.Error_Status);

    -- Open acquires a read lock on both the tree and the image, returning
    -- a handle that can be used to make further queries.  Nil_Handle is
    -- returned if the Open fails.


    procedure Close (Unit   : in out Handle;
                     Status : out    Directory_Tools.Object.Error_Code);

    procedure Close (Unit : in out Handle; Status : out Directory.Error_Status);

    -- Close releases the locks acquired by Open and sets Unit to
    -- Nil_Handle.  Using a handle or iterators obtained from it after the
    -- handle has been closed will have unpredictable erroneous results.


    function Root (Unit : Handle) return Diana.Tree;

    -- Returns the root of the unit; returns Diana.Empty if the
    -- opened Ada unit is in archive state.


    type Area is
        record
            First_Line   : Positive;
            First_Column : Positive;
```

```
        Last_Line    : Natural;
        Last_Column  : Natural;
    end record;

Nil_Area : constant Area := (1, 1, 0, 0);
-- An area indicates a stream of contiguous characters on the screen.
-- First_Line and First_Column are the coordinates of the first
-- character of the stream; Last_Line and Last_Column are the
-- coordinates of the last character in the stream.  Lines and
-- columns are numbered beginning from 1.


function Is_Empty (Where : Area) return Boolean;
-- An area A is considered empty iff A.Last_Line < A.First_Line or
-- A.First_Line = A.Last_Line and then A.Last_Column < A.First_Column.


function Entire (Unit : Handle) return Area;
-- Returns the area corresponding to the entire image.


function Has_Partial_Lines (Subtree : Diana.Tree) return Boolean;
-- Indicates whether the image of a subtree uses an integral number
-- of lines (such as a statement), or whether it may start or end
-- in the middle of a line (as an expression).  If this function
-- returns True for some node, it will also return True for all its
-- children (even for a child whose image happens to occupy an integral
-- number of lines.)


function Where_Is (Subtree          : Diana.Tree;
                   Unit             : Handle;
                   And_Post_Comment : Boolean := True) return Area;
-- Returns the area in the image that corresponds to some subtree.
-- If Has_Partial_Lines (Subtree) is true, then the area returned will
-- not included any leading blanks.  If Has_Partial_Lines (Subtree) is
-- False, then the area returned will begin in column 1.  The area
-- returned will include a trailing Same_Line comment (see below) if
-- one is present and the And_Post_Comment parameter is true.


function What_Line (Subtree : Diana.Tree; Unit : Handle) return Natural;
-- Returns 0 for Diana.Empty, and the line number the subtree begins
-- on for nonempty trees.


function What_Is (Where_Is : Area; Unit : Handle) return Diana.Tree;
-- Returns the smallest subtree whose image contains the area.
```





```
function What_Is (On_Line : Positive; Unit : Handle) return Diana.Tree;
-- Returns the smallest subtree T such that Has_Partial_Lines (T) is
-- False and the image of T contains line On_Line.


function What_Statement
           (On_Line : Positive; Unit : Handle) return Diana.Tree;
-- Similar to What_Is, but has a slightly coarser granularity.
-- What_Statement will always return a tree of class DECL or STM, or
-- an ancestor of such a tree.


type Comment_Kind is (Same_Line, Own_Line, Both);

-- A Same_Line comment begins on the same line as an Ada token.
-- Subsequent comment lines are considered to be a continuation of the
-- Same_Line comment, as long as their double-dash delimiter is in the
-- same column as the initial comment.  There may be intervening empty
-- lines as long as they are followed by another properly aligned comment
-- line.

-- An Own_Line comment consists of lines that contain only comments (no
-- Ada tokens).

-- The collection of comments and white space between two Ada tokens
-- are considered by this package to be either a Same_Line comment, an
-- Own_Line comment, or a Same_Line comment followed by an Own_Line
-- comment.

-- The Comment_Kind Both refers to either kind of comment if only one
-- is present, or their concatentation if both are present.


-- Examples:
--
--     A := 0; -- A single-line Same_Line comment
--     B := 0; -- This Same_Line comment
--             -- contains two lines.
--     C := 0;
--     -- These lines make up
--     -- an own-line comment.
--     D := 0;
--
--     -- The blanks lines before, after, and between these lines
--
--     -- are all part of the own-line comment
--
--     E := 0;
--
```

```
--      F := 0;     -- The blank line before this line is considered
--                  -- to be an Own-line comment.
--      G := 0;     -- A same-line comment
--      -- Followed by an Own-line comment
--      H := 0;     -- Same-line comments may have blank lines
--
--                  -- Embedded within them,
--
--      -- but the blank line that precedes this one is part of
--      -- the Own_Line comment.
--      I := 0;


function Pre_Comment (Tree : Diana.Tree; Kind : Comment_Kind; Unit : Handle)
                     return Area;

function Post_Comment
            (Tree : Diana.Tree; Kind : Comment_Kind; Unit : Handle)
            return Area;

-- These functions examine the comment text before the first token
-- or after the last token of program text corresponding to the given
-- Diana tree.  If there is a comment there that matches the Kind
-- parameter, then the Area for that comment is returned; otherwise
-- Nil_Area is returned.

-- In general, the same comment can be returned for more than one tree.
-- For example, in:
--
--      A.B := 1;
--      -- comment
--      C.D := 2;
--
-- the comment can be returned as a post-comment of the first Dn_Assign
-- node, or as a pre-comment on the second Dn_Assign node, the
-- Dn_Selected node for C.D or the Dn_Used_Name_Id node for C.

-- If a piece of a comment that matches the Kind parameter, just the
-- piece will be returned.  For example, in:
--
--      P; -- comment 1
--      -- comment 2
--      Q;
--
-- the first comment will be returned as the Same_Line post-comment of
-- P and the Same_Line pre-comment of Q.  Likewise, the second comment
-- be returned as the Own_Line post-comment of P and the Own_Line
-- pre-comment of Q.  The value of Same_Line pre-comments is dubious,
-- but they have been provided for the sake of completeness.
```





```
type Iterator is private;
Nil_Iterator : constant Iterator;


procedure Initialize (Unit : Handle; Where : Area; Iter : out Iterator);

function  Done (Iter : Iterator) return Boolean;
procedure Next (Iter : in out Iterator);

-- Iterators can be used to retreive the text that is in some area
-- of an image.  An iterator will return a sequence of strings;
-- one string for each line in the area.  For an area A,
-- if A.Last_Line < A.First_Line, then no strings will be returned.
-- Otherwise, A.Last_Line - A.First_Line + 1 strings will be returned.
-- The first string returned will be truncated so that characters
-- before (A.First_Line, A.Last_Line) will not be returned.  The
-- last string returned will be truncated so that characters after
-- (A.Last_Line, A.Last_Column) will not be returned.


function Value          (Iter : Iterator) return String;
function Leading_Blanks (Iter : Iterator) return Natural;
function Nonblank_Value (Iter : Iterator) return String;

-- Most of the strings returned by the iterator will begin with
-- some leading blanks.  The Value function returns the string
-- with its leading blanks.  Alternatively, the Leading_Blanks and
-- Nonblank_Value can be used to get these values separately.  These
-- functions obey the identity:
--
--    Value (I) = String'(1..Leading_Blanks (I) => ' ') &
--                Nonblank_Value(I)
--
-- Warning: the string values returned will be a slice of some internal
-- buffer, so that the numeric values of the lower and upper bounds
-- will not have any meaningful value.

-- If the iterator is not convenient, the following functions may
-- be used examine the image.

function Number_Of_Lines (Unit : Handle) return Natural;

function Get_Line (Line : Positive; Unit : Handle) return String;
-- Returns the null string if Line is out of bounds.
-- The lower bound of the returned string will be 1.

function Line_Length (Line : Positive; Unit : Handle) return Natural;
-- Returns zero if Line is out of bounds.
```

```
    function Get_Character (Line : Positive; Column : Positive; Unit : Handle)
                           return Character;
    -- Returns space if Line or Column is out of bounds.


private
    type Open_State;
    type Handle is access Open_State;
    pragma Segmented_Heap (Handle);
    Nil_Handle : constant Handle := null;

    type Iterator_State;
    type Iterator is access Iterator_State;
    pragma Segmented_Heap (Iterator);
    Nil_Iterator : constant Iterator := null;

end Ada_Text;
```





```
generic
    type Object is limited private;
    type Name   is access Object;

function Allows_Deallocation return Boolean;
```

```
package Bit_Operations is

    -- Operations on Integer and Long_Integer as an array of bits.
    -- Bit numbering is left to right, 0..31 and 0..63.
    -- Arguments specified outside bit range raise Constraint_Error

    -- Extract the bits from Start .. Start + Length - 1.  Equivalent to
    -- Logical_And with 1 in bits of the extraction range and 0 elsewhere.
    function Extract
                (W : Integer; Start : Natural; Length : Natural) return Integer;
    function Extract (W : Long_Integer; Start : Natural; Length : Natural)
                     return Long_Integer;

    -- Extract the single bit at B and return as Boolean
    -- Equivalent to Extract (W, B, 1) /= 0
    function Test_Bit (W : Integer; B : Natural)      return Boolean;
    function Test_Bit (W : Long_Integer; B : Natural) return Boolean;

    -- Replace the specified bits of Into with the rightmost Length bits of W.
    function Insert (W, Into : Integer; Start : Natural; Length : Natural)
                    return Integer;
    function Insert (W, Into : Long_Integer; Start : Natural; Length : Natural)
                    return Long_Integer;

    -- Set the specified Bit to One or Zero
    procedure Set_Bit_To_One  (W : in out Integer; B : Natural);
    procedure Set_Bit_To_One  (W : in out Long_Integer; B : Natural);
    procedure Set_Bit_To_Zero (W : in out Integer; B : Natural);
    procedure Set_Bit_To_Zero (W : in out Long_Integer; B : Natural);

    -- Logical operations on two operands.
    -- For shift operations, positive arguments shift Left and negative
    -- arguments shift Right.
    -- Intermediate results are stored as Long_Integer, so shifting
    -- ones into positions outside of Integer will raise Numeric_Error.
    function Logical_And   (X, Y : Integer)                return Integer;
    function Logical_And   (X, Y : Long_Integer)           return Long_Integer;
    function Logical_Or    (X, Y : Integer)                return Integer;
    function Logical_Or    (X, Y : Long_Integer)           return Long_Integer;
    function Logical_Xor   (X, Y : Integer)                return Integer;
    function Logical_Xor   (X, Y : Long_Integer)           return Long_Integer;
    function Logical_Not   (X : Integer)                   return Integer;
    function Logical_Not   (X : Long_Integer)              return Long_Integer;
    function Logical_Shift (X : Integer; Amount : Integer) return Integer;
    function Logical_Shift
                (X : Long_Integer; Amount : Integer)       return Long_Integer;
end Bit_Operations;
```





```
package Bounded_String is

    subtype String_Length is Natural;

    type Variable_String (Maximum_Length : String_Length) is private;
    -- initialized to have a length of 0

    procedure Copy (Target : in out Variable_String; Source : Variable_String);
    procedure Copy (Target : in out Variable_String; Source : String);
    procedure Copy (Target : in out Variable_String; Source : Character);

    procedure Move (Target : in out Variable_String;
                    Source : in out Variable_String);

    function Image (V : Variable_String) return String;

    -- Value function with maximum length = current length
    function Value (S : String) return Variable_String;

    -- Value function with specified maximum length
    function Value (S : String; Max_Length : Natural) return Variable_String;

    procedure Free (V : in out Variable_String);

    procedure Append (Target : in out Variable_String;
                      Source :        Variable_String);
    procedure Append (Target : in out Variable_String; Source : String);
    procedure Append (Target : in out Variable_String; Source : Character);
    procedure Append (Target : in out Variable_String;
                      Source :        Character;
                      Count  :        Natural);

    procedure Insert (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Source :        Variable_String);
    procedure Insert (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Source :        String);
    procedure Insert (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Source :        Character);
    procedure Insert (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Source :        Character;
                      Count  :        Natural);
```

```
    procedure Delete (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Count  :        Natural := 1);

    procedure Replace (Target : in out Variable_String;
                       At_Pos :        Positive;
                       Source :        Character);
    procedure Replace (Target : in out Variable_String;
                       At_Pos :        Positive;
                       Source :        Character;
                       Count  :        Natural);
    procedure Replace (Target : in out Variable_String;
                       At_Pos :        Positive;
                       Source :        String);
    procedure Replace (Target : in out Variable_String;
                       At_Pos :        Positive;
                       Source :        Variable_String);

    procedure Set_Length (Target     : in out Variable_String;
                          New_Length :        Natural;
                          Fill_With  :        Character := ' ');
    -- Truncate or extend with fill

    function Length (Source : Variable_String) return Natural;
    -- Get information about or contents of a string

    function Char_At (Source : Variable_String; At_Pos : Positive)
                 return Character;

    function Extract (Source    : Variable_String;
                      Start_Pos : Positive;
                      End_Pos   : Natural) return String;

    function Max_Length (Source : Variable_String) return Natural;
    -- get the allocated length of the string

private
    type Variable_String (Maximum_Length : String_Length) is
        record
            Length   : String_Length := 0;
            Contents : String (1 .. Maximum_Length);
        end record;

end Bounded_String;
```





```
package Ci is

    procedure Login (Terminal : String   := "!machine.devices.terminal_1";
                     Timeout  : Duration := 60.0);
    -- Prompt for username/password.  If they are OK, prompte for session,
    -- and start a new job running Interpret (below) on that session.
    -- When the interpreter job terminates, repeat the process.

    procedure Interpret (Terminal : String   := "!machine.devices.terminal_1";
                         Timeout  : Duration := 60.0);
    -- Prompt for and execute environment commands.
    -- Return when the command "quit" is entered, or
    -- the specified Timeout elapses with no input.

end Ci;
```

```
with Terminal;

package Run is

    procedure Full_Backup;
    procedure Primary_Backup;
    procedure Secondary_Backup;

    procedure Save (Object : String);
    procedure Restore;

    procedure Disable_Queue  (Device : String);
    procedure Enable_Queue   (Device : String := "all");
    procedure Cancel_Request (Id : Natural);

    procedure Schedule_Shutdown (At_Time : String := "23:59");
    procedure Cancel_Shutdown;

    procedure Disable_Login (Line : Terminal.Port);
    procedure Enable_Login  (Line : Terminal.Port);
    procedure Force_Logoff  (Line : Terminal.Port);

    procedure Send_Message (M : String; User : String := "<ALL>");
    procedure Set_Time     (T : String);
    procedure Snapshot;

end Run;
```





```
with Terminal;

package Show is

    procedure Backups  (Last : Positive := 10);
    procedure Daily_Message;
    procedure Disk_Space;
    procedure Jobs;
    procedure Queues   (Class : String := "all");
    procedure Requests (Class : String := "all");
    procedure Shutdown;
    procedure Settings (Line : Terminal.Port := 0);
    procedure Time;
    procedure Users;
    procedure Help;

end Show;
```

```
procedure Typ (Name : String := "[]");
```

```
package Code_Segment_Object_Editor is

    procedure Edit (The_Segment : Long_Integer;
                    The_Offset  : Integer := 0;
                    In_Place    : Boolean := False);

    procedure Edit (The_Unit         : String  := "<SELECTION>";
                    In_Place         : Boolean := False;
                    Elaboration_Code : Boolean := False);

end Code_Segment_Object_Editor;
```





```
package Cdb_Maintenance is

    procedure Verify (Subsystems       : String  := "<SELECTION>";
                      Verify_Cdb       : Boolean := True;
                      Verify_Ada_Units : Boolean := True;
                      Response         : String  := "<PROFILE> ~$$$");

    function Verify (Subsystems       : String  := "<SELECTION>";
                     Verify_Cdb       : Boolean := True;
                     Verify_Ada_Units : Boolean := True;
                     Response         : String  := "<QUIET>") return Boolean;

    -- Verify that the CDBs for a set of subsystems are internally
    -- consistent and that the ada units in the subsystems are
    -- consistent with the compatibility databases.  If there are
    -- errors in the CDB itself, then the CDB must be destroyed; if
    -- there are errors in ada units, the units must be demoted to
    -- source and recompiled.  The functional form returns TRUE
    -- if the CDB is valid and the ada units are consistent with
    -- the CDB.


    procedure Compare (Primary_Subsystem   : String := "<REGION>";
                       Secondary_Subsystem : String := "<IMAGE>";
                       Response            : String := "<PROFILE> ~$$$");

    function Compare (Primary_Subsystem   : String := "<REGION>";
                      Secondary_Subsystem : String := "<IMAGE>";
                      Response            : String := "<QUIET>") return Boolean;

    -- Check that the CDB for a secondary copy of a subsystem is
    -- consistent with that in the primary copy.  The CDBs for a
    -- secondary copy of a subsystem will be inconsistent with that
    -- of the primary copy when there has been unsynchronized
    -- development in the secondary copy (i.e., it was temporarily
    -- made into a primary copy.  This command only checks that the
    -- CDB for the secondary copy is consistent with that of the
    -- primary copy, it does not check that the primary copy is
    -- internally consistent; nor does it check that the ada units
    -- are consistent with the CDBs (both of which can be checked
    -- using the Verify command).  The functional form returns
    -- TRUE if the secondary copy is consistent with the primary.
```

```
    procedure Destroy (Subsystems : String := "<SELECTION>";
                       Response   : String := "<PROFILE>");

    -- Destroy the compatibility database for a subsystem.  All
    -- ada units in the subsystem will be demoted to source and
    -- all code views in the subsystem will be destroyed.  When
    -- destroying the CDB for a primary subsystem, the CDB must
    -- be destroyed in all secondary copies (on all machines).
    -- After destroying the CDB in a secondary subsystem, the
    -- CDB should be rebuilt by updating it from the primary
    -- copy (using the Update command).


    procedure Update (Subsystem    : String := "<SELECTION>";
                      From_Primary : String := "<ASSOCIATED_PRIMARY>";
                      Response     : String := "<PROFILE>");

    -- Updates the CDB for a secondary copy of a subsystem
    -- from the CDB in the primary copy, provided that the
    -- primary copy resides on the local machine or on a
    -- machine that is reachable over the network.


    procedure Display (Subsystems                 : String  := "<SELECTION>";
                       Unit_Maps                  : Boolean := True;
                       Declaration_Maps           : Boolean := False;
                       Offset_Maps                : Boolean := False;
                       Compatibility_Signatures   : Boolean := False;
                       Response                   : String  := "<PROFILE>");

    -- Display the contents of the CDB for a subsystem.  If the
    -- SUBSYSTEMS parameter specifies the names of ada units, then
    -- the maps and signatures for only those units will be displayed.


    procedure Convert (Subsystems : String := "<SELECTION>";
                       Response   : String := "<PROFILE>");

    -- Convert the CDB from the Delta-0 format to the Delta-1 format.
    -- This command MUST be run on each subsystem before performing
    -- ANY compilation in the subsystem.  The command can be run more
    -- than once on a subsystems with no ill effects.  This command
    -- does not perform a verification of the CDB; corrupted CDBs
    -- will be converted (possibly incorrectly).

end Cdb_Maintenance;
```





```
package Check is

    -- Check that units in a load view are compatible with the
    -- corresponding units in a spec view.  A unit in a load view
    -- is compatible with a spec view unit if every declaration
    -- exported by the spec view unit is also exported by the
    -- load view unit.  The declarations need not be in the same
    -- order in the two units nor do they need to be textually
    -- identical (two declarations are considered equaivalent
    -- if they match according to the subprogram specification
    -- conformance rules of LRM 6.3.1).  In addition, units
    -- must be (at least) installed and reside in worlds with
    -- compatible target keys.  For units in the coded state,
    -- a check is made that they were coded using compatible
    -- versions of the code generator.

    type Status is (Compatible, Incompatible, Error);


    procedure Units (Load_View_Units : String  := "<CURSOR>";
                     Spec_Views      : String  := "<ACTIVITY>";
                     Menu            : Boolean := False;
                     Response        : String  := "<PROFILE>");

    function Units (Load_View_Units : String := "<CURSOR>";
                    Spec_Views      : String := "<ACTIVITY>";
                    Response        : String := "<PROFILE>") return Status;

    -- For each load view unit, check its compatibility with its
    -- corresponding spec view unit.  The set of corresponding
    -- spec views may be given explicitly using a naming expression
    -- that resolves to a set of views, or may be given implicitly
    -- as a set of spec views in some activity.  In either case,
    -- there must be exactly one spec view for each load view.

    -- The results of the compatibility check may be reported
    -- in the log file or as a menu of the spec view units that
    -- were incompatible with their corresponding load view units.
    -- When a unit on the menu is visited, the incompatible
    -- declarations will be underlined.


    procedure Views (Load_Views : String  := "<CURSOR>";
                     Spec_Views : String  := "<ACTIVITY>";
                     Menu       : Boolean := False;
                     Response   : String  := "<PROFILE>");
```

```
    function Views (Load_Views : String := "<CURSOR>";
                    Spec_Views : String := "<ACTIVITY>";
                    Response   : String := "<PROFILE>") return Status;

    -- For each load view, find the corresponding spec view and
    -- check all of the exported ada specs for compatibility.
    -- The set of corresponding spec views may be given explicitly
    -- using a naming expression that resolves to a set of views,
    -- or may be given implicitly as the set of spec views in some
    -- activity.  In either case, there must be exactly one spec
    -- view for each load view.


    procedure Activity (The_Activity : String  := "<ACTIVITY>";
                        Menu         : Boolean := False;
                        Response     : String  := "<PROFILE>");

    function Activity (The_Activity : String := "<ACTIVITY>";
                       Response     : String := "<PROFILE>") return Status;

    -- Check that each spec view in the specified activity is
    -- compatible with its corresponding load view.


    procedure Code_Views (Code_Views : String := "<CURSOR>";
                          Response   : String := "<PROFILE>");

    function Code_Views (Code_Views : String := "<CURSOR>";
                         Response   : String := "<PROFILE>") return Status;

    -- For each code view, check that each spec view unit referenced
    -- when the code view was created still exists and still exports
    -- the entities used by the code view.  The naming expression specifying
    -- the Code_Views may resolve to any type of view.  Those that are
    -- not actually code views will be ignored.  In particular,
    -- Code_Views may be specified indirectly by an activity, in which
    -- case all code views referenced by that activity will be checked.

    -- The results of the code view compatibility check are reported
    -- in the log file.

end Check;
```





```
generic
    Size : Integer;
    -- number of buckets

    type Domain_Type is private;
    type Range_Type  is private;
    -- both types are pure values
    -- no initialization or finalization is necessary
    -- = and := can be used for equality and copy

    with function Hash (Key : Domain_Type) return Integer is <>;
    -- for efficiency, spread hash over an interval at least as great as size

    pragma Must_Be_Constrained (Yes => Domain_Type, Range_Type);

package Concurrent_Map_Generic is

    type Map is private;

    type Pair is
        record
            D : Domain_Type;
            R : Range_Type;
        end record;

    function  Eval (The_Map : Map; D : Domain_Type) return Range_Type;
    procedure Find (The_Map :        Map;
                    D       :        Domain_Type;
                    R       : in out Range_Type;
                    Success : out    Boolean);
    procedure Find (The_Map :        Map;
                    D       :        Domain_Type;
                    P       : in out Pair;
                    Success : out    Boolean);

    procedure Define   (The_Map        : in out Map;
                        D              :        Domain_Type;
                        R              :        Range_Type;
                        Trap_Multiples :        Boolean := False);
    procedure Undefine (The_Map : in out Map; D : Domain_Type);

    procedure Initialize (The_Map : out Map);
    function  Is_Empty   (The_Map : Map) return Boolean;
    procedure Make_Empty (The_Map : in out Map);

    procedure Copy (Target : in out Map; Source : Map);
```

```
    type Iterator is private;

    procedure Init  (Iter : out Iterator; The_Map : Map);
    procedure Next  (Iter : in out Iterator);
    function  Value (Iter : Iterator) return Domain_Type;
    function  Done  (Iter : Iterator) return Boolean;

    Undefined : exception;
    -- raised by eval if the domain value in not in the map

    Multiply_Defined : exception;
    -- raised by define if the domain value is already defined and
    -- the trap_multiples flag has been specified (ie. is true)

    function Nil                    return Map;
    function Is_Nil (The_Map : Map) return Boolean;

    function Cardinality (The_Map : Map) return Natural;

    ------------------------------------------------------
    -- Implementation Notes and Non-Standard Operations --
    ------------------------------------------------------

    -- := and = operate on references
    --    := implies sharing (introduces an alias)
    --    = means is the same map, not the same value of type map
    -- Initializing a map also makes it empty

    -- Maps must be initialized before use.

    -- garbage may be generated

    -- Concurrent Properties
    --    any number of find/eval/is_empty/copy may be safely done while one
    --   define/undefine/make_empty is taking place.
    --    Define/undefine/make_empty operations are serialized.
    --    Iterators may be used asynchronously, however the sequence of values
    --   yielded may never have been in the map at any one time.

private
    type Node;
    type Set is access Node;

    type Node is
        record
            Value : Pair;
            Link  : Set;
        end record;
```





```
    type Map_Data;

    type Map is access Map_Data;

    type Iterator is
        record
            The_Map     : Map;
            Index_Value : Natural;
            Set_Iter    : Set;
            Done        : Boolean;
        end record;

    subtype Index is Integer range 0 .. Size - 1;

    type Table is array (Index) of Set;

    type Map_Data is
        record
            Cache  : Set;             -- of at most one node
            Bucket : Table;
            Size   : Natural := 0;
        end record;

end Concurrent_Map_Generic;
```

```
package Debug_Tools is

    procedure Debug_On;
    procedure Debug_Off;

    -- Enable or disable debugging for the calling task's job.  When enabled,
    -- only tasks that are descendents of the caller can be debugged.
    -- When debugging is disabled, the task is released to execute, and all
    -- active debugger "hooks" are deactivated (eg, breakpoints, etc).

    function Debugging return Boolean;

    -- return true if calling task is being debugged.

    procedure Message (Info : String);

    -- Print the message string in the debugger window.  No operation if
    -- the debugger is not activated

    procedure User_Break (Info : String);

    -- "Break" in the debugger.  The calling task stops as though it
    -- encountered a breakpoint.  If the debugger is not active, no action
    -- is performed.  Otherwise, the task remains stopped until the
    -- debugger user explicitly continues its execution.

    procedure Set_Task_Name (Name : String);
    function  Get_Task_Name return String;

    -- Set or retrieve a string "synonym" for the calling task.  This name
    -- is used within the debugger to make identifying task easier.
    -- It is also useful for multiple instances of the same task type
    -- to distinguish themselves in the debugger.
    -- No operation if the debugger is not activate.

    function Ada_Location (Frame_Number  : Natural := 0;
                           Fully_Qualify : Boolean := True;
                           Machine_Info  : Boolean := False) return String;

    -- Return a string name for the Ada location of execution in the
    -- specified stack frame.  Frame_Number = 0 refers to the caller
    -- of Ada_Location.  Frame_Number = 1 refers to its caller, and so
    -- on.  The null string is returned if the frame is nonexistent or
    -- its location cannot be found for some other serious reason.
    -- This procedure works independent of whether there is an active
    -- debugger for the calling tasks, but it may return less information
    -- if there is not.
```





```
    function Get_Exception_Name (Fully_Qualify : Boolean := True;
                                 Machine_Info : Boolean := False) return String;

    -- return a string representation of the exception most recently
    -- executed by the calling task.  Get_Exception_Name must be called
    -- either directly or indirectly from an exception handler.  If
    -- no exception is found, the null string is returned.

    function Get_Raise_Location (Fully_Qualify : Boolean := True;
                                 Machine_Info : Boolean := False) return String;

    -- return a string representation of the location of the exception
    -- most recently executed by the calling task.  Get_Raise_Location
    -- must be called either directly or indirectly from an exception
    -- handler.  If  no exception is found or other problems encountered,
    -- the null string is returned.

    generic
        type T is limited private;
        -- Type that will be displayed using Image procedure as part
        -- of debugger's object display procedure.

        with function Image (Value           : T;
                             Level           : Natural;
                             Prefix          : String;
                             Expand_Pointers : Boolean) return String;

        -- Given the Value, return its image.  Level specifies the number
        -- of levels to detail to be displayed.  Expand_Pointers indicates
        -- that internal pointers should be expanded in the image.
        -- Level = 0 => entire object should be elided.

        -- If any ASCII.LF's are emitted, the following line should
        -- start with Prefix as an "indent" value.

    procedure Register;
    -- Make this special display procedure known to the debugger
    -- of the session making the call.

    generic
        type T is limited private;
    procedure Un_Register;
    -- Remove the display procedure from the calling session's
    -- database of special types for the type T given.

end Debug_Tools;
```

```
with Diana;

package Diana_Object_Editor is

    -----------------------------------------
    -- type safe interfaces for diana types --
    -----------------------------------------
    procedure Edit (Tree : Diana.Tree);
    procedure Edit (Seq_Type : Diana.Seq_Type);
    procedure Edit (Sequence : Diana.Sequence);
    procedure Edit (Temp_Seq : Diana.Temp_Seq);

    ----------------------------------------------------------------------
    -- unsafe interfaces, segment and offset may not be of the right type --
    ----------------------------------------------------------------------
    procedure Edit_Tree     (Segment, Offset : Long_Integer);
    procedure Edit_Seq_Type (Segment, Offset : Long_Integer);
    procedure Edit_Sequence (Segment, Offset : Long_Integer);
    procedure Edit_Temp_Seq (Segment, Offset : Long_Integer);

    --------------------------------------------------------
    -- image functions not provided by r1000 (native) diana --
    --------------------------------------------------------
    function Image (Attr_Name : Diana.Attr_Name) return String;

    -------------------------------
    -- functions to read an image --
    -------------------------------
    function Current_Image     return Diana.Tree;
    function Current_Cursor    return Diana.Tree;
    function Current_Selection return Diana.Tree;

end Diana_Object_Editor;
```





```
with Action;
with Calendar;
with Diana;
with Directory;
with Error_Messages;
with Profile;
with String_Table;

package Directory_Tools is

    -- DIRECTORY TOOLS ORGANIZATION
    -- The Directory system provides the structure for storing, managing
    -- and naming objects.

    -- Each object has a class, which determines the operations that can
    -- be applied to object. Program units belong to the class Ada.
    -- Libraries, the building blocks of the directory system, are
    -- objects of class Library. The class reflects which object manager
    -- manages objects of that type, as well as reflecting the type.

    -- Except for objects of class Library, each Directory Object has
    -- one or more versions, which can be selected by using the
    -- appropriate version name.

    -- Ada.Unit and Polymorphic_IO.File (and others) are ultimately of
    -- type Directory.Object and provide type specific operations. The
    -- general paradigm is that type independent operations (traversal,
    -- create, copy, destroy, etc.) are provided in directory, while
    -- type specific operations are provided by the packages
    -- (Directory.Ada, Polymorphic_io, etc.) which introduce specific
    -- managed types.

    -- No exceptions are propagated from this subsystem, except those
    -- associated with type specific operations.

    -- .  Package Object. Defines the object handle type and the
    -- iterator on object types.

    -- .  Package Naming.  Provides facilities for establishing a context
    -- for name resolution and facilities for resolving string names.

    -- .  Package Traversal.  Operations for traversing the directory
    -- structure (which extends through all Ada units in the system).

    -- .  Package Any_Object.  Standard Directory operations for
    -- Creating, Freezing, Destroying and Copying managed objects in the
    -- Directory.
```

```
-- .  Package Library_Object.  Defines operations specific to
-- objects of class library. Defines a Library object as a
-- distinguished point (World or Directory) in the directory system.
-- A world specifies the disk volume for storing its contents and
-- the policies which will apply to its contents.

-- .  Package Ada_Object.  Defines an Ada Unit as a kind of
-- Directory.Object. Provides type-specific operations for
-- constructing and manipulating Ada Units.

-- .  Package Statistics.  Queries about Directory Objects.
-- This package is the main interface to the directory subsystem.

-- .  Package Object.Low_Level. Defines interface between Object. Handles
-- and the low level types of Directory_Implementation.

-- .  Package Ada_Implementation. Defines operations for gaining
-- access to Diana Trees from Object.Handles;

package Object is
    package Di renames Directory;
    package St renames String_Table;

    -- Herein are defined the principle structures for accessing the
    -- Directory System objects: Object.Handle and Object.Iterator

    type Handle is private;

    -- Objects in the directory system and the Versions of those
    -- Objects are accessed via an Object.Handle. A handle may denote
    -- all Versions of an Object collectively or a specific Version of
    -- an Object. Most operations operate on specific Versions of an
    -- Object. If the Object.Handle passed to the operation denotes no
    -- specific Version, the Default Version is used.

    function Nil return Object.Handle;
    function Is_Nil (The_Object : Object.Handle) return Boolean;

    function Hash   (The_Object : Object.Handle) return Integer;
    function Unique (The_Object : Object.Handle) return Long_Integer;

    function Image (The_Object      : Object.Handle;
                    Level           : Natural;
                    Prefix          : String;
                    Expand_Pointers : Boolean) return String;

    -- See debug_tools.special_display
```





```
    function Version (The_Object : Object.Handle) return String;

    -- Returns a name of the form "V(nn)", for the Version denoted by the
    -- Object.Handle.

    function Same_Object (Left, Right : Object.Handle) return Boolean;
    function Equal       (Left, Right : Object.Handle) return Boolean
        renames Same_Object;

    -- Compare two handles to see if they refer to the same directory
    -- entity.  Please note, "=" will give incorrect results

    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

    type Class_Enumeration is new Natural range 0 .. 63;

    Unknown_Class  : constant Class_Enumeration := 0;
    Library_Class  : constant Class_Enumeration := 1;
    Ada_Class      : constant Class_Enumeration := 2;
    File_Class     : constant Class_Enumeration := 3;
    User_Class     : constant Class_Enumeration := 4;
    Session_Class  : constant Class_Enumeration := 5;
    Pipe_Class     : constant Class_Enumeration := 6;
    Terminal_Class : constant Class_Enumeration := 7;
    Tape_Class     : constant Class_Enumeration := 8;


    function Class (The_Object : Object.Handle) return Class_Enumeration;
    function Equal (Class1, Class2 : Class_Enumeration) return Boolean;
    function Image (The_Class : Class_Enumeration) return String;
    function Value (S : String) return Class_Enumeration;
    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
    type Subclass is private;
    function Nil return Subclass;
    function Is_Nil      (The_Subclass : Subclass) return Boolean;
    function Unique      (The_Subclass : Subclass) return Integer;
    function Subclass_Of (The_Object : Object.Handle) return Subclass;
    function Image       (The_Subclass : Subclass) return String;
    function Value       (S : String) return Subclass;
    function Class_Of    (The_Subclass : Subclass) return Class_Enumeration;

    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

    type Iterator is private;

    -- Representation of an ordered set of Objects (Object.Handles);

    procedure Next  (Iter : in out Object.Iterator);
```

```
function  Done  (Iter : Object.Iterator) return Boolean;
function  Value (Iter : Object.Iterator) return Object.Handle;

procedure Reset (Iter : Object.Iterator);

-- reset the iterator to the beginning of the list.

function Nil return Object.Iterator;
function Is_Nil (Iter : Object.Iterator) return Boolean;
function Image  (The_Iterator    : Object.Iterator;
                 Level           : Natural;
                 Prefix          : String;
                 Expand_Pointers : Boolean) return String;


function Create return Object.Iterator;

-- create a new (empty) iterator.  Note: an empty iterator is different
-- than a 'nil' iterator

function Has (Iter : Object.Iterator; An_Object : Object.Handle)
             return Boolean;

procedure Add (Iter      :     Object.Iterator;
               An_Object :     Object.Handle;
               Duplicate : out Boolean;
               Before    :     Object.Handle := Object.Nil);

-- The given Object is added to the Iterator just before the object
-- denoted by the Before parameter.  If the Before parameter is not
-- found, the object is added at the end of the list of Objects.

procedure Remove (Iter      :     Object.Iterator;
                  An_Object :     Object.Handle;
                  Found     : out Boolean);

-- The specified object is removed from the iterator if it is there.

procedure Invert (Iter : Object.Iterator);
-- reverse the ordering of the given object list.

--------------------------------------------------------------------

-- procedures and functions to handle errors

type Error_Code is private;
-- All procedures return an object.Code, which describes the
-- success or failure of the operation.
```





```
function Err_Code (The_Object : Object.Handle) return Object.Error_Code;
function Err_Code (The_Objects : Object.Iterator)
                  return Object.Error_Code;


-- Object handles and Iterators contains an object.Code for the last
-- operation performed on them.  This error code is the only way
-- problems are reported by the Directory System functions.  This
-- error code is a copy of the error code returned by
-- Directory.System procedures.  Procedures and functions within the
-- Directory System propagate bad error codes from their input to
-- their outputs. Except for this propagation of error code,
-- procedures passed bad objects are no-ops.

function Is_Bad (Error_Code : Object.Error_Code) return Boolean;
function Is_Bad (The_Object : Object.Handle)     return Boolean;
function Is_Bad (The_Objects : Object.Iterator)  return Boolean;

function Is_Ok (Error_Code : Object.Error_Code) return Boolean;
function Is_Ok (The_Object : Object.Handle)     return Boolean;
function Is_Ok (The_Objects : Object.Iterator)  return Boolean;

-- Test the object.Code for Success/Failure status

function Message (Error_Code : Object.Error_Code) return String;
function Message (The_Object : Object.Handle)     return String;
function Message (The_Objects : Object.Iterator)  return String;
function Message (Error_Code : Object.Error_Code) return St.Item;
function Message (The_Object : Object.Handle)     return St.Item;
function Message (The_Objects : Object.Iterator)  return St.Item;

-- Transforms the object.Code into an English explanation of the
-- Error.

procedure Report (Error_Code : Object.Error_Code;
                  Response   : Profile.Response_Profile := Profile.Get);
procedure Report (The_Object : Object.Handle;
                  Response   : Profile.Response_Profile := Profile.Get);
procedure Report (The_Objects : Object.Iterator;
                  Response : Profile.Response_Profile := Profile.Get);

-- If the object.Code is Bad, a message is formulated from the code and
-- sent to the current Log device.  If requested by the given response
-- profile, the exception Failure or Abandon is raised.

-- If the Code is Bad, a message is formulated from the code and sent
-- the current Log device.  If requested by the given response profile,
-- the exception Failure or Abandon is raised.
```

```
Error : exception;

-- Raised by Report in the event of an error when the given Response
-- Profile asks that an exception be propagated to the caller.

Abandon : exception;

-- Raised by Report in the event of an error when the given response
-- Profile askes that the operationbe Abandonded without propagating
-- exceptions to the caller.


-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

-- Depending on the operation that generated a bad error code, additional
-- useful information may be associated with an error code. First, each
-- error code is assigned to one of the following categories:

type Category_Enumeration is
   (Successful,
    -- No problems encountered.
    Warning,
    -- Some non-fatal error.
    Lock_Error,
    -- Some synchronization error occurred,
    -- usually failure to acquire access to some
    -- object within the specified maximum delay
    Semantic_Error,
    -- An operation requiring (Ada) semantic
    -- consistency discovered semantic errors.
    Code_Generation_Error,
    -- An error was detected during cg.
    Obsolescence_Error,
    -- A change was prevented because it
    -- obsolesced installed declarations.
    Bad_Tree_Parameter,
    -- An actual tree parameter failed to meet
    -- the requirements of the formal subtype.
    Illegal_Operation,
    -- The attempted operation is not legal
    -- when applied to the given parameters.
    Consistency_Error,
    -- The operation is inconsistent with the
    --   current state of the universe.
    Version_Error,
    -- The specified version does not exist.
    Policy_Error,
    -- The operation violates some other policy
```





```
    -- that applies at this point.
    Bad_Naming_Context,
    -- The context was not a valid context for
    -- name resolution.
    Ill_Formed_Name,
    -- The name was not well formed lexically or
    -- syntactically.
    Undefined_Name,
    -- The name could not be found in the given
    -- context.
    Ambiguous_Name,
    -- Because of overloading or wildcards, the
    -- name resolved to more than one entity.
    Name_Error,
    -- other errors occured resolving a name.
    Access_Error,
    -- The operation violates access control
    -- policies.
    Class_Error,
    -- The class of the object passed to the
    --   operation is incompatible with op
    --   either because the op expects a
    --   particular class, or because the
    --   op is a type independent op which
    --   is not supported for the given class.

    -- various selection errors
    No_Selection, Cursor_Not_In_Selection,
    Selections_Not_Supported, No_Declaration, No_Object, No_Editor,

    Other_Error);                          -- When all else fails ...

function Category (Error_Code : Object.Error_Code)
                  return Object.Category_Enumeration;

-- Extracts from each error code, the category it belongs to.

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

-- Error codes in the category Semantic_Error and Code_Generation_Error
-- (and perhaps others) may have a list of error messages associated
-- them.

type Message_List is private;

type Severity_Enumeration is (Note, Warning, Lrm_Error,
                              Internal_Error, Exception_Handled);
function Severity (Result : Object.Message_List)
                  return Severity_Enumeration;
```

```
function Message  (Result : Object.Message_List) return String;
function Message  (Result : Object.Message_List) return St.Item;

-- Properties of the current message in the list

function Next (Result : Object.Message_List) return Object.Message_List;
function Done (Result : Object.Message_List) return Boolean;
function Nil return Object.Message_List;

procedure Report (Messages : Object.Message_List;
                  Response : Profile.Response_Profile := Profile.Get;
                  Top_Only : Boolean                  := False);

-- Displays the error messages in the standard format according to the
-- supplied profile. If Top_Only is true, only the first message in the
-- list will be displayed; other wise all messages in the list will be
-- Displayed.

function Messages (Error_Code : Object.Error_Code)
                  return Object.Message_List;

-- Extracts the message list from the error code. Returns the Nil list
-- if there are no messages.

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

-- Error codes in the category Obsolescence_Error may have a list of
-- the objects that would have been obsolesced.

function Change_Impact (Error_Code : Object.Error_Code)
                       return Object.Iterator;

-- Extracts the list of obsolesced objects from the error code.  Returns
-- the Nil iterator if there are none.

function Modified_Units
            (Error_Code : Object.Error_Code) return Object.Iterator;

-- Extracts the list of units that were implicitly coded/uncoded by the
-- operation that returned the error code.

function Value (Category       : Object.Category_Enumeration;
                Message        : String              := "";
                Messages       : Object.Message_List := Object.Nil;
                Change_Impact  : Object.Iterator     := Object.Nil;
                Modified_Units : Object.Iterator     := Object.Nil)
               return Object.Error_Code;
```





```
function Value (Category       : Object.Category_Enumeration;
                Message        : St.Item             := St.Nil;
                Messages       : Object.Message_List := Object.Nil;
                Change_Impact  : Object.Iterator     := Object.Nil;
                Modified_Units : Object.Iterator     := Object.Nil)
               return Object.Error_Code;

function Nil return Object.Error_Code;
-- Construct an Error code of a the given class and associated message.

function Image (The_Code        : Object.Error_Code;
                Level           : Natural;
                Prefix          : String;
                Expand_Pointers : Boolean) return String;


-------------------------------------------------------------------

package Low_Level is

    -- lower level routines to build and extract from handles,
    -- error codes, and iterators

    -------------------------------------------------------------

    -- Routines to set and retrieve the default action. This action is
    -- the action with a handle if one is not explicitly supplied.
    -- default action is never finished.  It is initially the
    -- null action id.
    -- Directory_tools routines take a handle as an argument will use
    -- the action from the handle.  If  source and destination are
    -- both arguments, the action comes from the destination.
    -- If the routine takes a name and context, the action from the
    -- context is used.  If neither these are true, the default action
    -- is used directly.

    -- Handles created by the directory system are created with the
    -- default action, with three exceptions.  The first exception is
    -- traversal.recursion.  This routine propagates the action
    -- from the supplied handle instead of using the default.

    -- The second exception is ada_implementation.open, which
    -- will start an action if one is not supplied and the access
    -- mode is READ.  This action is finished by closing the unit.

    -- The third exception is library_object.create, which
    -- will always start and finish an action, making the addition
```

```
-- permanent.  The user must manually back out by destroying any
-- created directories if the desire is to abandon the operation.
-- After the library is created, the action is set to the
-- default action.  As such, objects created in it will be covered
-- by the default in force at the time the directory is created.

procedure Set_Default_Action_Id (The_Action : Action.Id);

function Default_Action_Id return Action.Id;

-------------------------------------------------------------

-- object handle routines.  These extract the underlying
-- components supplied by the environment directory system.
-- If no action is supplied, the action from the handle is used.
-- If the handle's action is NULL, the default action is used.

procedure Get_Declaration (Handle    : Object.Handle;
                           The_Decl  : out Di.Declaration;
                           Status    : out Di.Error_Status;
                           Action_Id : Action.Id := Action.Null_Id;
                           Max_Wait  : Duration := Di.Default_Wait);

procedure Get_Object (Handle     :     Object.Handle;
                      The_Object : out Di.Object;
                      Status     : out Di.Error_Status);

procedure Get_Version (Handle      : Object.Handle;
                       The_Version : out Di.Version;
                       Status      : out Di.Error_Status;
                       Action_Id   : Action.Id := Action.Null_Id;
                       Max_Wait    : Duration := Di.Default_Wait);

procedure Get_Root (Handle    :     Object.Handle;
                    The_Root  : out Diana.Tree;
                    Status    : out Di.Error_Status;
                    Action_Id :     Action.Id := Action.Null_Id;
                    Max_Wait  :     Duration  := Di.Default_Wait);

function Get_Class (Handle : Object.Handle) return Di.Class;

function Get_Subclass (The_Subclass : Subclass) return Di.Subclass;
function Get_Subclass (Handle : Object.Handle)  return Di.Subclass;

-- return the string supplied as the object name.

function Object_Name (Handle : Object.Handle) return String;
function Object_Name (Handle : Object.Handle) return St.Item;
```





```
function Action_Id (Handle : Object.Handle) return Action.Id;
function Finish_Action_On_Close
            (Handle : Object.Handle)        return Boolean;

-- constructors to make object.handles

procedure Set_Class (Handle : Object.Handle; Class : Di.Class);

procedure Set_Object_Name
             (Handle : Object.Handle; The_Name : String);
procedure Set_Object_Name
             (Handle : Object.Handle; The_Name : St.Item);
procedure Set_Error_Code (Handle   : Object.Handle;
                          The_Code : Object.Error_Code);

procedure Set_Action_Id (Handle                 : Object.Handle;
                         The_Action             : Action.Id;
                         Finish_Action_On_Close : Boolean := False);

procedure Set_Root (Handle : Object.Handle; The_Root : Diana.Tree);

procedure Set_Handle_Data
             (Handle          : Object.Handle;
              The_Error       : Object.Error_Code := Object.Nil;
              The_Name        : String            := "";
              The_Object      : Di.Object         := Di.Nil;
              The_Version     : Di.Version        := Di.Nil;
              The_Class       : Di.Class          := Di.Nil;
              The_Declaration : Di.Declaration    := Diana.Empty;
              The_Root        : Diana.Tree        := Diana.Empty;
              The_Action      : Action.Id         :=
                 Object.Low_Level.Default_Action_Id);

function Make_Handle
            (The_Error       : Object.Error_Code;
             The_Name        : String         := "";
             The_Object      : Di.Object      := Di.Nil;
             The_Version     : Di.Version     := Di.Nil;
             The_Class       : Di.Class       := Di.Nil;
             The_Declaration : Di.Declaration := Diana.Empty;
             The_Root        : Diana.Tree     := Diana.Empty;
             The_Action      : Action.Id      :=
                Object.Low_Level.Default_Action_Id)
            return Object.Handle;

function Make_Handle
            (The_Error       : Object.Error_Code;
             The_Name        : St.Item        := St.Nil;
             The_Object      : Di.Object      := Di.Nil;
```

```
                The_Version     : Di.Version     := Di.Nil;
                The_Class       : Di.Class       := Di.Nil;
                The_Declaration : Di.Declaration := Diana.Empty;
                The_Root        : Diana.Tree     := Diana.Empty;
                The_Action      : Action.Id      :=
                   Object.Low_Level.Default_Action_Id)
               return Object.Handle;

    -----------------------------------------------------------------

    -- object iterator constructors

    procedure Set_Error_Code (Iter : Object.Iterator;
                              Code : Object.Error_Code);

    procedure Set_Pattern (Iter : Object.Iterator; Pattern : String);
    procedure Set_Pattern (Iter : Object.Iterator; Pattern : St.Item);

    function Pattern (Iter : Object.Iterator) return String;
    function Pattern (Iter : Object.Iterator) return St.Item;

    -- If possible, the iterator uses the environment iterators.
    -- To do this, constructors are supplied for the types of interest.

    function Make_Iterator
               (Code      : Object.Error_Code;
                An_Object : Object.Handle;
                Pattern   : String                    := "?";
                The_Class : Object.Class_Enumeration :=
                   Object.Unknown_Class) return Object.Iterator;
    function Make_Iterator
               (Code          : Object.Error_Code;
                A_Naming_Iter : Di.Naming.Iterator;
                Pattern       : String                    := "?";
                The_Class     : Object.Class_Enumeration :=
                   Object.Unknown_Class) return Object.Iterator;
    function Make_Iterator
               (Code           : Object.Error_Code;
                A_Version_Iter : Di.Traversal.Version_Iterator;
                Pattern        : String                    := "?";
                The_Class      : Object.Class_Enumeration :=
                   Object.Unknown_Class) return Object.Iterator;
    function Make_Iterator
               (Code      : Object.Error_Code;
                An_Object : Object.Handle;
                Pattern   : St.Item                   := St.Nil;
                The_Class : Object.Class_Enumeration :=
                   Object.Unknown_Class) return Object.Iterator;
```





```
        function Make_Iterator
                   (Code          : Object.Error_Code;
                    A_Naming_Iter : Di.Naming.Iterator;
                    Pattern       : St.Item                   := St.Nil;
                    The_Class     : Object.Class_Enumeration :=
                       Object.Unknown_Class) return Object.Iterator;
        function Make_Iterator
                   (Code           : Object.Error_Code;
                    A_Version_Iter : Di.Traversal.Version_Iterator;
                    Pattern        : St.Item                   := St.Nil;
                    The_Class      : Object.Class_Enumeration :=
                       Object.Unknown_Class) return Object.Iterator;
        -- the class argument specifies a filter class.  Only items of the
        -- specified class will be returned.  If unknown_class is given,
        -- the filter is disabled.

        -------------------------------------------------------------

        -- routines that handle low level error code actions.

        function Translate_Status (The_Status : Di.Error_Status)
                                  return Object.Category_Enumeration;
        function Translate_Status (The_Status : Di.Naming.Name_Status)
                                  return Object.Category_Enumeration;

        procedure Set_Category (Code  : Object.Error_Code;
                                Class : Object.Category_Enumeration);

        procedure Set_Message (Code : Object.Error_Code; Msg : String);
        procedure Set_Message (Code : Object.Error_Code; Msg : St.Item);
        procedure Set_Message_List (Code : Object.Error_Code;
                                    List : Error_Messages.Errors);
    end Low_Level;

private
    type Handle_Data;
    type Handle is access Handle_Data;
    pragma Segmented_Heap (Handle);
    type Iterator_Kind_Enum is (Version_Iter, Name_Iter, No_Iter);
    type Iterator_Data (Iterator_Kind : Iterator_Kind_Enum);
    type Iterator is access Iterator_Data;
    pragma Segmented_Heap (Iterator);
    type Error_Code_Data;
    type Error_Code is access Error_Code_Data;
    pragma Segmented_Heap (Error_Code);
    type Message_List is new Error_Messages.Errors;
    type Subclass is new Di.Subclass;
end Object;
```

```
package Naming is

    -- Provides mechanisms for manipulating and resolving names and for
    -- establishing a context for name resolution.

    subtype String_Name is String;

    -- Lexically and syntactically a Directory system string name.

    subtype Simple_String_Name is String;

    -- A single segment of a string name: an Ada identfier with or wihout
    -- attributes

    subtype Context is Object.Handle;

    -- The Directory System Object that serves as the initial context
    -- for name resolution.  May be any Object

    procedure Set_Default_Context (The_Context :     Naming.String_Name;
                                   Status      : out Object.Error_Code);
    procedure Set_Default_Context (The_Context :     Naming.Context;
                                   Status      : out Object.Error_Code);

    -- Establishes the default naming context for the job.

    function Default_Context return Naming.String_Name;
    function Default_Context return Naming.Context;

    -- Returns the default name resolution context for the job.

    function Is_Well_Formed (A_Name : String_Name) return Boolean;

    -- Tests whether a name is lexically and syntactically valid.

    function Prefix (The_Name : String_Name) return String_Name;

    -- Removes the last segment from a selected name and returns
    -- the prefix.
    -- Prefix ("A.B.C") => "A.B"
    -- Prefix ("A") => ""

    function Simple_Name (The_Name : String_Name) return Simple_String_Name;
    -- Returns only the last segment of a selected name, without attributes
    -- Simple_name ("A.B.C") => "C"
    -- Simple_name ("A") => "A"
```





```
    function Head (The_Name : String_Name) return Simple_String_Name;

    -- Returns only the first segment of a selected name.
    --      Head ("A.B.C") => "A"
    -- Head ("A") => "A"
    -- Head ("!A") => "!"

    function Tail (The_Name : String_Name) return String_Name;

    -- Removes the first segment from a selected name and returns the tail.
    --      Tail ("A.B.C") => "B.C"
    -- Tail ("A") => ""

    function Attributes (A_Name : String_Name) return String;

    -- Returns the Attributes at the end of the given string name.
    -- If the simple name of the given string name has no attributes,
    -- the null string is returned.  The returned string starts with '''.

    function Attribute
                (A_Name : String_Name; Kind : String := "C") return String;

    -- Returns the argument of the attribute designated by the kind
    -- parameter that appears in the simple name of the given name.
    -- If no argument follows the named attribute, the name of the attribute
    -- is returned. (Parentheses are not part of the returned string.) If
    -- the named attribute does not appear, the null string is returned.

    function Nickname_Attribute
                (A_Name : String_Name; Kind : String := "N") return String
        renames Attribute;
    function Class_Attribute
                (A_Name : String_Name; Kind : String := "C") return String
        renames Attribute;
    function Version_Attribute
                (A_Name : String_Name; Kind : String := "V") return String
        renames Attribute;

    -- returns the argument to the Nickname, Class and Version attributes
    -- respectively.

    function Part_Attribute (A_Name : String_Name) return String;

    -- Returns either "'SPEC"  or "'BODY" or the null string if neither
    -- of these are present
```

```
function Expanded_Name (The_Name : Naming.String_Name;
                        Context  : Naming.Context := Default_Context)
                       return String_Name;
-- Expands any prefix characters in the name appropriately.

function Full_Name (The_Object : Object.Handle)
                   return Naming.String_Name;
function Full_Name (The_Object : String_Name) return Naming.String_Name;

-- Computes the fully qualified string name for the The_Object
-- exclusive of qualifying attributes.

function Simple_Name (The_Object : Object.Handle)
                     return Simple_String_Name;

-- Computes the simple name for the The_Object exclusive of qualifying
-- attributes. (= Simple_Name (Full_Name (The_Object)));

function Unique_Full_Name
            (The_Object : Object.Handle)           return String_Name;
function Unique_Full_Name (The_Object : String_Name) return String_Name;

-- Full_Name with 'body, 'n(), and 'v() attributes as needed.

function Unique_Simple_Name
            (The_Object : Object.Handle) return Simple_String_Name;
function Unique_Simple_Name
            (The_Object : String_Name)   return Simple_String_Name;

-- Simple_Name with 'body, 'n(), and 'v() attributes as needed.


function Ada_Name (The_Object : Object.Handle) return String_Name;
function Ada_Name (The_Object : String_Name)   return String_Name;

-- Returns a valid ada name for an Object.  (No extra attributes,
-- no library names, no "!", etc.)

function Resolution (Name        : Naming.String_Name;
                     Context     : Naming.Context := Default_Context;
                     Object_Only : Boolean        := True)
                    return Object.Handle;

-- Resolve name to a single Object.  Wild cards may be used, but
-- the name must resolve to a unique Object.
```





```
function Resolution (Name         : Naming.String_Name;
                     Context      : Naming.Context := Default_Context;
                     Objects_Only : Boolean        := True)
                    return Object.Iterator;

-- Resolves (ambiguous) Source name in the given context. If
-- Objects_Only is true, only (separate) Objects that match the
-- name will be included; when false, Ada declarations will
-- be included even if no separate Object is associated with them.
-- Resolution is more efficient if Objects_Only is true.

function Pattern (Iter : Object.Iterator) return String_Name;
function Pattern (Iter : Object.Iterator) return String_Table.Item;

-- Returns a string_name that describes the objects of the iterator. For
-- the iterator returned by resolution, this is the value of the Name
-- parameter.


function Has_Substitution_Characters
            (Target : String_Name) return Boolean;

function Target_Name (Iter : Object.Iterator; Target : String_Name)
                     return String_Name;

-- Replaces the substitution characters in the given Target name
-- with the appropriate values derived from the current Object of
-- the iteration (using the Pattern of the iterator).

function Target_Name (The_Object : Object.Handle;
                      Pattern    : String_Name;
                      Target     : String_Name) return String_Name;

function Target_Name (Source : String_Name; Target : String_Name)
                     return String_Name;

-- Given an Object and a source name (with wild cards) that
-- matches the name of the The_Object, returns a target string in which
-- substitution characters have been replaced by the matching
-- portions of the The_Object's name as indicated by the source name
-- pattern.

function Nickname (Def_Id : Object.Handle)      return String;
function Nickname (Def_Id : Naming.String_Name) return String;

-- Returns the user-defined nickname associated with Def_Id, if one has
-- been specified; returns the system-defined nickname otherwise.
```

```
    function System_Nickname (Def_Id : Object.Handle)      return String;
    function System_Nickname (Def_Id : Naming.String_Name) return String;

    -- returns the system-assigned nickname for the given Def_Id, whether
    -- or not a user-defined nickname has been assigned. The
    -- system-assigned nickname is the image of the ordinal position
    -- (1-origin) of the def_id among its namesakes in its declarative
    -- region.

    function Is_Overloaded (Def_Id : Object.Handle)      return Boolean;
    function Is_Overloaded (Def_Id : Naming.String_Name) return Boolean;

    -- returns true if the given Def_Id is an overloaded Ada declaration.
end Naming;

package Traversal is
    -- Provides operations for traversing the Directory System
    -- in a variety of ways.


    function Position (The_Object : Naming.String_Name) return Natural;
    function Position (The_Object : Object.Handle)      return Natural;

    -- The position of the object in its declarative context.  The first
    -- item is position '0'.

    Default_Position : constant Natural := Natural'Last;

    -- Specifies the end of the list.

    function Universe return Object.Handle;

    -- Returns the (somewhat special) Object corresponding to the
    -- root of the universe.

    function Parent (The_Object : Object.Handle) return Object.Handle;

    -- Returns the parent Object for The_Object.

    function Enclosing_World
                (The_Object : Object.Handle) return Object.Handle;

    function Enclosing_Library
                (The_Object : Object.Handle) return Object.Handle;

    -- Returns the nearest enclosing Library/World that contains
    -- the specified Object.
```





```
    function Associated_World
                (The_Object : Object.Handle) return Object.Handle;

    function Associated_Library
                (The_Object : Object.Handle) return Object.Handle;

    -- Returns the nearest enclosing Library/World that contains
    -- the specified Object, but if the object is a Library/World, that
    -- value is returned.


    function Child (The_Object : Object.Handle;
                    Child_Name : Naming.Simple_String_Name)
                   return Object.Handle;

    -- Retrieve the named subobject.

    function Children (The_Object : Object.Handle;
                       Pattern    : Naming.Simple_String_Name := "@";
                       Declared   : Boolean                   := True;
                       Class      : Object.Class_Enumeration  :=
                          Object.Unknown_Class) return Object.Iterator;

    -- Initializes the iteration over the children that match the
    -- specified name pattern.  If Declared is True, only children
    -- that are declared in the given Version of the the Object will be
    -- returned. If Declared is false, all existing children of the
    -- Object are returned even if they have no stub declaration in the
    -- given Version of the unit.
    -- If a class is provided, only children of that class are returned


    function Versions (The_Object : Object.Handle;
                       Forward    : Boolean := True) return Object.Iterator;

    -- Get all versions of the given object; iterator is ordered forward
    -- or backward according to creation time based on the Forward boolean.

    type Control_Enumeration is (Continue, Abandon_Level,
                                 Abandon_Recursion, Skip_Children);

    generic
        type State_Record is private;
        with procedure Op (Depth : Positive;
                           State : in out State_Record;
                           The_Object : Object.Handle;
                           Status : out Object.Error_Code;
                           Control : in out Traversal.Control_Enumeration);
```

```
procedure Recursion (State        : in out State_Record;
                     The_Object   : Object.Handle;
                     Status       : out Object.Error_Code;
                     Pattern      : Naming.Simple_String_Name := "@";
                     Class        : Object.Class_Enumeration :=
                        Object.Unknown_Class;
                     Subunits     : Boolean := True;
                     Directories  : Boolean := True;
                     Worlds       : Boolean := False;
                     Objects_Only : Boolean := True;
                     Deleted      : Boolean := False);

-- Performs a depth-first traversal of the Directory structure
-- rooted at the given Object.

-- The Boolean parameters control scope of the traversal as follows:
--      Subunits     : Subunits of Ada units are included
--      Directories  : Nested directories are included
--      Worlds       : Nested worlds are included
--      Objects_Only : Only separate Ada objects are included
--      Deleted      : deleted objects are included

-- The formal procedure Op is called for each Object visited. The
-- State variable is passed from call to call.  The traversal will
-- terminate immediately if the error code parameter has a bad value
-- when the Op procedure returns. This error code is returned as the
-- error code of the Recursion procedure. Recursion can also be
-- controlled by the Control parameter.

-- Only Objects with a simple name that matches the given Pattern are
-- visited.

-- Only Objects of the given Class (if not nil) are visited.

-- The Pattern and Class attributes do not affect the scope of the
-- traversal, just the Objects on which Op is called.  For example, if
-- Class is Object.Class (Object.Ada), Op will be called for each Ada
-- object (including subunits) nested within the given object and within
-- any directory nested within the given Object and in the same world as
-- the given object.

generic
    type State_Record is private;
    with procedure Op (State : in out State_Record;
                       The_Object : Object.Handle;
                       The_Objects : out Object.Iterator;
                       Status : out Object.Error_Code;
                       Control : in out Traversal.Control_Enumeration);
```





```
procedure Closure (State       : in out State_Record;
                   In_Objects  :        Object.Iterator;
                   Out_Objects : out    Object.Iterator;
                   Status      : out    Object.Error_Code);

-- Computes the transitive closure of the Op procedure applied to the
-- input objects.

-- The only control_enumeration values supported are
--      Continue, Abandon_Recursion

-- The objects in the iterator Out_objects are in a known order. This
-- order is such that if the "op" procedure were computing the closure
-- of "with x;" statements, the objects in the iterator could be
-- promoted in order without incurring 'uninstalled' errors.  If the
-- desire is to demote the objects, the list should be reversed.

-- A formal way of stating the order is:
--  For each object, operate on its children, then operate on the object

-- The algorithm is:
--  1.  result_iterator := object.create;
--  2.  For each each object in in_objects:
--      i.  If the object is not 'in' the result_iterator, then
--          a.  'Visit' the object.
--          b.  Recurse, (step 2) with the result of visit as the
--              new in_objects.
--          c.  add the object to the result_iterator.
--  3.  out_objects := result_iterator.

end Traversal;

package Any_Object is
    -- Operations to Create, Copy and Destroy Objects.

    -- Several versions of an object may exist simultaneously. One of
    -- these versions may be designated the default version, which will
    -- be the one accessed if no specific version is referenced.  For
    -- Ada objects, only the default version may be at the installed
    -- state or higher.  For an object in a library, if there is no
    -- default version, there will be no declaration for the object
    -- visible. (However, for an Ada subunit, a stub declaration may be
    -- visible in the parent object even though there is no default
    -- version for the subunit.)

    -- The versions that are not the default version are called deleted
    -- versions. The number of versions of an object that are retained in
    -- the system is controlled by a retention count parameter, which may
```

```
-- be set for each object individually. When the number of deleted
-- versions exceeds the retention count (either because the count has
-- been changed, or additional versions are deleted), existing versions
-- are destroyed (oldest first) until the count is satisfied.

Default_Retention_Count : constant := -1;
-- A special retention count value that directs an operation to use the
-- existing count for the object or inherit one from the parent object.

function Retention_Count (The_Object : Object.Handle) return Natural;

procedure Set_Retention_Count
            (The_Object      :     Object.Handle;
             Retention_Count :     Integer := Default_Retention_Count;
             Status          : out Object.Error_Code);

function Is_Visible (The_Object : Object.Handle)       return Boolean;
function Is_Visible (The_Object : Naming.String_Name) return Boolean;

function Has_Versions (The_Object : Object.Handle)       return Boolean;
function Has_Versions (The_Object : Naming.String_Name) return Boolean;

function Has_Default_Version
           (The_Object : Object.Handle)      return Boolean;
function Has_Default_Version
           (The_Object : Naming.String_Name) return Boolean;


procedure Create (The_Object  :     Object.Handle;
                  New_Version : out Object.Handle;
                  Status      : out Object.Error_Code);

-- Create a new version of an existing Object. The new version becomes
-- the default version.

procedure Create (Object_Name : Naming.String_Name;
                  New_Version : out Object.Handle;
                  Status : out Object.Error_Code;
                  Class : Object.Class_Enumeration :=
                     Object.Unknown_Class;
                  Context : Naming.Context := Naming.Default_Context;
                  Position : Natural := Traversal.Default_Position;
                  Subclass : Object.Subclass := Object.Nil);

-- Creates a new version of the named object, which becomes the default
-- version.  If an Object does not
-- yet exist with that name, one is created of the indicated class.
-- The declaration for the object is placed at the indicated position in
-- its parent context.
```





```
procedure Copy (Source      :     Object.Handle;
                Destination :     Object.Handle;
                New_Version : out Object.Handle;
                Status      : out Object.Error_Code);

-- The version of the Source Object specified by the Source handle is
-- copied to the Destination Object, where it becomes the default
-- version of the Destination Object.

procedure Copy (Source : Naming.String_Name;
                Destination : Naming.String_Name;
                New_Version : out Object.Handle;
                Status : out Object.Error_Code;
                Source_Context : Naming.Context :=
                   Naming.Default_Context;
                Destination_Context : Naming.Context :=
                   Naming.Default_Context;
                Position : Natural := Traversal.Default_Position;
                Subclass : Object.Subclass := Object.Nil);

-- Copies the value.  Creates an entirely new declaration and Object
-- at the destination if one did not exist.  If the declaration
-- existed, but the specified Version did not, creates
-- a new Version of the destination Object and makes it the default.
-- If the destination Version already
-- exists, overwrites the old value with the new.  Copied Ada
-- units are source only, regardless of the state of the Source.
-- Copies only the Source Object (no sub-Objects).

procedure Delete (The_Object      : Object.Handle;
                  Status          : out Object.Error_Code;
                  Retention_Count : Integer := Default_Retention_Count);

procedure Delete (The_Object      : Naming.String_Name;
                  Status          : out Object.Error_Code;
                  The_Context     : Naming.Context :=
                     Naming.Default_Context;
                  Retention_Count : Integer := Default_Retention_Count);

-- Deletes the default version of the specified Object. If,
-- after the deletion, the number of deleted versions exceeds the
-- retention count, the oldest version will be destroyed.
-- If the target of any delete is an installed Ada unit, first
-- attempts to withdraw the unit, then (if there were no errors)
-- performs the delete.
```

```
procedure Undelete (The_Object :      Object.Handle;
                    Status     : out Object.Error_Code);

procedure Undelete (The_Object  :      Naming.String_Name;
                    Status      : out Object.Error_Code;
                    The_Context :      Naming.Context :=
                       Naming.Default_Context);

-- Make the specified version of the given object the default version.
-- Reinstates the declaration (visibility) of the object if it had been
-- deleted.

procedure Expunge (The_Object : Naming.String_Name;
                   Status : out Object.Error_Code;
                   Retention_Count : Integer := 0;
                   Context : Naming.Context := Naming.Default_Context);

procedure Expunge (The_Object      :      Object.Handle;
                   Status          : out Object.Error_Code;
                   Retention_Count :      Integer := 0);

-- Destroy deleted versions of an Object (oldest first) until
-- the numer of deleted versions remaining is no more than the retention
-- count.

procedure Destroy (The_Object      :      Object.Handle;
                   Status          : out Object.Error_Code;
                   Retention_Count :      Integer :=
                      Default_Retention_Count);

procedure Destroy (The_Object      :      Naming.String_Name;
                   Status          : out Object.Error_Code;
                   The_Context     :      Naming.Context :=
                      Naming.Default_Context;
                   Retention_Count :      Integer        :=
                      Default_Retention_Count);

-- Destroys the specified Version of the Object. If this is the
-- default version of the object, the declaration is deleted as well.
-- If the target of any destroy is an installed Ada unit, Destroy first
-- attempts to withdraw the unit, then (if there were no errors)
-- performs the destroy.  After the destroy the object is expunged,
-- using the supplied retention count.

function Is_Frozen (The_Object : Naming.String_Name) return Boolean;
function Is_Frozen (The_Object : Object.Handle)      return Boolean;
-- Test whether an Object is frozen.  Frozen objects cannot be changed.
```





```
    procedure Freeze (The_Object :      Object.Handle;
                      Recursive  :      Boolean := False;
                      Status     : out Object.Error_Code);

    -- Freeze an Object or a unit (and its children which are in the
    -- same control point) so that it cannot be changed.

    procedure Unfreeze (The_Object :      Object.Handle;
                        Recursive  :      Boolean := False;
                        Status     : out Object.Error_Code);

    -- Unfreeze an Object or a unit (and its children which are in
    -- the same control point) so that it can be manipulated normally.

end Any_Object;

package Ada_Object is

    -- Directory operations specific to the class ADA are defined here.

    -- Objects of class Ada correspond to the notion of Compilation Unit
    -- as defined by the LRM.  These Ada units can be in one of six states:

    type Unit_State is
       (Nonexistent, Archived,     -- text only; no Diana tree
        Source,        -- Source ready to be installed
        Installed,     -- Semantically consistent.
        Coded          -- Has been code generated.
        );

    function State (For_Unit : Object.Handle) return Ada_Object.Unit_State;
    function State (For_Unit : Naming.String_Name)
               return Ada_Object.Unit_State;

    function Is_Source (The_Unit : Object.Handle)      return Boolean;
    function Is_Source (The_Unit : Naming.String_Name) return Boolean;

    function Is_Installed (The_Unit : Object.Handle)      return Boolean;
    function Is_Installed (The_Unit : Naming.String_Name) return Boolean;

    type Compilation_Kind is (Not_Class_Ada, Uncertain,
                              Library_Unit, Library_Unit_Body,
                              Subunit, Internal_Declaration);

    -- Uncertain is returned for source units in which insufficient
    -- text is present to determine it's kind.  Internal_declaration
    -- is returned for declarations with bodies contained in a
    -- library unit
```

```
function Kind (Ada_Unit : Object.Handle)      return Compilation_Kind;
function Kind (Ada_Unit : Naming.String_Name) return Compilation_Kind;

type Unit_Kind is (Not_Class_Ada, Uncertain, Function_Spec,
                   Function_Body, Procedure_Spec, Procedure_Body,
                   Package_Spec, Package_Body, Function_Instantiation,
                   Procedure_Instantiation, Package_Instantiation,
                   Generic_Function, Generic_Procedure, Generic_Package,
                   Function_Rename, Procedure_Rename, Package_Rename,
                   Task_Spec, Task_Type, Task_Body, Not_A_Unit);

-- More specific classification of type of Ada unit (declaration).

function Kind (Ada_Unit : Object.Handle)      return Unit_Kind;
function Kind (Ada_Unit : Naming.String_Name) return Unit_Kind;

function Is_Visible_Part (Ada_Unit : Object.Handle)      return Boolean;
function Is_Visible_Part (Ada_Unit : Naming.String_Name) return Boolean;

-- Determines whether the given unit corresponds to a visible part.

function Is_Subunit (Ada_Unit : Object.Handle)      return Boolean;
function Is_Subunit (Ada_Unit : Naming.String_Name) return Boolean;

function Other_Part (Ada_Unit : Object.Handle) return Object.Handle;

-- Given the visible part, return the body, and vice versa. Returns
-- a nil unit if there is no complement.
-- May have to actually create the Object.


function Subunit (Ada_Unit     : Object.Handle;
                  Subunit_Name : Naming.Simple_String_Name)
                 return Object.Handle;

-- Retrieve the named subunit.

function Subunits (Ada_Unit : Object.Handle;
                   Pattern  : Naming.Simple_String_Name := "@";
                   Declared : Boolean := True) return Object.Iterator;

-- Computes a list of all subunits of the given unit whose name
-- matches the given Pattern.  If Declared is True, only subunits
-- that are declared in the given Version of the the unit will be
-- returned. If Declared is false, all existing subunits of the unit
-- are returned even if they have no stub declaration in the given
-- Version of the unit.
```





```
function Depends_On (Defining_Id : Naming.String_Name;
                     The_Context : Naming.Context :=
                        Naming.Default_Context) return Object.Iterator;
function Depends_On (Defining_Id_Handle : Object.Handle)
                    return Object.Iterator;

-- Computes the set of ada units that depend upon the defining_id given.
-- A defining_id is the full name of the defining occurance of the item,
-- and can be any ada object, from a package to a variable.

function List_Of_Withs (Ada_Unit    : Naming.String_Name;
                        The_Context : Naming.Context :=
                           Naming.Default_Context)
                       return Object.Iterator;
function List_Of_Withs (Ada_Unit_Handle : Object.Handle)
                       return Object.Iterator;
-- computes the set of units 'with'ed by the supplied unit.

----------------------------------------------------------

-- Operations to promote and demote declarations. Promoting
-- declarations moves them "up" to higher declaration states (toward
-- Coded), while demotion moves declarations "down" to lower
-- declaration states (toward Nonexistent). Promoting to a lower
-- state or demoting to a higher state is an Illegal_Operation.

procedure Promote (Ada_Unit   :     Object.Handle;
                   Status     : out Object.Error_Code;
                   Goal_State :     Ada_Object.Unit_State :=
                      Ada_Object.Installed;
                   Switches   :     Object.Handle := Object.Nil);

procedure Promote (Ada_Unit : Naming.String_Name;
                   Status : out Object.Error_Code;
                   Goal_State : Ada_Object.Unit_State :=
                      Ada_Object.Installed;
                   Switches : Object.Handle := Object.Nil;
                   Context  : Naming.Context := Naming.Default_Context);

-- A subunit may not be promoted to a state higher than that of
-- its parent, except that a subunit may be coded before the parent
-- is coded.

procedure Demote (Location   :     Object.Handle;
                  Status     : out Object.Error_Code;
                  Goal_State :     Ada_Object.Unit_State :=
                     Ada_Object.Source;
                  Switches   :     Object.Handle        := Object.Nil);
```

```
      procedure Demote (Location : Naming.String_Name;
                        Status : out Object.Error_Code;
                        Goal_State : Ada_Object.Unit_State :=
                           Ada_Object.Source;
                        Switches : Object.Handle := Object.Nil;
                        Context  : Naming.Context := Naming.Default_Context);

      -- This operation will fail with obsolescence error if any
      -- declarations (including installed subunits) depend upon
      -- demoted declarations.
   end Ada_Object;

   package Library_Object is

      -- Directory operations specific to Libraries. Unlike other objects,
      -- there can be only one version of a library object.

      type Library_Kind is (Not_A_Library, Directory, World);

      function Kind (Any_Object : Object.Handle)
                     return Library_Object.Library_Kind;
      function Kind (Any_Object : Naming.String_Name)
                     return Library_Object.Library_Kind;

      function Is_Library (Any_Object : Object.Handle)      return Boolean;
      function Is_Library (Any_Object : Naming.String_Name) return Boolean;

      -- Returns true IFF the indicated object is an object of class Library

      function Is_World (Any_Object : Object.Handle)      return Boolean;
      function Is_World (Any_Object : Naming.String_Name) return Boolean;

      function Is_Directory (Any_Object : Object.Handle)      return Boolean;
      function Is_Directory (Any_Object : Naming.String_Name) return Boolean;

      procedure Set_Switch_Object (The_Library :     Object.Handle;
                                   The_File    :     Object.Handle;
                                   Status      : out Object.Error_Code);
      procedure Set_Switch_Object (The_Library :     Naming.String_Name;
                                   The_File    :     Naming.String_Name;
                                   Status      : out Object.Error_Code);

      function Switch_Object
                  (The_Library : Object.Handle) return Object.Handle;
      function Switch_Object (The_Library : Naming.String_Name)
                          return Naming.String_Name;
      -- Used to manipulate the file Object used to store switch files.
```





```
      subtype Volume_Id is Natural range 0 .. 31;
      -- Used to represent a disk volume.

      function Nil return Library_Object.Volume_Id;
      function Is_Nil (The_Volume : Library_Object.Volume_Id) return Boolean;
      function Volume (The_Library : Object.Handle)
                      return Library_Object.Volume_Id;

      procedure Create (Name : Naming.Simple_String_Name;
                        Kind : Library_Object.Library_Kind;
                        New_Library : out Object.Handle;
                        Status : out Object.Error_Code;
                        Volume : Library_Object.Volume_Id :=
                           Library_Object.Nil;
                        Context : Naming.Context := Naming.Default_Context;
                        Position : Natural := Traversal.Default_Position;
                        Subclass : Object.Subclass := Object.Nil);

      -- Creates a new Library (Directory or World) at the indicated
      -- position.  If an appropriate declaration already exists and has
      -- no directory Object associated with it, that stub will be used
      -- rather than creating a new one.
   end Library_Object;

   package Ada_Implementation is
      subtype Root     is Diana.Tree;
      subtype Any_Node is Diana.Tree;

      package Ai renames Ada_Implementation;

      function Is_Source    (For_Node : Any_Node) return Boolean;
      function Is_Installed (For_Node : Any_Node) return Boolean;

      procedure Will_Be_A_Comp_Unit (Root    :     Ai.Root;
                                     Verdict : out Boolean;
                                     Status  : out Object.Error_Code);
      -- A predicate which determines if the root of a child unit will
      -- be promoted in place or made into a comp_unit.

      --    procedure Replace_Comment (Node : Diana.Tree;
      --                               New_Comment :
      --                                  Comment_Definitions.Comment;
      --                               Pre_Comment : Boolean := True;
      --                               Status : out Error.Code);

      -- Make New_Comment the Pre_/Post_Comment of Node.
      -- Node must be in an installed unit.
```

```
    type Open_Mode is
       (None,  -- Mode None only applies to installed units.
        --     There is no synchronization with mode None.
        Read   -- Mode Read applies to either source or installed
        --     units, and aquires a non-exclusive read lock
        --     (exclusive of update, but not other readers).
        );

    procedure Open (The_Unit :     Object.Handle;
                    Mode     :     Open_Mode;
                    Root     : out Ai.Root;
                    Status   : out Object.Error_Code);

    -- Returns the root of the separate tree designated by The_Unit.
    -- Opens the unit with the specified access Mode.
    -- Incompatible access modes Error in queueing or Lock_Error.

    -- Open and Close invoke policy specific pre and post operations
    -- before and after execution.

    -- Open first tries to open an object.  If that fails, it
    -- then tries to open the object containing the declaration.
    -- In the latter case, the return value is the declaration within
    -- the object.


    procedure Close (The_Unit :     Object.Handle;
                     Status   : out Object.Error_Code);

    -- Closes the indicated unit, releasing access.

    procedure Get_Root (Node   :     Any_Node;
                        Root   : out Ai.Root;
                        Status : out Object.Error_Code);

    -- Returns the Root of the unit represented by the Node.


    procedure Get_Handle (Node   :     Any_Node;
                          Handle : out Object.Handle;
                          Status : out Object.Error_Code);

    -- Returns the Object containing the Node.
end Ada_Implementation;
```





```
    package Statistics is
        subtype User    is Object.Handle;
        subtype Session is Object.Handle;

        function Time_Of_Last_Update
                    (The_Object : Object.Handle) return Calendar.Time;

        function Time_Of_Last_Read
                    (The_Object : Object.Handle) return Calendar.Time;

        function Time_Of_Creation
                    (The_Object : Object.Handle) return Calendar.Time;

        function Last_Updater (The_Object : Object.Handle) return User;

        function Session_Of_Last_Updater
                    (The_Object : Object.Handle) return Session;

        function Last_Reader (The_Object : Object.Handle) return User;

        function Session_Of_Last_Reader
                    (The_Object : Object.Handle) return Session;

        function Creator (The_Object : Object.Handle) return User;

        function Session_Of_Creator (The_Object : Object.Handle) return Session;

        function Total_Size (The_Object : Object.Handle) return Long_Integer;

        function Header_Size (The_Object : Object.Handle) return Natural;

        function Object_Size (The_Object : Object.Handle) return Long_Integer;
        function Last_Edit_Time (The_Unit : Object.Handle) return Calendar.Time;
    end Statistics;

end Directory_Tools;
```

```
with Calendar;
with Machine;

package Disk_Daemon is

    subtype Volume_Number is Integer range 0 .. 31;
    subtype Task_Vpids    is Machine.Job_Id;

    -- The procedural operations of this package are noops when given
    -- invalid parameters (out of range, volume does not exist, ...).
    -- The functions return 'first when given invalid parameters.

    type Threshold_Kinds is
       (Start_Collection,     -- Default 25%; start collection on this volume.
        Raise_Priority,       -- Default 15%; raise priority of all collection
        --     until this volume gets above this threshold.
        Stop_Jobs,            -- Default 10%; stop user jobs and max priority
        --     until this volume gets above this threshold.
        Suspend_System);      -- Default 3%; suspend the system.

    subtype Percentage is Natural range 0 .. 100;

    function Exists (Volume : Volume_Number) return Boolean;

    procedure Set_Threshold (Volume                : Volume_Number;
                             Kind                  : Threshold_Kinds;
                             At_Remaining_Capacity : Percentage);
    function  Get_Threshold (Volume : Volume_Number; Kind : Threshold_Kinds)
                           return Percentage;

    function Capacity        (Volume : Volume_Number) return Natural;
    function Used_Capacity   (Volume : Volume_Number) return Natural;
    function Unused_Capacity (Volume : Volume_Number) return Natural;

    -- The capacity functions return the size in number of 1024 byte pages.

    subtype Collection_Priority is Integer range -1 .. 6;

    -- -1 => Attempts to collect using just "spare cpu cycles"; if there are
    --       no spare cycles, will wait forever (or until some agent increases
    --       the priority).
    --
    --  0 => Slowest priority without backoff; runs on par with background jobs
    --       that do not use 'priority.  Small impact on performance.
    --
    --  2 => Will preempt most background jobs.  Runs on par with a background
    --       job that uses the best 'priority.
```





```
    --
    --  3 => Runs on par with most foreground jobs.  Tends to have a big impact
    --       on performance, since it will compete with commands.
    --
    --  4 => Preempts most foreground jobs.  Should still be able to edit.
    --       But commands will run VERY slowly.
    --
    --  6 => Preempts virtually all activity, except that from the console.

    -- Note that there is a policy function which places a lower bound on the
    -- current collection priority.  See the Set_Priority_Policy operation,
    -- below.

    procedure Perform_Garbage_Collection
                 (Volume                    :     Volume_Number;
                  Max_Wait                  :     Duration;
                  Desired_Priority          :     Collection_Priority := 0;
                  This_Call_Did_A_Gc_Pass : out Boolean);

    -- Causes a garbage collection pass on the specified volume.  Caller will
    -- wait for at most MAX_WAIT seconds for the garbage collection pass to
    -- begin.  (A low space condition will cause garbage collection to happen
    -- automatically.)  If this call starts a garbage collection pass, then
    -- This_Call_Did_A_Gc_Pass will be returned true, and the caller will wait
    -- until the pass is completed  (a potentially long time, since that
    -- involves lots of work).

    -- The garbage collection pass will be executed at the better of the
    -- specified collection priority and that determined by the priority
    -- policy function, defined below (with the Set_Priority_Policy
    -- operation).

    -- There is only 1 worker task.  This serializes garbage collection to
    -- help prevent anamolous situations in which the garbage collector runs
    -- for extraordinarily long periods of time.

    generic
        with procedure Put_Line (S : String);
    procedure Display_Current_Gc_State;

    -- Displays various pieces of gc state.

    function Garbage_Collector_Is_Running
                (Volume : Volume_Number) return Boolean;

    -- Returns true iff the garbage collector is running on the specified
    -- volume.

    function Garbage_Collector_Is_Running return Boolean;
```

```
-- Returns true iff the garbage collector is running.

function Jobs_Are_Stopped_By_Volume (Volume : Volume_Number) return Boolean;

-- Returns true iff all jobs were stopped because the specified Volume
-- reached the warning threshold.

function Jobs_Are_Stopped return Boolean;

-- Returns true iff all jobs were stopped because some volume reached the
-- warning threshold.

function Start_Time return Calendar.Time;
-- Returns 'first when gc is not running.

type Collection_Phase is (Idle, Waiting_For_Backup_To_Finish,
                          Taking_Snapshot, Deleting_Segments,
                          Traversing_Virtual_Memory, Reclaiming_Blocks);

function Current_Phase return Collection_Phase;
-- Returns Idle when gc is not running.

function Current_Priority (Volume : Volume_Number)
                          return Collection_Priority;

-- If the Volume does not exist, or the garbage collector for that volume
-- has never run, returns worst priority (-1).  If the garbage collector is
-- currently running, returns the value that it is running at; otherwise
-- returns the value it last ran at.

procedure Set_Current_Priority (Volume           : Volume_Number;
                                Desired_Priority : Collection_Priority);

-- If the garbage collector is currently running in the
-- Traversing_Virtual_Memory phase, this operation will change the rate
-- at which its running, otherwise has no effect.

-- Note that there is a policy function that places a lower bound on the
-- current collection priority.  (See the Set_Priority_Policy operation,
-- below.)  If the specified priority is below that specified by the
-- priority policy, the Set_Current_Priority operation is a noop.

function Reclaimed_Segments return Natural;

-- Returns the number of segments which were deleted by gc.
-- These values are set to 0 when gc starts.  Don't expect them to
-- become non-zero until Current_Phase is >= Deleting_Segments.

function Visited_Segments    return Natural;
```





```
function Visited_Blocks      return Natural;
function Time_Spent_Delaying return Duration;

-- The first 2 functions return the number of segments/blocks which have
-- been "visited" by the virtual memory traversal.  The 3rd function
-- returns an approximate value for how much time gc has spent backing off,
-- as a result of the load parameters.  These values are set to 0 when
-- gc starts.  Don't expect them to become non-zero until Current_Phase is
-- >= Traversing_Virtual_Memory.

function  Stop_Run_Load return Natural;  -- default: 250
procedure Set_Stop_Run_Load (Run_Load : Natural);

function  Stop_Withheld_Load return Natural;  -- default: 1
procedure Set_Stop_Withheld_Load (Withheld_Load : Natural);

function  Start_Run_Load return Natural;  -- default: 125
procedure Set_Start_Run_Load (Run_Load : Natural);

function  Start_Withheld_Load return Natural;  -- 0
procedure Set_Start_Withheld_Load (Withheld_Load : Natural);

procedure Use_Standard_Backoff_Algorithm;

-- During the Traversing_Virtual_Memory phase,the collector follows
-- roughly the following algorithm:
--    loop
--        do about 500 milliseconds of work, and a few disk accesses
--        if (Run_Load > Stop_Run_Load) or
--           (Withheld_Load > Stop_Withheld_Load) then
--            loop
--                delay 30.0; -- seconds
--                exit when (Run_Load < Start_Run_Load) and
--                          (Withheld_Load < Start_Withheld_Load);
--            end loop;
--        end if;
--    end loop;
-- The loads for the stop tests are sampled over the last minute.  The
-- load for the restart tests are sampled over the last 5 minutes.

-- The values supplied to the Set_ operations should be MTS loads,
-- multiplied by 100. For example, 125 means an MTS load of 1.25.

-- The Use_Standard_Backoff_Algorithm procedure will revert the collector
-- to using its built in backoff algorithm.

function  Footprinting_Enabled return Boolean;
procedure Set_Footprinting (Desired_Value : Boolean);
```

```
    -- When the system boots, footprinting is disabled. True means footprinting
    -- is enabled.  When footprinting is enabled, reclaimed disk blocks are
    -- cleared.  This lengthens the amount of time spent in the
    -- Reclaiming_Blocks phase, since 1 disk I/O is required for every
    -- block reclaimed.

    function  Backup_Killing_Enabled return Boolean;
    procedure Set_Backup_Killing (Desired_Value : Boolean);

    -- When the system boots, backup kill mode is enabled.  True means to
    -- enable backup kill mode.  In backup kill mode, if the garbage collector
    -- is caused to run (via any method, including direct call, daemon
    -- scheduling, or disk space thresholds) and a backup is in progress,
    -- the backup will be terminated.  Recall that this whole issue stems
    -- from the fact that the garbage collection mechanisms don't work in
    -- the case where backup has a retained snapshot which is earlier than
    -- the retained snapshot which the garbage collector is using.

    function  Prevent_Stop_By_Warning     (Job : Task_Vpids) return Boolean;
    procedure Set_Prevent_Stop_By_Warning (Desired_Value : Boolean);

    -- This mechanism allows individual jobs to prevent themselves from being
    -- stopped when the garbage collector reaches the warning threshold (and
    -- stops all jobs). By default, jobs 4 & 5 are registered as not being
    -- stopped; otherwise the compaction daemons (such as DDB) and the gc can
    -- get deadlocked. This mechanism is also used by backup to prevent a
    -- deadlock.


    function  Get_Priority_Policy
                (Raised_Count : Integer; Started_Count : Integer)
                return Collection_Priority;
    procedure Set_Priority_Policy (Raised_Count     : Integer;
                                   Started_Count    : Integer;
                                   Desired_Priority : Collection_Priority);

    -- There is a policy function which specifies the minimum allowable
    -- collection priority as a function of the number of volumes which have
    -- reached the Start_Collection threshold and the number of volumes which
    -- have reached the Raise_Priority threshold.  The default policy is:

    -- If there are 0 volumes past the Raise_Priority threshold, then the
    -- priority policy is the following function of the number of volumes
    -- past the Start_Collection threshold:
    --            1 volume  => Collection_Priority'(-1)
    --            2 volumes =>                      0
    --            3 volumes =>                      1
    -- For all remaining combinations, the policy is Collection_Priority'(2).
```





```
    -- The default policy is intended to have the following properties:
    -- When just a single volume wants to collect, let it run using
    -- spare cycles.  When more than one volume wants to collect, stop
    -- backing off.  When any Raise_Priority threshold is reached,
    -- raise priority such that the collection will preempt most background
    -- jobs.

    -- The set operation allows you to define your own priority policy. Hints
    -- for "rolling your own": (1) Setting the policy to return -1 for all
    -- inputs effectively disables the policy altogether. (2) The default
    -- policy does not increase the priority to 3 on the assumption that
    -- either (a) the mts parameters in effect limit the length of time an
    -- attached job can remain in the foreground or (b) that people do not
    -- typically leave attached jobs run for long periods of time. (3)
    -- Creating a policy which increases the priority to 5 or greater will
    -- probably have roughly the same impact as reaching the Stop_Jobs
    -- threshold.

end Disk_Daemon;
```

```
package Hash is

    -- simple hash functions to integer and long_integer
    -- all functions are guaranteed not to raise an exception

    generic
        type T   is limited private;
        type Ptr is access T;
    function Pointer_To_Integer (P : Ptr) return Integer;

    generic
        type T   is limited private;
        type Ptr is access T;
    function Pointer_To_Long_Integer (P : Ptr) return Long_Integer;

    generic
        type T   is limited private;
        type Ptr is access T;
        pragma Segmented_Heap (Ptr);
    function Heap_Pointer_To_Integer (P : Ptr) return Integer;

    generic
        type T   is limited private;
        type Ptr is access T;
        pragma Segmented_Heap (Ptr);
    function Heap_Pointer_To_Long_Integer (P : Ptr) return Long_Integer;

    function Long_Integer_To_Integer (Value : Long_Integer) return Integer;

end Hash;
```





```
with Directory;

package Library_Object_Editor is

    -- Cause the specified library to be displayed.  If selection is not
    -- nil, then that object will be selected as the library is displayed.
    -- If Force_Redraw is true, the image will be redrawn even if it
    -- already exists.

    procedure Display (Library      : Directory.Object;
                       Selection    : Directory.Object := Directory.Nil;
                       In_Place     : Boolean          := False;
                       Force_Redraw : Boolean          := False);

    procedure Update_Changed_Images;

    procedure Create_World (Name   : String  := ">>WORLD NAME<<";
                            Volume : Natural := 0);

    procedure Create_Directory (Name : String := ">>DIRECTORY NAME<<");

    procedure Create_Unit (Name : String := ">>ADA NAME<<");

end Library_Object_Editor;
```

```
with Diana;
with Directory;
with Links_Implementation;
with Io_Exceptions;

package Link_Tools is

    package Links renames Links_Implementation;

    subtype World_Name is String;
    This_World : constant World_Name := "$";

    -- The string name for the World associated with the current naming
    -- context.

    subtype Link_Name is String;

    -- An Ada simple name. When used as an in-parameter, except in Add and
    -- Replace, it may contain wildcard characters.  In Add and Replace it
    -- may contain substitution characters.

    subtype Source_Name is String;

    -- A directory string name that specifies an existing Ada Library Unit.
    -- (The unit does not have to be installed, but its declaration must be
    -- in a library.)  May contain wildcard characters when used as an
    -- in-parameter.

    subtype Link_Kind is Links.Link_Kind;

    -- If the source unit is in the same world as the link,
    -- the link is Internal; otherwise it is External.

    Any      : constant Link_Kind := Links.Any;
    External : constant Link_Kind := Links.External;
    Internal : constant Link_Kind := Links.Internal;

    function "=" (L, R : Link_Kind) return Boolean renames Links."=";

    Name_Error : exception renames Io_Exceptions.Name_Error;
    -- Raised if a World parameter cannot be resolved.

    Use_Error : exception renames Io_Exceptions.Use_Error;
    -- Raised of the link pack cannot be opened for other reasons.
```





```
    function Has (Link   : Link_Name   := "@";
                  Source : Source_Name := "?";
                  Kind   : Link_Kind   := Link_Tools.Any;
                  World  : World_Name  := This_World) return Boolean;

    -- Returns true iff the pack contains at least one link that matches the
    -- given link, source, and kind parameters.


    function Link (Source : Source_Name;
                   Kind   : Link_Kind  := Link_Tools.Any;
                   World  : World_Name := This_World) return Link_Name;

    function Link (Source : Diana.Tree;
                   Kind   : Link_Kind  := Link_Tools.Any;
                   World  : World_Name := This_World) return Link_Name;

    function Link (Source : Directory.Object;
                   Kind   : Link_Kind  := Link_Tools.Any;
                   World  : World_Name := This_World) return Link_Name;

    -- Given a Source_Name, Def_ID, or Directory.Object for an Ada unit,
    -- returns a link for that unit in the given world that matches the
    -- Kind parameter.  A null string is returned if no such link can be
    -- found.


    function Source (Link  : Link_Name;
                     Kind  : Link_Kind  := Link_Tools.Any;
                     World : World_Name := This_World) return Source_Name;

    function Source (Link  : Link_Name;
                     Kind  : Link_Kind  := Link_Tools.Any;
                     World : World_Name := This_World) return Diana.Tree;

    function Source (Link  : Link_Name;
                     Kind  : Link_Kind  := Link_Tools.Any;
                     World : World_Name := This_World) return Directory.Object;


    -- Returns either the Source_Name, Def_ID, or Directory.Object for an Ada
    -- unit corresponding to a link that matches the Link and Kind parameter.
    -- Returns a null object if no match can be found.

    type Dependent_Iterator is private;
```

```
    function Dependents (Link   : Link_Name   := "@";
                        Source : Source_Name := "?";
                        Kind   : Link_Kind   := Link_Tools.Any;
                        World  : World_Name  := This_World)
                      return Dependent_Iterator;

    procedure Next  (Iter : in out Dependent_Iterator);
    function  Value (Iter : Dependent_Iterator) return Diana.Tree;
    function  Done  (Iter : Dependent_Iterator) return Boolean;


    -- Computes the Library Units of the world that are installed or coded
    -- and reference any of the links specified by the Source, Link and Kind
    -- parameters.

    type Link_Iterator is private;

    procedure Init (Iter   : out Link_Iterator;
                    Source :     Source_Name := "?";
                    Link   :     Link_Name   := "@";
                    Kind   :     Link_Kind   := Link_Tools.Any;
                    World  :     World_Name  := This_World);

    procedure Next   (Iter : in out Link_Iterator);
    function  Link   (Iter : Link_Iterator) return Link_Name;
    function  Source (Iter : Link_Iterator) return Source_Name;
    function  Source (Iter : Link_Iterator) return Diana.Tree;
    function  Source (Iter : Link_Iterator) return Directory.Object;
    function  Kind   (Iter : Link_Iterator) return Link_Kind;
    function  Done   (Iter : Link_Iterator) return Boolean;

    -- For iterating over the Links in a link pack that match given Source
    -- and Link patterns

end Link_Tools;
```





```
generic
    type Element is private;
    -- must be a pure value
    -- ie. no initialization or finalization is necessary
    -- = and := are equality and copy

    pragma Must_Be_Constrained (Yes => Element);

package List_Generic is

    type List is private;
    -- may generate garbage
    -- = and := operate on references
    -- "make" constructs lists with structural sharing

    -- constraint error is raised when nil i provided to any of
    --    first, rest, set_first, or set_rest

    function Make (X : Element; L : List) return List;

    function Nil                     return List;
    function Is_Empty (L : List) return Boolean;

    procedure Free (L : in out List);
    -- make L empty

    function  First     (L : List) return Element;
    function  Rest      (L : List) return List;
    procedure Set_Rest  (L : List; To_Be : List);
    procedure Set_First (L : List; To_Be : Element);

    function Length (L : List) return Natural;

    type Iterator is private;

    procedure Init  (Iter : out Iterator; L : List);
    procedure Next  (Iter : in out Iterator);
    function  Value (Iter : Iterator) return Element;
    function  Done  (Iter : Iterator) return Boolean;

private

    type Listdata;
    type List is access Listdata;
    -- variables of type list are initialized to null
```

```
    type Listdata is
        record
            First : Element;
            Rest  : List;
        end record;

    type Iterator is new List;

end List_Generic;
```





```
with Action;
with Directory_Tools;
with Diana;
with Errors;
with Ada_Text;

package Ada_Program is

    --
    --    The use of this system is subject to the software license terms and
    --    conditions agreed upon between Rational and the Customer.
    --
    --               Copyright 1987, 1988, 1989, 1990 by Rational.
    --
    --                          RESTRICTED RIGHTS LEGEND
    --
    --    Use, duplication, or disclosure by the Government is subject to
    --    restrictions as set forth in subdivision (b)(3)(ii) of the Rights in
    --    Technical Data and Computer Software clause at 52.227-7013.
    --
    --                Rational
    --                3320 Scott Boulevard
    --                Santa Clara, California 95054
    --
    --   PROPRIETARY AND CONFIDENTIAL INFORMATION OF RATIONAL;
    --   USE OR COPYING WITHOUT EXPRESS WRITTEN AUTHORIZATION
    --   IS STRICTLY PROHIBITED.  THIS MATERIAL IS PROTECTED AS
    --   AN UNPUBLISHED WORK UNDER THE U.S. COPYRIGHT ACT OF
    --   1976.  CREATED 1987, 1988, 1989, 1990.  ALL RIGHTS RESERVED.
    --

    package Object renames Directory_Tools.Object;

    type Element is private;
    Nil_Element : constant Element;

    -- Ada programs are composed of a hierarchical structure
    -- of elements.  Operations are defined to determine what
    -- kind a particular element is and to decompose elements
    -- into sub-elements (children).
    --
    -- Each element may also have attributes that provide information
    -- about that element.  For example, all declaration elements will
    -- have a name (identifier) associated with them.  Certain static
    -- expressions may have a value associated with them.  Operations
    -- are defined to provide this information for specific kinds of
    -- elements.
```

```
-- If a query for a specific attribute is made to an inappropriate
-- kind of element the Inappropriate_Program_Element exception is raised.
--
-- Some elements "make reference to" other elements.
-- Declarations, for example, define named elements.
-- Other elements such as statements and expressions may make
-- reference to declarations.
--
-- Operations are available to find defining elements from
-- the elements that use those definitions.
--
-- All elements have an image, that is, a pretty printing of
-- that fragment of the program. Images of elements other than
-- top level kinds and IDs may not be very meaningful.
-- The images of these elements may be locked, or inaccessible
-- and so these operations may fail.
--
-- Comments may be isolated, adjacent to or attached to an element.
-- Comments that stand alone or are adjacent to an element
-- are generally intended to be related to some program element such
-- as a declaration or statement.  This relation must be derived from the
-- convention established in the program using this interface and is not
-- captured in the semantics of this interface.
--
-- Many operations in this interface traverse from one program tree to
-- another. If in this traversal a new Ada object must be opened, the
-- traversal may fail due to a lock or access error. In this case
-- the FAILED exception is raised.
--
-- When an operation fails for any reason (a defined exception
-- propagates out of the operation) the DIAGNOSIS and STATUS calls
-- may be made to find out why the operation failed.

pragma Page;

function Is_Nil (Program_Element : Element) return Boolean;
-- Some program elements have optional attributes or sub-elements.
-- In the case where an attribute or sub-element could, but does
-- does not actually exist, a nil element will be returned.

function Parent (Program_Element : Element) return Element;
-- Returns the immediate parent element of the specified element.
-- If the element is a compilation_unit, (see definition below and
-- package Compilation_Unit) a nil element is returned.

function Line_Number (Of_Element : Element) return Natural;
-- Returns the line number on which the element resides.
-- A nil element returns 0.
```





```
-- This operation uses the element's image.

procedure Definition (Of_Element :        Element;
                      In_Place   :        Boolean := False;
                      Edit       :        Boolean := False;
                      Status     : in out Errors.Condition);
--
-- Brings up an Ada object editor on the unit containing OF_ELEMENT.


------------------------------------------------------------------

type Element_Iterator is private; -- A (read only) ordered set of Elements.
Nil_Iterator : constant Element_Iterator;

procedure Next  (Iter : in out Element_Iterator);
function  Done  (Iter : Element_Iterator) return Boolean;
function  Value (Iter : Element_Iterator) return Element;

procedure Reset (Iter : in out Element_Iterator);
-- Resets the iterator to the beginning of the list.

------------------------------------------------------------------

type Element_List is private; -- A (read/write) ordered list of Elements.
Nil_List : constant Element_List;

-- Elements lists are used for collecting together lists of elements
-- during traversal.
-- Assignment on LISTs DOES NOT CAUSE A COPY TO BE MADE!
-- Use COPY to do that.

procedure Copy (From_Iter : Element_Iterator; To_List : out Element_List);
procedure Copy (From_List : Element_List; To_List : out Element_List);
-- The entire contents of FROM_xxx is copied regardless of the current
-- 'position' of FROM_xxx or TO_LIST.
-- The contents of TO_LIST are lost.

procedure Append  (Program_Element :        Element;
                   To_List         : in out Element_List);
procedure Prepend (Program_Element :        Element;
                   To_List         : in out Element_List);
procedure Append  (From_List :        Element_List;
                   To_List   : in out Element_List);
-- Append/Prepend to the end/beginning of TO_LIST.  The current position
-- of the list is undefined after these calls.
```

```
generic
    with function Discard (Program_Element : Element) return Boolean;
procedure Filter (Source_List :     Element_List;
                  Target_List : out Element_List);

generic
    with function Discard (Program_Element : Element) return Boolean;
procedure Filter_Iterator (Source_Iterator :     Element_Iterator;
                           Target_List     : out Element_List);

procedure Next  (List : in out Element_List);
function  Done  (List : Element_List) return Boolean;
function  Value (List : Element_List) return Element;

procedure Reset (List : in out Element_List);
-- Resets the list to the beginning.

procedure Invert (List : in out Element_List);
-- Reverse the ordering of the given element list.

------------------------------------------------------------------

type Traversal_Control is (Continue,
                           Abandon_Children,
                           Abandon_Siblings,
                           Terminate_Immediately);
generic
    type State_Record is private;
    with procedure Pre_Operation (Program_Element : Element;
                                  State : in out State_Record;
                                  Control : in out Traversal_Control);
    with procedure Post_Operation (Program_Element : Element;
                                   State : in out State_Record;
                                   Control : in out Traversal_Control);
procedure Depth_First_Traversal (Root_Element        : Element;
                                 State               : in out State_Record;
                                 Major_Elements_Only : Boolean := True);

-- Performs a depth-first traversal of Ada_Program elements rooted at
-- the given element.  If Major_Elements_Only is True, then only
-- MAJOR Ada_Program elements are visited  (see ELEMENT_KINDS enumeration
-- below)
--
-- For each element:
--   The formal procedure Pre_Operation is called when first visiting
--   the element.  All sub-elements are then visited and then the
--   Post_Operation procedure is called when returning from visiting all
--   sub_elements.  The State variable is passed from call to call.
```





```
--
-- Traversal can be controlled with the Control parameter.
-- The Abandon_Children option prevents traversal to the current element's
-- children, but picks up with the next sibling.
-- The Abandon_Sibling option abandons traversal through the
-- remaining siblings but continues traversal at the parent.
-- The Terminate_Immediately option does the obvious.
--
-- NOTES:
-- Abandon_Children in a POST_OPERATION is the same as CONTINUE (all
-- the children have already been visited).
-- Abandon_Siblings in a PRE_OPERATION skips the associated
-- POST_OPERATION.
------------------------------------------------------------------

type Line_Iterator is private;

subtype Line is String;

function  Done  (Iter : Line_Iterator) return Boolean;
function  Value (Iter : Line_Iterator) return Line;
procedure Next  (Iter : in out Line_Iterator);

function Image (Program_Element : Element) return Line_Iterator;
-- The image of a program element is made up of some number of lines.
-- Images can be iterated over to get each individual line.

function Image (Program_Element : Element) return String;
-- The image of a program element in a single string.  Lines are separated
-- by Ascii.Lf characters.

function Preceding_Comments (An_Element : Element) return Line_Iterator;
function Following_Comments (An_Element : Element) return Line_Iterator;
-- Returns the comments, if any, that appear before or after the specified
-- element (including blank lines) Non blank comment lines include "--"s.
-- This function is appropriate for major elements such as statements,
-- declarations, context clauses, and generally things that can appear on
-- on a line by themselves.  If no comments are present, a nil iterator is
-- returned.

function Internal_Comments (An_Element : Element) return Line_Iterator;
-- Returns the comments, if any, that appear attached to the internal
-- structure of an element. Examples of elements that have internal
-- structure are: Package Specs, Procedure Bodies.

function Attached_Comments (An_Element : Element) return Line_Iterator;
-- Returns the comments, if any, that are directly attached to
-- an element.  In the case that no comments exist, a "Done"
-- iterator will be returned.
```

```
-----------------------------------------------------------------
-- MAJOR program elements:

type Element_Kinds is (A_Compilation_Unit,
                       A_Context_Clause,
                       A_Declaration,
                       A_Statement,
                       A_Pragma,
                       A_Representation_Clause,
                       Not_A_Major_Element);

function Kind (Program_Element : Element) return Element_Kinds;
-- Once the KIND of an element is determined, further decomposition
-- or selection can be done by calling functions in the package that
-- deals with a specific element kind. (e.g. the COMPILATION_UNITS
-- package for kind A_COMPILATION_UNIT)

subtype Association             is Element;
subtype Choice                  is Element;
subtype Compilation_Unit        is Element;
subtype Context_Clause          is Element;
subtype Declaration             is Element;
subtype Expression              is Element;
subtype Name                    is Element;
subtype Pragma_Usage            is Element;
subtype Representation_Clause   is Element;
subtype Statement               is Element;
subtype Type_Definition         is Element;

subtype Association_Iterator              is Element_Iterator;
subtype Choice_Iterator                   is Element_Iterator;
subtype Compilation_Unit_Iterator         is Element_Iterator;
subtype Context_Clause_Or_Pragma_Iterator is Element_Iterator;
subtype
Declaration_Or_Context_Clause_Or_Representation_Clause_Or_Pragma_Iterator
    is Element_Iterator;
subtype Expression_Iterator               is Element_Iterator;
subtype Name_Iterator                     is Element_Iterator;
subtype Pragma_Iterator                   is Element_Iterator;
subtype Representation_Clause_Iterator    is Element_Iterator;
subtype Statement_Or_Pragma_Iterator      is Element_Iterator;
subtype Type_Definition_Iterator          is Element_Iterator;
-- Note that some of the iterators can mix items of different major
-- kinds.  Their name attempts to convey this information.  For
-- instance a declarative part can contain, besides declarations,
-- context clauses (viz. use clauses), representation clauses or
-- pragmas.
```





```
-----------------------------------------------------------------
-- IDENTIFIERS:  -- LRM 2.3

subtype Identifier_Definition is Element;
subtype Identifier_Reference  is Element;

-- The image of all Identifier_Definitions and Identifier_References
-- will provide the string name.

type Id_Kinds is (An_Identifier_Definition,
                  An_Identifier_Reference,
                  Not_An_Identifier);

function Id_Kind (An_Identifier : Element) return Id_Kinds;

function Definition (Reference : Element; Visible : Boolean := True)
                    return Identifier_Definition;
-- This call follows the ADA OBJECT EDITOR definition model.
-- The parameter VISIBLE indicates a preference. It may be that
-- the returned definition is not visible.

function Usage (Reference : Element;
                Global    : Boolean := True;
                Limit     : String  := "<ALL_WORLDS>";
                Closure   : Boolean := False) return Element_List;
-- This call follows the ADA OBJECT EDITOR show usage model.

function Other_Part (Reference : Element) return Identifier_Definition;
-- Returns the other part of the given reference.  If the given
-- reference has no other part, it returns Nil_Element. This call
-- follows the ADA OBJECT EDITOR other part model.

function String_Name (An_Identifier : Element) return String;


-----------------------------------------------------------------
-- PROMPTS:
--

subtype Prompt is Element;
function Is_Prompt (An_Element : Element) return Boolean;

type Prompt_Kinds is (An_Alternative_Prompt,
                      A_Compilation_Unit_Prompt,
                      A_Context_Clause_Prompt,
                      A_Declaration_Prompt,
                      An_Expression_Prompt,
                      A_Generic_Parameter_Prompt,
                      An_Identifier_Prompt,
```

```
                  A_Pragma_Prompt,
                  A_Statement_Prompt,
                  Not_A_Prompt);

function Prompt_Kind (A_Prompt : Element) return Prompt_Kinds;


-----------------------------------------------------------------

package Conversion is

    function Normalize (Tree : Diana.Tree) return Element;
    -- Given an arbitrary diana tree find the closest corresponding
    -- ELEMENT. (This routine may walk UP a diana tree);

    function Convert (A_Tree : Diana.Tree)  return Element;
    function Convert (An_Element : Element) return Diana.Tree;

    procedure Register_Action (Action_Id : Action.Id);
    -- Once an action is registered, all "opens" will be performed
    -- under the specified action.  Opens can be implicitly performed
    -- when one traverses to definitions located in other objects
    -- or if one accesses the element's image.

    procedure Close_All_Objects;
    -- Closes all objects opened by the current job.  This operation
    -- DOES NOT affect the Actions associated with the current job.

    procedure Finish_Action;
    -- Closes all objects opened under the previously registered
    -- action or under the default action.  This operation is similar
    -- to CLOSE_ALL_OBJECTS except that the currently registered
    -- action is also committed.

    -- Construction of iterators:

    function Build_Element_Iterator
                (Sequence : Diana.Sequence) return Element_Iterator;
    function Build_Element_Iterator
                (Seq_Type : Diana.Seq_Type) return Element_Iterator;

    -- Conversion functions to and from directory object handles and names.

    function To_Directory_Object
                (Comp_Unit : Compilation_Unit) return Object.Handle;

    function To_Compilation_Unit (Directory_Object : Object.Handle;
                                  Action_Id : Action.Id := Action.Null_Id)
                                 return Compilation_Unit;
```





```
    -- If the default Null_Id is provided, the currently registered
    -- action will be used for this and all subsequent opens.
    -- If no action has previously been registered, then one will be
    -- constructed.
    -- If a non-null action is specified it will be used for this open
    -- and registered for all subsequent opens.

    function Resolve (Element_Name : String) return Element;
    -- Does the best it can to resolve an unambiguous name of an
    -- element to it's internal form. See notes below for GET_NAME.

    function Resolve (Element_Names      : String;
                      Visible            : Boolean := True;
                      Look_Through_Stubs : Boolean := True)
                     return Element_List;
    --
    -- Resolves any name to a list of one or more Ada_Program
    -- Elements. This form of RESOLVE behaves in a way similar to
    -- COMMON.DEFINITION.  When ELEMENT_NAMES resolves to multiple
    -- units/declarations, the VISIBLE parameter has no effect.  If
    -- the name resolves to a subunit, the stub is returned if
    -- Look_Through_Stubs is False, and the subunit if it is True.

    -- This procedure is declared at the end of this package
    -- to maintain compatibility.
    --
    -- procedure Resolve (Element_Names : String;
    --                    Result        : out Element_List;
    --                    Status        : in out Errors.Condition;
    --                    Context       : Directory_Tools.Naming.Context :=
    --                       Directory_Tools.Naming.Default_Context;
    --                    Objects_Only  : Boolean := False);
    --
    -- Resolves (ambiguous) naming expression in the given context.
    -- This operation is similar in behavior to
    -- Directory_Tools.Naming.Resolution'N(2).  If Objects_Only is
    -- true, only library level objects that match the name will be
    -- included; when false, Ada_Program Elements will be included
    -- in RESULT even if no separate directory object is associated
    -- with them.

    function Get_Name (Of_Element : Element) return String;
    -- Does the best it can to give a fully resolved name of the
    -- element, this works well for declarations and comp_units but
    -- may not give useful results for other kinds of elements.
    -- If it succeeds, the string may be used in RESOLVE to convert back
    -- to the originating ELEMENT
```

```
   -----------------------------------

   function Handle_Of (The_Element : Element) return Ada_Text.Handle;
   -- Returns the currently open handle for the Ada image containing
   -- THE_ELEMENT.  If no handle exists, one will be opened and
   -- future calls will return that handle.

   -- ELEMENTs cannot be stored in permanent objects
   -- like files. Use the following operations to generate an external
   -- representation that can be saved and converted.

   type Element_Permanent_Representation is new String;

   function Convert (An_Element : Element;
                     Within     : String := "<SUBSYSTEM>")
                    return Element_Permanent_Representation;
   -- The WITHIN parameter specifies how fully qualified the
   -- representation is (or should be resolved). When converting from
   -- an element to a permanent representation, the values can be :
   --   <FULL>          - The element's fully qualified resolution is
   --                     returned.
   --   <SUBSYSTEM>     - The resolution of the element is subsystem
   --                     relative. Subsystem name and spec or load origin
   --                     is preserved.
   --   <VIEW>          - The resolution of the element is view relative.
   --                     No origin information is preserved.
   --
   -- Storage for <SUBSYSTEM> or <VIEW> representation are less than
   -- <FULL> but some origin information is lost.
   --

   procedure Convert (An_Element_Rep : Element_Permanent_Representation;
                      Result         : out Element;
                      Status         : in out Errors.Condition;
                      Within         : String := "<DEFAULT>");
   -- When converting from a permanent representation to an element,
   -- an attempt is made to fill in any missing information in the
   -- permanent representation from the WITHIN parameter. So, in all
   -- cases, if the permanent representation was FULLy resolved the
   -- WITHIN parameter is ignored, otherwise :
   --   <DEFAULT>       - For <SUBSYSTEM>, the view selected by
   --                     the current activity for the stored subsystem is
   --                     used as the origin. The spec or load origin of the
   --                     representation is used to pick the specific view.
   --                     For <VIEW>, the current view context is used as
   --                     the origin.
   --   subsystem name - For <SUBSYSTEM>, same as <DEFAULT>.
   --                     For <VIEW>, the view selected by the current
```





```
   --                     activity for the specified subsystem is used.
   --                     The subsystem name can be followed by "'SPEC" or
   --                     "'LOAD" to force use of the spec view or load
   --                     view from the activity. (Defaults to Spec View)
   --   view name       - For <SUBSYSTEM>, if the subsystem containing the
   --                     specified view matches the representation's
   --                     subsystem, the specified view is used.
   --                     If no match, same as <DEFAULT>.
   --                     For <VIEW> the specified view is used.

   function Unique_Id (For_Element : Element) return Long_Integer;
   -- Generates a unique number for any element.
   -- Useful for building maps for elements.

   -- This procedure is provided here to maintain compatibility.
   --
   procedure Resolve (Element_Names : String;
                      Result        : out Element_List;
                      Status        : in out Errors.Condition;
                      Context       : Directory_Tools.Naming.Context :=
                         Directory_Tools.Naming.Default_Context;
                      Objects_Only  : Boolean := False);
   --
   -- Resolves (ambiguous) naming expression in the given context.
   -- This operation is similar in behavior to
   -- Directory_Tools.Naming.Resolution'N(2).  If Objects_Only is
   -- true, only library level objects that match the name will be
   -- included; when false, Ada_Program Elements will be included
   -- in RESULT even if no separate directory object is associated
   -- with them.

end Conversion;

Inappropriate_Program_Element : exception;
-- Raised when an operation is applied to an inappropriate
-- program element.
-- Use the DIAGNOSIS or STATUS calls to get more info.

Failed : exception;
-- Catch-all exception raised when an operation fails for reasons other
-- than the above inappropriate element reason.
-- Use the DIAGNOSIS or STATUS calls to get more info.

function Diagnosis return String;
function Status    return Errors.Condition;
-- Whenever an error condition is detected (and exception is raised)
-- a diagnostic message/status is stored. These functions retrieve the
-- diagnostic for the most recent error.
```

```
    -- The following provide debugger image functions for the
    -- private types defined in this package.

    function Debug_Image (Of_Element      : Element;
                          Level           : Natural;
                          Prefix          : String;
                          Expand_Pointers : Boolean) return String;

    function Debug_Image (Of_Iterator     : Element_Iterator;
                          Level           : Natural;
                          Prefix          : String;
                          Expand_Pointers : Boolean) return String;

    function Debug_Image (Of_List         : Element_List;
                          Level           : Natural;
                          Prefix          : String;
                          Expand_Pointers : Boolean) return String;

    function Debug_Image (Of_Lines        : Line_Iterator;
                          Level           : Natural;
                          Prefix          : String;
                          Expand_Pointers : Boolean) return String;


    --
    -- Hide these interfaces ...
    --

    procedure Field_Copy (From_List :        Element_List;
                          To_List   : in out Element_List);
    -- if a data type contains a LIST and that data type is
    --   stored in a segmented heap and a selector on that pointed
    --   to object returns a LIST, use this to get the LIST out, it
    --   will un-normalize the internal segmented heap pointers.

private
    type Element is new Diana.Tree;
    Nil_Element : constant Element := Element (Diana.Empty);


    -- ELEMENT_ITERATORs are read only

    type Iterator_Kinds is (Sequence, Seq_Type);

    type Element_Iterator is
        record
            Kind                              : Iterator_Kinds;
            Sequence_Root, Sequence_Current : Diana.Sequence;
            Seq_Type_Root, Seq_Type_Current : Diana.Seq_Type;
        end record;
```





```
    Nil_Iterator : constant Element_Iterator :=
       Element_Iterator'(Kind             => Sequence,
                         Sequence_Root    => Diana.Sequence'(Diana.Make),
                         Sequence_Current => Diana.Sequence'(Diana.Make),
                         Seq_Type_Root    => Diana.Seq_Type'(Diana.Make),
                         Seq_Type_Current => Diana.Seq_Type'(Diana.Make));


    -- ELEMENT_LISTs are read/write
    type Element_List is
        record
            First, Current, Last : Diana.Temp_Seq;
        end record;

    Nil_List : constant Element_List :=
       Element_List'(First   => Diana.Temp_Seq'(Diana.Make),
                     Current => Diana.Temp_Seq'(Diana.Make),
                     Last    => Diana.Temp_Seq'(Diana.Make));

    type Line_Iterator is new Ada_Text.Iterator;

end Ada_Program;
```

```
with Ada_Program;

package Associations is

    --
    --    The use of this system is subject to the software license terms and
    --    conditions agreed upon between Rational and the Customer.
    --
    --               Copyright 1987, 1988, 1989, 1990 by Rational.
    --
    --                          RESTRICTED RIGHTS LEGEND
    --
    --    Use, duplication, or disclosure by the Government is subject to
    --    restrictions as set forth in subdivision (b)(3)(ii) of the Rights in
    --    Technical Data and Computer Software clause at 52.227-7013.
    --
    --
    --                Rational
    --                3320 Scott Boulevard
    --                Santa Clara, California 95054
    --
    --   PROPRIETARY AND CONFIDENTIAL INFORMATION OF RATIONAL;
    --   USE OR COPYING WITHOUT EXPRESS WRITTEN AUTHORIZATION
    --   IS STRICTLY PROHIBITED.  THIS MATERIAL IS PROTECTED AS
    --   AN UNPUBLISHED WORK UNDER THE U.S. COPYRIGHT ACT OF
    --   1976.  CREATED 1987, 1988, 1989, 1990.  ALL RIGHTS RESERVED.
    --


    -- LRM 3.7.2, 6.4.1 and 12.3
    -- This package provides operations on argument associations,
    -- generic associations and parameter associations.

    -- Local Renamings:
    subtype Association is Ada_Program.Association;
    subtype Identifier_Definition is Ada_Program.Identifier_Definition;
    subtype Name_Expression is Ada_Program.Expression;
    -------------------------------------------------------------------

    type Association_Kinds is (Named_Association,
                               Positional_Association,
                               Defaulted,
                               Not_An_Association);

    function Association_Kind
                (An_Association : Association) return Association_Kinds;
    -- Returns the kind of an association.
```





```
    function Formal_Parameter (An_Association : Association)
                                    return Identifier_Definition;
    -- Returns the identifier of the formal name for the given
    -- association.  This function tries hard to return an identifier
    -- definition.  However, in the case of a pragma argument, only a
    -- reference can be returned.

    function Actual_Parameter
                (An_Association : Association) return Name_Expression;
    -- Returns the actual name or expression for the given association.

end Associations;
```

```
with Ada_Program;

package Compilation_Units is

    --
    --   The use of this system is subject to the software license terms and
    --   conditions agreed upon between Rational and the Customer.
    --
    --              Copyright 1987, 1988, 1989, 1990 by Rational.
    --
    --                         RESTRICTED RIGHTS LEGEND
    --
    --   Use, duplication, or disclosure by the Government is subject to
    --   restrictions as set forth in subdivision (b)(3)(ii) of the Rights in
    --   Technical Data and Computer Software clause at 52.227-7013.
    --
    --
    --              Rational
    --              3320 Scott Boulevard
    --              Santa Clara, California 95054
    --
    --   PROPRIETARY AND CONFIDENTIAL INFORMATION OF RATIONAL;
    --   USE OR COPYING WITHOUT EXPRESS WRITTEN AUTHORIZATION
    --   IS STRICTLY PROHIBITED.  THIS MATERIAL IS PROTECTED AS
    --   AN UNPUBLISHED WORK UNDER THE U.S. COPYRIGHT ACT OF
    --   1976.  CREATED 1987, 1988, 1989, 1990.  ALL RIGHTS RESERVED.
    --

    -- LRM 10.1
    --
    -- Compilation units are composed of two distinct parts:
    -- A context clause part and a declarative part.
    -- The context clause part may contain actual context clauses
    -- (with and use clauses) and pragmas.
    -- The declaration associated with a compilation unit can either
    -- be a package, procedure, function, or subunit.

    -- Local Renamings:
    subtype Compilation_Unit is Ada_Program.Compilation_Unit;
    subtype Context_Clause   is Ada_Program.Context_Clause;
    subtype Declaration      is Ada_Program.Declaration;

    subtype Context_Clause_Or_Pragma_Iterator is
       Ada_Program.Context_Clause_Or_Pragma_Iterator;
    subtype Name_Iterator                   is Ada_Program.Name_Iterator;
    subtype Pragma_Iterator                 is Ada_Program.Pragma_Iterator;
```





```
    ---------------------------------------------------------------

    function Unit_Declaration
                (A_Compilation_Unit : Compilation_Unit) return Declaration;
    -- Returns the package, procedure, or function declaration of the
    -- compilation unit.  (Library and Secondary units as defined in the
    -- LRM.  The operations on declarations can be used to further
    -- decompose these elements.

    function Context_Clause_Elements (Of_Compilation_Unit : Compilation_Unit)
                                 return Context_Clause_Or_Pragma_Iterator;
    -- Returns a list of context clauses and pragmas that
    -- reside in the context part of the compilation unit.

    -- Context clauses are made up of one or more unit references.
    --
    -- ie  with Bar;        -- has one unit reference
    --     use Foo, Bar;    -- has two named unit references
    --
    -- All context clauses have an Image but only their elements
    -- have a name.

    type Context_Clause_Kinds is
       (A_With_Clause, A_Use_Clause, Not_A_Context_Clause);

    function Context_Clause_Kind
                (A_Context_Clause : Context_Clause) return Context_Clause_Kinds;

    function Referenced_Units
                (A_Context_Clause : Context_Clause) return Name_Iterator;
    -- Returns a list of identifier references in a context clause

    function Parent_Compilation_Unit (Of_Program_Element : Ada_Program.Element)
                                 return Compilation_Unit;
    -- Returns the compilation unit that contains a particular
    -- program element. If the PROGRAM_ELEMENT in question is a
    -- compilation unit then this is an identity function.

    function Is_Subunit (A_Compilation_Unit : Compilation_Unit) return Boolean;
    -- Returns true if the compilation unit is a package, subprogram, or task
    -- subunit.

    function Subunit_Parent (Of_Subunit : Compilation_Unit)
                         return Compilation_Unit;
    -- Returns the compilation unit that is the parent unit of the subunit.

    function Body_Stub (Of_Subunit : Compilation_Unit) return Declaration;
    -- Returns the subunit declaration in the parent.
```

```
    function Attached_Pragmas
                (To_Compilation_Unit : Compilation_Unit) return Pragma_Iterator;
    -- Returns the list of pragmas attached to a compilation unit.  Only
    -- those pragmas that follow the compilation unit are returned here.
    -- Pragmas that precede the compilation unit are part of the context
    -- clause.

    function Is_Main_Program
                (Procedure_Or_Function : Compilation_Unit) return Boolean;
    -- Returns true if the unit has a pragma Main attached.

    subtype Name_Expression is Ada_Program.Expression;

    function Parent_Unit_Name
                (Of_Subunit : Compilation_Unit) return Name_Expression;
    -- Returns a name expression describing the parent unit.

end Compilation_Units;
```





```
with Ada_Program;

package Declarations is

    --
    --    The use of this system is subject to the software license terms and
    --    conditions agreed upon between Rational and the Customer.
    --
    --                 Copyright 1987, 1988, 1989, 1990 by Rational.
    --
    --                               RESTRICTED RIGHTS LEGEND
    --
    --    Use, duplication, or disclosure by the Government is subject to
    --    restrictions as set forth in subdivision (b)(3)(ii) of the Rights in
    --    Technical Data and Computer Software clause at 52.227-7013.
    --
    --
    --                 Rational
    --                 3320 Scott Boulevard
    --                 Santa Clara, California 95054
    --
    --   PROPRIETARY AND CONFIDENTIAL INFORMATION OF RATIONAL;
    --   USE OR COPYING WITHOUT EXPRESS WRITTEN AUTHORIZATION
    --   IS STRICTLY PROHIBITED.  THIS MATERIAL IS PROTECTED AS
    --   AN UNPUBLISHED WORK UNDER THE U.S. COPYRIGHT ACT OF
    --   1976.  CREATED 1987, 1988, 1989, 1990.  ALL RIGHTS RESERVED.
    --


    -- LRM Chapter 3

    -- Local Renaming:

    subtype Compilation_Unit is Ada_Program.Compilation_Unit;
    subtype Declaration      is Ada_Program.Declaration;
    subtype Expression       is Ada_Program.Expression;
    subtype Name_Expression  is Ada_Program.Name;
    subtype Statement        is Ada_Program.Statement;
    subtype Type_Definition  is Ada_Program.Type_Definition;

    subtype Association_Iterator      is Ada_Program.Association_Iterator;
    subtype Declarative_Part_Iterator is
       Ada_Program.
Declaration_Or_Context_Clause_Or_Representation_Clause_Or_Pragma_Iterator;
    ----------------------------------------------------------------
```

```
type Declaration_Kinds is (A_Variable_Declaration,
                           A_Constant_Declaration,
                           A_Deferred_Constant_Declaration,
                           An_Integer_Number_Declaration,
                           A_Real_Number_Declaration,

                           A_Type_Declaration,
                           An_Incomplete_Type_Declaration,
                           A_Subtype_Declaration,

                           A_Package_Declaration,
                           A_Package_Body_Declaration,

                           A_Procedure_Declaration,
                           A_Procedure_Body_Declaration,

                           A_Function_Declaration,
                           A_Function_Body_Declaration,

                           A_Package_Rename_Declaration,
                           A_Procedure_Rename_Declaration,
                           A_Function_Rename_Declaration,
                           An_Object_Rename_Declaration,
                           An_Exception_Rename_Declaration,

                           A_Generic_Package_Declaration,
                           A_Generic_Procedure_Declaration,
                           A_Generic_Function_Declaration,

                           A_Package_Instantiation,
                           A_Procedure_Instantiation,
                           A_Function_Instantiation,

                           A_Task_Declaration,
                           A_Task_Body_Declaration,
                           A_Task_Type_Declaration,

                           An_Entry_Declaration,

                           An_Exception_Declaration,

                           A_Procedure_Subunit,
                           A_Function_Subunit,
                           A_Package_Subunit,
                           A_Task_Subunit,

                           A_Subprogram_Formal_Parameter,
                           A_Generic_Formal_Parameter,
```





```
                           A_Discriminant,
                           A_Record_Component,

                           Not_A_Declaration);

function Kind (A_Declaration : Declaration) return Declaration_Kinds;

subtype Object_Declaration    is Declaration;  -- LRM 3.2.1
subtype Type_Declaration      is Declaration;  -- LRM 3.3.1
subtype Subtype_Declaration   is Declaration;  -- LRM 3.3.2
subtype Package_Declaration   is Declaration;  -- LRM 7.1
subtype Procedure_Declaration is Declaration;  -- LRM 6.1
subtype Function_Declaration  is Declaration;  -- LRM 6.1
subtype Task_Declaration      is Declaration;  -- LRM 9.1
subtype Task_Specification    is Type_Definition;  -- LRM 9.2
subtype Entry_Declaration     is Declaration;  -- LRM 9.5
subtype Exception_Declaration is Declaration;  -- LRM 11.1

function Is_Visible (A_Declaration : Declaration) return Boolean;
-- Returns True for units that are library level package or subprogram
-- specs, and for declarations inside library level package specs.

function Is_In_Private_Part (A_Declaration : Declaration) return Boolean;
-- Returns true is the declaration appears in the private part
-- of a package.

function Is_Generic_Formal (Element : Ada_Program.Element) return Boolean;
-- Returns true if a subprogram, type or object declaration is
-- a generic formal parameter or was derived from the GENERIC_PARAMETERS
-- selector.

function Is_Subprogram_Formal
            (Element : Ada_Program.Element) return Boolean;
-- Returns true if the element was derived from the
-- SUBPROGRAM_PARAMETERS selector.

function Identifiers (A_Declaration : Declaration)
                     return Ada_Program.Element_List;
-- Returns a list of the identifier(s) introduced by the declaration.

function Name (A_Declaration : Declaration) return String;
-- Returns the identifier(s) image. Basically the same as calling
-- ADA_PROGRAM.STRING_NAME on the result of IDENTIFIERS for a
-- declaration.

function Enclosing_Declaration
            (Element : Ada_Program.Element) return Declaration;
-- Returns the enclosing declaration for any element.
-- NOTE: This is the identity function if given a declaration.
```

```
---------------------------------------------------------------
-- OBJECT DECLARATIONS - LRM 3.2.1

-- The following operations apply to object declarations, component
-- declarations and discriminant specifications.

function Is_Initialized (Object_Decl : Object_Declaration) return Boolean;
-- Determines whether the object declaration includes an initial
-- value.

function Initial_Value (Object_Decl : Object_Declaration) return Expression;
-- Returns the optional expression that initializes an object or number
-- declaration, NIL_ELEMENT is otherwise returned.

function Object_Type (Object_Declaration_Or_Id : Object_Declaration)
                      return Type_Definition;
-- Returns the subtype indication following the colon in the object
-- declaration.  This function will take either a declaration element
-- or identifier definition associated with the declaration.

function Type_Mark (Subprogram_Formal_Or_Deferred_Constant : Declaration)
                    return Name_Expression;
--
-- Returns the type mark associated with Subprogram Formal
-- parameters and Deferred Constants.  Selectors in
-- NAMES_AND_EXPRESSIONS can then be used to extract more information.


---------------------------------------------------------------
-- TYPE DECLARATIONS - LRM 3.3

function Is_Private    (Type_Decl : Type_Declaration) return Boolean;
function Is_Limited    (Type_Decl : Type_Declaration) return Boolean;
function Is_Incomplete (Type_Decl : Type_Declaration) return Boolean;

function Type_Specification
            (Type_Declaration_Or_Id : Declaration) return Type_Definition;
-- Returns the type definition for a given type or subtype declaration
-- or the identifier definition associated with the declaration.
-- See the enumeration TYPE_INFORMATION.TYPE_DEFINITION_KINDS for
-- the various kinds of type definitions.

function Full_Type_Declaration
            (Type_Declaration_Or_Id : Declaration) return Type_Declaration;
-- Given the declaration of an incomplete type, returns the
-- corresponding full type declaration.  A nil element is returned
-- if the full type declaration is not yet compiled.  NOTE: this is
-- the identity function if given a non-incomplete type declaration.
```





```
---------------------------------------------------------------
-- PROGRAM UNIT DECLARATIONS
-- LRM Chapters 5, 6, 9

-- Program units are package, subprogram, and task specifications
-- and bodies.

function Is_Spec (Program_Unit : Declaration) return Boolean;
-- Determines whether a package, procedure, function, or task
-- is a specification.

function Is_Package    (A_Declaration : Declaration) return Boolean;
function Is_Task       (A_Declaration : Declaration) return Boolean;
function Is_Procedure  (A_Declaration : Declaration) return Boolean;
function Is_Function   (A_Declaration : Declaration) return Boolean;
function Is_Subprogram (A_Declaration : Declaration) return Boolean;
-- Determines if a declaration is a particular kind of program unit
-- declaration regardless if it is generic, an instantiation, rename,
-- spec or body.

function Specification (Decl_Or_Id : Declaration) return Declaration;
-- Returns the specification of a program unit declaration or identifier.
-- If a specification is provided the same element is returned.  If no
-- no specification exists a nil element is returned.

function Unit_Body (Decl_Or_Id : Declaration) return Declaration;
-- Returns the body for a given program unit declaration or identifier
-- definition associated with the declaration.  If a body is input,
-- the same element is returned.  If no body exists a nil element is
-- returned.  If a stub is given, the subunit is returned.


---------------------------------------------------------------
-- PACKAGES
-- LRM Chapter 7

function Visible_Part_Declarations
            (Package_Specification : Package_Declaration)
            return Declarative_Part_Iterator;
-- Returns a list of all declarations, representation
-- specifications, and pragmas in the visible part of a package in
-- the order of appearance.  When applied to a package
-- instantiation, this operation yields the instance's visible
-- declarations.

function Private_Part_Declarations
            (Package_Specification : Package_Declaration)
            return Declarative_Part_Iterator;
-- Returns a list of all declarations, representation
```

```
-- specifications, and pragmas in the private part of the package in
-- order of appearance.  When applied to a package instantiation,
-- this operation yields the instance's private declarations.

function Package_Body_Block
            (Package_Body : Package_Declaration) return Statement;
-- Returns the block statement for a package body including
-- the declarative part, any elaboration statements, and
-- exception handler if any.  The selectors for blocks in STATEMENTS
-- can then be used for further decomposition.


---------------------------------------------------------------
-- SUBPROGRAMS

subtype Subprogram_Formal_Parameter          is Ada_Program.Element;
subtype Subprogram_Formal_Parameter_Iterator is
   Ada_Program.Element_Iterator;

type Subprogram_Parameter_Kinds is (Default_In_Parameter, In_Parameter,
                                    Out_Parameter,
                                    In_Out_Parameter,
                                    Not_A_Parameter);

function Subprogram_Parameter_Kind
            (Of_Parameter : Subprogram_Formal_Parameter)
            return Subprogram_Parameter_Kinds;

function Subprogram_Parameters (Subprogram_Or_Entry : Declaration)
                               return Subprogram_Formal_Parameter_Iterator;
-- Returns an ordered list of formal parameter declarations.
-- Use IS_INITIALIZED and INITIAL_VALUE to query
-- the information related to the presence of the default parameter
-- initialization, and use TYPE_MARK to obtain the parameter type mark.
-- When applied to a subprogram instantiation, this operation yields
-- the instance's parameters.

function Return_Type (Of_Function : Function_Declaration)
                     return Name_Expression;
-- Returns the name expression of the return type, selectors in
-- NAMES_AND_EXPRESSIONS can then be used to extract more information.
-- When applied to a function instantiation, this operation yields
-- the instance's return type.

function Is_Operator_Definition
            (Function_Declaration : Declaration) return Boolean;

function Subprogram_Block (Subprogram_Body : Declaration) return Statement;
-- Returns the block statement for the body including
```





```
-- the declarative part, any elaboration statements, and
-- exception handler if any.  The selectors for blocks in STATEMENTS
-- can then be used for further decomposition.


---------------------------------------------------------------
-- RENAMINGS
-- LRM Chapter 8.5

function Is_Renaming_Declaration
            (A_Declaration : Declaration) return Boolean;

function Renamed_Name (A_Declaration : Declaration) return Name_Expression;
-- Returns the name of the entity being renamed.  It can be a simple
-- name, an operator symbol, an indexed component, a slice, a
-- selected component or an attribute.

function Renamed_Declaration
            (A_Declaration : Declaration) return Declaration;
-- If applied to the renaming of a simple name, an operator symbol
-- or an expanded name, returns the name's declaration.  Returns nil
-- element otherwise.  Use of this function is discouraged.

---------------------------------------------------------------
-- GENERIC PACKAGE and SUBPROGRAM SPECIFICATIONS
-- LRM Chapter 12

function Is_Generic
            (Package_Or_Subprogram_Decl : Declaration) return Boolean;

subtype Generic_Formal_Parameter                    is Ada_Program.Element;
subtype Generic_Formal_Parameter_Or_Pragma_Iterator is
   Ada_Program.Element_Iterator;

type Generic_Parameter_Kinds is (Subprogram,
                                 Object,
                                 Private_Type,
                                 Limited_Private_Type,
                                 Discrete_Type,
                                 Integer_Type,
                                 Floating_Point_Type,
                                 Fixed_Point_Type,
                                 Array_Type,
                                 Access_Type,
                                 Not_A_Generic_Parameter);

function Generic_Parameter_Kind
            (Generic_Parameter : Generic_Formal_Parameter)
            return Generic_Parameter_Kinds;
```

```
function Generic_Parameters
            (Generic_Decl : Declaration)
            return Generic_Formal_Parameter_Or_Pragma_Iterator;
-- Returns a list of formal parameters to the generic in order of
-- appearance.
-- Object parameters can be decomposed with subprogram formal parameter
-- and object declaration operations.
-- Array and access type declarations can be decomposed with the
-- operations corresponding to their types.
-- Subprogram parameters can be queried by the following operations and
-- can be further decomposed with subprogram declaration operations.

type Generic_Formal_Subprogram_Default_Kinds is
   (Box, Name, None, Not_A_Generic_Formal_Subprogram);

function Generic_Formal_Subprogram_Default_Kind
            (A_Generic_Formal_Subprogram : Generic_Formal_Parameter)
            return Generic_Formal_Subprogram_Default_Kinds;

function Generic_Formal_Subprogram_Default
            (A_Generic_Formal_Subprogram : Generic_Formal_Parameter)
            return Name_Expression;

function Is_Generic_Instantiation
            (Package_Or_Subprogram_Decl : Declaration) return Boolean;

function Generic_Unit_Declaration
            (Generic_Instantiation_Or_Unit_Declaration : Declaration)
            return Declaration;
-- Returns the declaration of the generic unit being instantiated.

function Generic_Instantiation_Parameters
            (Generic_Instantiation : Declaration)
            return Association_Iterator;
-- Returns an ordered list of parameter associations of a generic
-- instantiation.  The operations defined in package ASSOCIATIONS
-- can be used to decompose them.

function Generic_Actual_Parameters (Generic_Instantiation : Declaration)
                                   return Association_Iterator
    renames Generic_Instantiation_Parameters;
-- Use of this form is discouraged.


---------------------------------------------------------------
-- TASK DECLARATIONS
-- LRM Chapter 9
```





```
function Task_Type_Specification
            (Task_Decl : Task_Declaration) return Task_Specification;
-- Returns a task specification for a given task declaration.

function Entry_Declarations (Task_Decl : Task_Declaration)
                            return Declarative_Part_Iterator;
-- Returns a list of entry declarations associated with a TASK declaration.

function Task_Body_Block (Task_Body : Task_Declaration) return Statement;
-- Returns the block statement for the body including
-- the declarative part, any elaboration statements, and
-- exception handler if any.


---------------------------------------------------------------
-- ENTRY DECLARATIONS
-- LRM Chapter 9.5

-- The operations available for decomposing subprograms will
-- also work for entry declarations.

subtype Family_Index_Range is Ada_Program.Element;

function Family_Index (Entry_Family : Entry_Declaration)
                      return Family_Index_Range;
-- Returns the index range for the entry family.  If the entry is not
-- a family, a nil element is returned.
-- Use operations on discrete ranges in TYPE_INFORMATION to analyze
-- the family index range.


---------------------------------------------------------------
-- EXCEPTION DECLARATIONS
-- LRM Chapter 11

-- The selector IDENTIFIERS can be used to get a list of identifier
-- definitions introduced by an exception declaration.


---------------------------------------------------------------
-- SUBUNIT STUBS
-- LRM Chapter 10.2

function Subunit (Of_Body_Stub : Declaration) return Compilation_Unit;
-- Returns the compilation unit corresponding to the subunit stub.

end Declarations;
```

```
with Ada_Program;

package Names_And_Expressions is

    --
    --   The use of this system is subject to the software license terms and
    --   conditions agreed upon between Rational and the Customer.
    --
    --              Copyright 1987, 1988, 1989, 1990 by Rational.
    --
    --                          RESTRICTED RIGHTS LEGEND
    --
    --   Use, duplication, or disclosure by the Government is subject to
    --   restrictions as set forth in subdivision (b)(3)(ii) of the Rights in
    --   Technical Data and Computer Software clause at 52.227-7013.
    --
    --
    --               Rational
    --               3320 Scott Boulevard
    --               Santa Clara, California 95054
    --
    --  PROPRIETARY AND CONFIDENTIAL INFORMATION OF RATIONAL;
    --  USE OR COPYING WITHOUT EXPRESS WRITTEN AUTHORIZATION
    --  IS STRICTLY PROHIBITED.  THIS MATERIAL IS PROTECTED AS
    --  AN UNPUBLISHED WORK UNDER THE U.S. COPYRIGHT ACT OF
    --  1976.  CREATED 1987, 1988, 1989, 1990.  ALL RIGHTS RESERVED.
    --


    -- LRM Chapter 4

    -- This chapter contains operations for manipulating names and
    -- expressions.

    -- Local Renamings:
    subtype Declaration         is Ada_Program.Declaration;
    subtype Expression          is Ada_Program.Expression;
    subtype Name_Expression     is Ada_Program.Name;
    subtype Name                is Ada_Program.Name;
    subtype Subtype_Indication  is Ada_Program.Element;
    subtype Type_Definition     is Ada_Program.Type_Definition;

    subtype Association_Iterator is Ada_Program.Association_Iterator;
    subtype Choice_Iterator      is Ada_Program.Choice_Iterator;
    subtype Expression_Iterator  is Ada_Program.Expression_Iterator;
    --------------------------------------------------------------------
```





```
    function Expression_Type
                (An_Expression : Expression) return Type_Definition;
    -- Returns the type specification for the expression.

    type Expression_Kinds is
       (A_Simple_Name, An_Operator_Symbol,                          -- LRM 4.1
        An_Indexed_Component,                                       -- LRM 4.1.1
        A_Slice,                                                    -- LRM 4.1.2
        A_Selected_Component,                                       -- LRM 4.1.3
        An_Attribute,                                               -- LRM 4.1.4
        A_Character_Literal, An_Integer_Literal, A_Real_Literal,
        An_Enumeration_Literal, A_Null_Literal, A_String_Literal,-- LRM 4.2
        An_Aggregate,                                               -- LRM 4.3
        A_Type_Conversion,                                          -- LRM 4.6
        A_Qualified_Expression,                                     -- LRM 4.7
        An_Allocator,                                               -- LRM 4.8
        A_Complex_Expression,                                       -- LRM 4.4/5
        A_Function_Call,
        Not_An_Expression);

    function Kind (An_Expression : Expression) return Expression_Kinds;

    function Is_Constant (A_Name : Name) return Boolean;
    -- Returns True if the given name is constant.  The name must be
    -- of a syntactic form suitable for the left hand side of an
    -- assignment (ie. not an attribute, a character, etc.).

    function Is_Static (An_Expression : Expression) return Boolean;

    function Static_Value
                (An_Integer_Expression : Expression) return Long_Integer;
    function Static_Value
                (A_Character_Expression : Expression) return Character;
    function Static_Value (A_Real_Expression : Expression) return Float;
    function Static_Value (A_String_Expression : Expression) return String;
    -- Note that STATIC_VALUE for strings will not return the quotes around
    -- a string literal.

    function Used_Names (An_Expression : Expression)
                        return Ada_Program.Element_List;
    -- Returns a list of names of objects/types and operators in an expression.
    -- EG. the expression (A + B.D (Q'(4))) would return the list :
    --        A   : A_SIMPLE_NAME
    --        +   : AN_OPERATOR_SYMBOL
    --        B.D : A_SELECTED_COMPONENT
    --        Q   : A_SIMPLE_NAME
    --        4   : A_NUMERIC_LITERAL
```

```
---------------------------------------------------------------
-- NAMES LRM 4.1
--

-- Simple_Names and operator symbols are instances of identifier references
-- The Ada_program.Definition function will return the defining Id.

subtype Discrete_Range is Ada_Program.Element;

function Prefix (Of_Name : Name) return Name;
-- Returns the prefix (the construct to the left of the rightmost
-- left paren in indexed or sliced objects, the rightmost 'dot' for
-- selected components or the rightmost tick for attributes).


-- LRM 4.1.1 -- Array component
function Index_Expressions (An_Indexed_Component : Name)
                           return Expression_Iterator;
-- Returns a list of expressions (possibly only one) within the parens.


-- LRM 4.1.2
function Slice_Range (A_Slice : Name) return Discrete_Range;


-- LRM 4.1.3
type Selection_Kinds is (Record_Discriminant,  -- LRM 4.1.3 (a)
                         Record_Component,     -- LRM 4.1.3 (b)
                         Task_Entry,           -- LRM 4.1.3 (c)
                         Access_Object,        -- LRM 4.1.3 (d)
                         Expanded_Name         -- LRM 4.1.3 (e,f)
                         );

function Selection_Kind (Selected_Component : Name) return Selection_Kinds;

function Selector (Selected_Component : Name) return Name;
-- Returns the selector (the construct to the right of the rightmost
-- 'dot' in the selected component).  Fails on selections of kind
-- Access_Object.


-- LRM 4.1.3 (a,b)
function Record_Object
         (Discriminant_Or_Component_Selection : Name) return Declaration;
-- Returns the record object declaration for the selected object.

function Selected_Component (Discriminant_Or_Component_Selection : Name)
                            return Ada_Program.Element;
-- Returns the component declaration or discriminant in the record type
```





```
-- declaration. Operations in the package Declarations can be used to
-- manipulate record components.


-- LRM 4.1.3 (c)
function Selected_Task_Entry
         (Task_Entry_Selection : Name) return Declaration;
-- Returns the entry declaration within the task type.


-- LRM 4.1.3 (d)
function Selected_Access_Type
         (Access_Object_Selection : Name) return Declaration;
-- Returns the access type declaration.


-- LRM 4.1.3 (f)
function Named_Declaration (Expanded_Name : Name) return Declaration;
-- Returns the named declaration.


-- LRM 4.1.4
type Attribute_Designator_Kinds is (Address_Attribute,
                                    Aft_Attribute,
                                    Base_Attribute,
                                    Callable_Attribute,
                                    Constrained_Attribute,
                                    Count_Attribute,
                                    Delta_Attribute,
                                    Digits_Attribute,
                                    Emax_Attribute,
                                    Epsilon_Attribute,
                                    First_Attribute,
                                    First_Bit_Attribute,
                                    Fore_Attribute,
                                    Image_Attribute,
                                    Large_Attribute,
                                    Last_Attribute,
                                    Last_Bit_Attribute,
                                    Length_Attribute,
                                    Machine_Emax_Attribute,
                                    Machine_Emin_Attribute,
                                    Machine_Mantissa_Attribute,
                                    Machine_Overflows_Attribute,
                                    Machine_Radix_Attribute,
                                    Machine_Rounds_Attribute,
                                    Mantissa_Attribute,
                                    Pos_Attribute,
                                    Position_Attribute,
```

```
                              Pred_Attribute,
                              Range_Attribute,
                              Safe_Emax_Attribute,
                              Safe_Large_Attribute,
                              Safe_Small_Attribute,
                              Size_Attribute,
                              Small_Attribute,
                              Storage_Size_Attribute,
                              Succ_Attribute,
                              Terminated_Attribute,
                              Val_Attribute,
                              Value_Attribute,
                              Width_Attribute,
                              Not_A_Predefined_Attribute);

function Attribute_Designator_Kind
            (Attribute : Name) return Attribute_Designator_Kinds;
-- Returns the kind of an attribute.  If the attribute is
-- implementation-specific, Not_A_Predefined_Attribute is returned.

function Attribute_Designator_Name (Attribute : Name) return String;
-- This is the preferred way to analyze an implementation-specific
-- attribute.  It returns an uppercase string for the attribute
-- simple name.

function Attribute_Designator_Name (Attribute : Name) return Name;
-- The Simple_Name returned here is only intended for use by
-- ADA_PROGRAM.STRING_NAME.  Use of this function is discouraged.

function Attribute_Designator_Argument (Attribute : Name) return Expression;
-- Returns the static expression associated with the optional argument
-- of the attribute designator if one exists, Nil_Element otherwise.


----------------------------------------------------------------------
-- LITERALS LRM 4.2
--
-- The value of literals can be determined by using the
-- STATIC_VALUE selectors.

function Is_Literal (An_Expression : Expression) return Boolean;

function Position_Number (An_Enumeration_Or_Character_Literal : Expression)
                          return Long_Integer;
-- Returns the cardinality of the enumeration literal within the base type
-- of the enumeration type. (same as "'POS")
```





```
function Representation_Value
            (An_Enumeration_Or_Character_Literal : Expression)
            return Long_Integer;
-- Returns the internal representation of the enumeration literal.
-- (same as "'POS" if no rep spec defined for the enumeration type)

function Enumeration_Definition
            (An_Enumeration_Or_Character_Literal : Expression)
            return Declaration;
-- Since characters are enumerations, both regular enumeration and
-- character literals have enumeration root type declarations.  The
-- operations in DECLARATIONS can be used to further analyze these
-- declarations.


----------------------------------------------------------------------
-- AGGREGATES LRM 4.3

subtype Aggregate_Component          is Ada_Program.Element;
subtype Aggregate_Component_Iterator is Ada_Program.Element_Iterator;

function Components (An_Aggregate : Expression;
                     Normalized   : Boolean := False)
                    return Aggregate_Component_Iterator;
-- Returns a list of the components of an aggregate.
-- If NORMALIZED is true a normalized list of the components of
-- an aggregate is returned (using positional notation).
-- NOTE THAT NORMALIZED INFO IS NOT AVAILIABLE FOR ARRAY AGGREGATES.

function Component_Choices
            (Component : Aggregate_Component) return Choice_Iterator;
-- Returns the list of choices in the aggregate component.
-- May be a nil list if positional notation is used.
-- Use the CHOICE operations in TYPE_INFORMATION for further analysis.

function Component_Expression
            (Component : Aggregate_Component) return Expression;
-- Returns the expression for the component association.

function Aggregate_Range (An_Aggregate : Expression) return Discrete_Range;
-- For an array aggregate, returns a range specifying the bounds of
-- the aggregate.


----------------------------------------------------------------------
-- TYPE CONVERSIONS and QUALIFIED EXPRESSIONS
-- LRM 4.6, 4.7
```

```
function Type_Mark (Type_Conversion_Or_Qualified_Expression : Expression)
                   return Name;
-- Returns the name of the type to which the expression is
-- being converted or the qualifying type.  Use DEFINITION to get
-- the the defining type id.

function Converted_Or_Qualified_Expression
            (Type_Conversion_Or_Qualified_Expression : Expression)
            return Expression;
-- Returns the expression being converted or qualified.


---------------------------------------------------------------------
-- ALLOCATORS LRM 4.8

type Allocation_Kinds is (Allocation_From_Subtype,
                          Allocation_From_Qualified_Expression);

function Allocator_Kind (An_Allocator : Expression) return Allocation_Kinds;

function Allocation_Type (An_Allocator : Expression)
                         return Subtype_Indication;
-- Returns the subtype indication for the object being allocated.

function Qualified_Object_Expression
            (An_Allocator : Expression) return Expression;
-- Returns the qualified expression for the object being allocated.
-- (in other words the KIND of the returned expression will be
-- A_QUALIFIED_EXPRESSION)



---------------------------------------------------------------------
-- COMPLEX EXPRESSIONS - LRM 4.4
--
-- When an expression kind is A_COMPLEX_EXPRESSION the following
-- operations can be used to do more detailed analysis.

subtype Special_Operation        is Expression;
subtype Parenthesized_Expression is Expression;
subtype Range_Info               is Ada_Program.Element;

type Complex_Expression_Kinds is (A_Parenthesized_Expression,
                                  A_Special_Operation,
                                  Not_A_Complex_Expression);

function Complex_Expression_Kind
            (An_Expression : Expression) return Complex_Expression_Kinds;
```





```
function Expression_Parenthesized
            (A_Parenthesized_Expression : Parenthesized_Expression)
            return Expression;
-- Returns the expression within the parenthesis.

type Special_Operation_Kinds is (In_Range, Not_In_Range,
                                 In_Type, Not_In_Type,
                                 And_Then, Or_Else,
                                 Not_A_Special_Operation);

function Special_Operation_Kind (An_Operation : Special_Operation)
                                return Special_Operation_Kinds;

function Special_Operation_Left_Hand_Side
            (For_Special_Operation : Special_Operation) return Expression;
-- All special operation left hand sides are expressions.

function In_Range_Operation_Right_Hand_Side
            (For_In_Range_Operation : Special_Operation) return Range_Info;
-- The right hand side for an IN_RANGE operation is a range which can be
-- analyzed using the range operations in TYPE_INFORMATION.

function In_Type_Operation_Right_Hand_Side
            (For_In_Type_Operation : Special_Operation) return Name;
-- The right hand side for an IN_TYPE operation is a type mark which is
-- a name and can be further analyzed using this package.

function Short_Circuit_Operation_Right_Hand_Side
            (For_Short_Circuit_Operation : Special_Operation)
            return Expression;
-- The right hand side for a short circuit operation can be any
-- expression kind which can be further analyzed using this package.


---------------------------------------------------------------------
-- FUNCTION CALLS

-- Note that references to enumeration literals renamed as functions
-- are treated as genuine function calls.

function Is_Prefix_Call (A_Function_Call : Expression) return Boolean;
-- Returns true if the function call is in prefix form.
-- EG. - Foo (A, B);    -- Returns TRUE
--       "<" (A, B);    -- Returns TRUE
--       ... A < B ... -- Returns FALSE
```

```
    function Is_Predefined (A_Function_Call : Expression) return Boolean;
    -- Returns true if the function call has no real declaration associated
    -- with it. (EG. STANDARD."+")

    function Called_Function (A_Function_Call : Expression) return Declaration;
    -- Returns the declaration of the called function if it is not predefined,
    --   NIL_ELEMENT otherwise.

    function Function_Call_Parameters
                (A_Function_Call : Expression;
                 Normalized : Boolean := False) return Association_Iterator;
    -- Returns a list of actual parameters for the call.  If Normalized
    -- is set to true, the (unspecified) default parameters will also be
    -- included in the iterator.  Use the operations from package
    -- ASSOCIATIONS to further decompose the associations.

end Names_And_Expressions;
```





```
with Ada_Program;

package Pragmas is

    --
    --    The use of this system is subject to the software license terms and
    --    conditions agreed upon between Rational and the Customer.
    --
    --                Copyright 1987, 1988, 1989, 1990 by Rational.
    --
    --                          RESTRICTED RIGHTS LEGEND
    --
    --    Use, duplication, or disclosure by the Government is subject to
    --    restrictions as set forth in subdivision (b)(3)(ii) of the Rights in
    --    Technical Data and Computer Software clause at 52.227-7013.
    --
    --
    --                Rational
    --                3320 Scott Boulevard
    --                Santa Clara, California 95054
    --
    --   PROPRIETARY AND CONFIDENTIAL INFORMATION OF RATIONAL;
    --   USE OR COPYING WITHOUT EXPRESS WRITTEN AUTHORIZATION
    --   IS STRICTLY PROHIBITED.  THIS MATERIAL IS PROTECTED AS
    --   AN UNPUBLISHED WORK UNDER THE U.S. COPYRIGHT ACT OF
    --   1976.  CREATED 1987, 1988, 1989, 1990.  ALL RIGHTS RESERVED.
    --

    -- LRM 2.8
    -- This package provides operations on pragma elements

    -- Local Renamings:
    subtype Pragma_Usage is Ada_Program.Pragma_Usage;
    subtype Declaration  is Ada_Program.Declaration;

    subtype Association_Iterator is Ada_Program.Association_Iterator;
    ---------------------------------------------------------------------

    function Is_Predefined (A_Pragma : Pragma_Usage) return Boolean;
```

```
    type Pragma_Kinds is (Controlled,
                          Elaborate,
                          Inline,
                          Interface,
                          List,
                          Memory_Size,
                          Optimize,
                          Pack,
                          Page,
                          Priority,
                          Shared,
                          Storage_Unit,
                          Suppress,
                          System_Name,
                          Not_A_Predefined_Pragma);

    Unknown : constant Pragma_Kinds := Not_A_Predefined_Pragma;

    function Kind (A_Pragma : Pragma_Usage) return Pragma_Kinds;
    -- Returns the kind of a pragma.  Returns Not_A_Predefined_Pragma on
    -- implementation-specific pragmas.

    function Name (A_Pragma : Pragma_Usage) return String;
    -- Returns the uppercase simple name of any pragma.  This is the way
    -- to analyze implementation-specific pragmas.

    function Arguments (A_Pragma : Pragma_Usage) return Association_Iterator;
    -- Returns a list of the arguments to a pragma.  Operations from
    -- package ASSOCIATIONS can be used to decompose them.

end Pragmas;
```





```
with Ada_Program;

package Representation_Clauses is

    --
    --    The use of this system is subject to the software license terms and
    --    conditions agreed upon between Rational and the Customer.
    --
    --                 Copyright 1987, 1988, 1989, 1990 by Rational.
    --
    --                            RESTRICTED RIGHTS LEGEND
    --
    --    Use, duplication, or disclosure by the Government is subject to
    --    restrictions as set forth in subdivision (b)(3)(ii) of the Rights in
    --    Technical Data and Computer Software clause at 52.227-7013.
    --
    --
    --                 Rational
    --                 3320 Scott Boulevard
    --                 Santa Clara, California 95054
    --
    --   PROPRIETARY AND CONFIDENTIAL INFORMATION OF RATIONAL;
    --   USE OR COPYING WITHOUT EXPRESS WRITTEN AUTHORIZATION
    --   IS STRICTLY PROHIBITED.  THIS MATERIAL IS PROTECTED AS
    --   AN UNPUBLISHED WORK UNDER THE U.S. COPYRIGHT ACT OF
    --   1976.  CREATED 1987, 1988, 1989, 1990.  ALL RIGHTS RESERVED.
    --


    -- LRM Chapter 13

    -- Local Renamings:
    subtype Declaration            is Ada_Program.Declaration;
    subtype Expression             is Ada_Program.Expression;
    subtype Identifier_Definition  is Ada_Program.Identifier_Definition;
    subtype Representation_Clause  is Ada_Program.Representation_Clause;
    subtype Type_Definition        is Ada_Program.Element;
    ------------------------------------------------------------------

    type Representation_Clause_Kinds is (A_Length_Clause,
                                         An_Enumeration_Representation_Clause,
                                         A_Record_Representation_Clause,
                                         An_Address_Clause,
                                         Not_A_Representation_Clause);

    function Kind (Clause : Representation_Clause)
                  return Representation_Clause_Kinds;
```

```
function Associated_Type
            (Clause : Representation_Clause) return Type_Definition;
-- Returns the definition of the type specified in the length clause,
-- enumeration representation clause or record representation clause.

function Associated_Size (For_Type : Type_Definition) return Expression;
-- Returns the expression that describes the size associated with a type
--  definition if there is a rep spec associated with this type,
--  NIL_ELEMENT otherwise.

function Associated_Storage_Size
            (For_Type : Type_Definition) return Expression;
-- Returns the expression that describes the size associated with a type
--  definition. Valid for access and task types and their derived types.

function Associated_Enumeration_Representation
            (For_Enumeration_Literal : Identifier_Definition)
            return Expression;
-- Returns the expression that describes the representation of
--  enumeration literal.

function Associated_Enumeration_Representation
            (For_Enumeration_Literal : Identifier_Definition)
            return Long_Integer;
-- Returns the representation value associated with
--  FOR_ENUMERATION_LITERAL.

function Associated_Record_Representation
            (Record_Or_Component : Ada_Program.Element)
            return Representation_Clause;
-- Returns the representation spec associated with the element.

function Associated_Address
            (For_Element : Ada_Program.Element) return Ada_Program.Element;
-- Returns the element that defines the address value of FOR_ELEMENT,
--  valid for VARs, CONSTANTs, PACKAGEs, ENTRYs and TASK SPECs (or
--  their associated IDs).


--  For all the above routines, if no associated element exists
--  a NIL_ELEMENT is returned.

--------------------------------------------------------------------
-- LRM 13.2

type Length_Clause_Attribute_Kinds is (Size,
                                       Collection_Storage_Size,
                                       Task_Storage_Size,
                                       Small);
```





```
function Attribute_Kind (A_Length_Clause : Representation_Clause)
                         return Length_Clause_Attribute_Kinds;


--------------------------------------------------------------------
-- LRM 13.3

function Representation_Aggregate
            (Enumeration_Clause : Representation_Clause) return Expression;
-- Returns the aggregate of the representation clause.



--------------------------------------------------------------------
-- LRM 13.4

function Alignment_Expression
            (Record_Clause : Representation_Clause) return Expression;
-- Returns the alignment expression for the representation clause.
-- If an alignment expression is not present, a nil element is
-- returned.

subtype A_Range is Ada_Program.Element;
subtype Record_Component_Clause is Ada_Program.Element;
subtype Record_Component_Clause_Or_Pragma_Iterator is
   Ada_Program.Element_Iterator;

function Clause_Components
            (Record_Clause : Representation_Clause)
            return Record_Component_Clause_Or_Pragma_Iterator;
-- Returns a list of the components of the record clause in order
-- of appearance.

function Valid_Component
            (Component_Clause : Record_Component_Clause) return Boolean;
-- Some components can be PRAGMAs, this return true if this component
-- is NOT a pragma.

function Component_Name
            (Component_Clause : Record_Component_Clause) return String;
-- Returns the name of the component.

function Component_Offset
            (Component_Clause : Record_Component_Clause) return Expression;
-- Returns the offset expression for the component.

function Component_Range
            (Component_Clause : Record_Component_Clause) return A_Range;
-- Returns the range constraint for the component.
```

```
    -----------------------------------------------------------------
    -- LRM 13.5

    function Addressed_Declaration
                (Address_Clause : Representation_Clause) return Declaration;
    -- Returns the declaration of the object, subprogram or entry
    -- whose address is being specified.


    function Address_Expression
                (Address_Clause : Representation_Clause) return Expression;
    -- Returns the expression for the address of the declaration.

end Representation_Clauses;
```





```
with Ada_Program;
with Declarations;

package Statements is

    --
    --    The use of this system is subject to the software license terms and
    --    conditions agreed upon between Rational and the Customer.
    --
    --               Copyright 1987, 1988, 1989, 1990 by Rational.
    --
    --                          RESTRICTED RIGHTS LEGEND
    --
    --    Use, duplication, or disclosure by the Government is subject to
    --    restrictions as set forth in subdivision (b)(3)(ii) of the Rights in
    --    Technical Data and Computer Software clause at 52.227-7013.
    --
    --
    --               Rational
    --               3320 Scott Boulevard
    --               Santa Clara, California 95054
    --
    --   PROPRIETARY AND CONFIDENTIAL INFORMATION OF RATIONAL;
    --   USE OR COPYING WITHOUT EXPRESS WRITTEN AUTHORIZATION
    --   IS STRICTLY PROHIBITED.  THIS MATERIAL IS PROTECTED AS
    --   AN UNPUBLISHED WORK UNDER THE U.S. COPYRIGHT ACT OF
    --   1976.  CREATED 1987, 1988, 1989, 1990.  ALL RIGHTS RESERVED.
    --

    -- LRM Chapter 5

    -- Local Renames:
    subtype Association          is Ada_Program.Association;
    subtype Declaration          is Ada_Program.Declaration;
    subtype Expression           is Ada_Program.Expression;
    subtype Identifier_Reference is Ada_Program.Identifier_Reference;
    subtype Name_Expression      is Ada_Program.Name;
    subtype Name                 is Ada_Program.Name;
    subtype Statement            is Ada_Program.Statement;

    subtype Association_Iterator      is Ada_Program.Association_Iterator;
    subtype Choice_Iterator           is Ada_Program.Choice_Iterator;
    subtype Declarative_Part_Iterator is
       Ada_Program.
Declaration_Or_Context_Clause_Or_Representation_Clause_Or_Pragma_Iterator;
    subtype Expression_Iterator       is Ada_Program.Expression_Iterator;
```

```
subtype Name_Iterator             is Ada_Program.Name_Iterator;
subtype Statement_Part_Iterator   is
   Ada_Program.Statement_Or_Pragma_Iterator;
-------------------------------------------------------------------

function Is_Labeled (A_Statement : Statement) return Boolean;

function Labels (A_Statement : Statement) return Name_Iterator;
-- Returns an iterator on the names of the labels of a statement.  A
-- statement can have several labels.

function Label_Name (A_Statement : Statement) return String;
-- Returns the null string if no label is present.  Use of this
-- function is discouraged.

function Is_Named_Statement (A_Statement : Statement) return Boolean;
-- Returns true if applied to a loop or block that has a name.

function Statement_Name (A_Statement : Statement) return Name;
-- Returns the name of a block or loop.   Returns Nil_Element if not
-- a block or loop, or if no name is present.

type Statement_Kinds is

   -- Simple statements:
   (A_Null_Statement, An_Assignment_Statement, A_Procedure_Call_Statement,
    An_Exit_Statement, A_Return_Statement, A_Goto_Statement,
    An_Entry_Call_Statement, A_Delay_Statement,
    An_Abort_Statement, A_Raise_Statement, A_Code_Statement,

    -- compound statements:
    An_If_Statement, A_Case_Statement, A_Loop_Statement,
    A_Block_Statement, An_Accept_Statement,
    A_Select_Statement, A_Conditional_Entry_Call_Statement,
    A_Timed_Entry_Call_Statement, Not_A_Statement);

function Kind (A_Statement : Statement) return Statement_Kinds;

---------------------------------------------------------------------
-- ASSIGNMENT STATEMENTS - LRM 5.2

function Object_Assigned_To
            (Assignment_Statement : Statement) return Name;
-- Returns the name of object to which the assignment is being made.

function Assignment_Expression
            (Assignment_Statement : Statement) return Expression;
-- Returns the expression on the right hand side of the assignment.
```





```
-------------------------------------------------------------------
-- EXIT STATEMENTS - LRM 5.7

function Exit_Label (Exit_Statement : Statement) return String;
-- Returns the name of the exited loop if present, "" if not present

function Exit_Condition (Exit_Statement : Statement) return Expression;
-- Returns the when condition of the exit statement if present;
-- returns a nil element if not present.

function Loop_Exited (Exit_Statement : Statement) return Statement;
-- Returns the loop statement exited by this exit statement.


-------------------------------------------------------------------
-- RETURN STATEMENTS - LRM 5.8

function Return_Expression (Return_Statement : Statement) return Expression;
-- Returns the expression returned in the statement.
-- If no expression exists, a nil element is returned.


-------------------------------------------------------------------
-- GOTO STATEMENTS - LRM 5.9

function Goto_Label (Goto_Statement : Statement) return String;
-- Returns the name of label to which the goto statement may go.

function Destination_Statement
            (Goto_Statement : Statement) return Statement;
-- Returns the statement to which the goto statement may go.


-------------------------------------------------------------------
-- IF STATEMENTS - LRM 5.3

subtype If_Statement_Arm is Ada_Program.Element;
subtype If_Statement_Arm_Iterator is Ada_Program.Element_Iterator;

function If_Arm_List (If_Statement : Statement)
                     return If_Statement_Arm_Iterator;
-- returns a list of the arms of the if statement

function Is_Else_Arm (Arm : If_Statement_Arm) return Boolean;
-- Returns true if the arm is an 'ELSE' arm, false if the arm is an
-- 'IF' or 'ELSIF'. The function Condition_Expression below may be used
-- on the arm in the false case.

function Condition_Expression (If_Arm : If_Statement_Arm) return Expression;
```

```
-- Returns the condition expression for an if statement or elsif arm.

function If_Arm_Statements (Arm : If_Statement_Arm)
                            return Statement_Part_Iterator;
-- Returns a list of the statements and pragmas in an arm.


-----------------------------------------------------------------------
-- CASE STATEMENTS - LRM 5.4

function Case_Expression (Case_Statement : Statement) return Expression;
-- Returns the expression of the case statement that determines
-- which alternative will be taken.

subtype Case_Statement_Alternative is Ada_Program.Element;
subtype Case_Statement_Alternative_Iterator is Ada_Program.Element_Iterator;

function Case_Arms_List (Case_Statement : Statement)
                         return Case_Statement_Alternative_Iterator;
-- Return a list of all alternatives of the case statement.

function Is_When_Others
            (Case_Alternative : Case_Statement_Alternative) return Boolean;

function Case_Alternative_Choices
            (Case_Alternative : Case_Statement_Alternative)
            return Choice_Iterator;
-- Returns a list of the 'WHEN <choice> | <choice>' choices.
-- Use the TYPE_INFORMATION package's CHOICES queries to extract
--  further information about the <choice>s.

function Case_Alternative_Statements
            (Case_Alternative : Case_Statement_Alternative)
            return Statement_Part_Iterator;
-- Returns a list of the statements and pragmas in this alternative.


-----------------------------------------------------------------------
-- LOOP STATEMENTS - LRM 5.5

subtype For_Loop_Range is Ada_Program.Element;

type Loop_Kinds is (A_For_Loop, A_While_Loop, A_Simple_Loop);

function Loop_Kind (Loop_Statement : Statement) return Loop_Kinds;

function While_Condition (Loop_Statement : Statement) return Expression;
-- Returns the condition expression associated with the while loop.
```





```
function For_Loop_Index (Loop_Statement : Statement) return For_Loop_Range;
-- Returns the range for the loop index.
-- Use the TYPE_INFORMATION package's operations for discrete ranges
-- for more information.

function For_Loop_Index_Variable (Loop_Statement : Statement) return String;
-- Returns the name of the loop index variable.

function Is_Reverse_Iterator (Loop_Statement : Statement) return Boolean;

function Loop_Statements (Loop_Statement : Statement)
                          return Statement_Part_Iterator;
-- Returns a list of the statements pragmas in the body part of
-- the loop statement.


-----------------------------------------------------------------------
-- BLOCK STATEMENTS - LRM 5.6

function Declarative_Items (Block_Statement : Statement)
                            return Declarative_Part_Iterator;
-- Returns a list of the declarations, pragmas, representation
-- specifications, and use clauses in the declarative part of the block.
-- A "Done" iterator indicates that there are no declarations.

function Block_Body_Statements (Block_Statement : Statement)
                                return Statement_Part_Iterator;
-- Returns a list of the statements and pragmas in the body part of the
-- block.

subtype Exception_Handler_Arm is Ada_Program.Element;
subtype Exception_Handler_Arm_Iterator is Ada_Program.Element_Iterator;

function Block_Exception_Handler_Arms (Block_Statement : Statement)
                                       return Exception_Handler_Arm_Iterator;
-- Returns a list of the exception handler arms of the block.

function Exception_Choices (Exception_Arm : Exception_Handler_Arm)
          (Exception_Arm : Exception_Handler_Arm) return Choice_Iterator;
-- Returns a list of exception choices in the handler arm.
-- Use the TYPE_INFORMATION package's CHOICES queries to extract
--  further information about the <choice>s.

function Handler_Statements (Exception_Arm : Exception_Handler_Arm)
                             return Statement_Part_Iterator;
-- Returns a list of the statements and pragmas in the body part of
-- the handler.
```

```
---------------------------------------------------------------------
-- PROCEDURE CALL STATEMENTS - LRM 6.4

function Called_Procedure (Procedure_Or_Entry_Call_Statement : Statement)
                          return Declaration;
-- Returns the declaration of the called procedure or entry.

function Procedure_Call_Parameters
            (Procedure_Or_Entry_Call_Statement : Statement;
             Normalized                        : Boolean := False)
            return Association_Iterator;
-- Returns a list of actual parameters for the call.
-- If Normalized is set to true, the (unspecified) default
-- parameters will also be included in the iterator.

function Parameter_Expression (Parameter : Association) return Expression;
-- Returns the expression passed in for the actual parameter.  Use
-- of this function is discouraged.  Operations from package
-- PARAMETER_ASSOCIATIONS should be used instead.


---------------------------------------------------------------------
-- RAISE STATEMENTS - LRM 11.3

function Raised_Exception (Raise_Statement : Statement) return Name;
-- Returns the name of the raised exception or NIL_ELEMENT if there is
-- none. The NAMES_AND_EXPRESSIONS package can be used to decompose
-- the name.


---------------------------------------------------------------------
-- CODE STATEMENTS - LRM 13.8

function Qualified_Expression
            (Code_Statement : Statement) return Expression;
-- Returns the qualified expression representing the code statement.


---------------------------------------------------------------------
-- ENTRY CALL STATEMENTS - LRM 9.5

function Family_Index
            (Entry_Call_Or_Accept_Statement : Statement) return Expression;
-- Returns NIL_ELEMENT if not a family call/accept.

-- The operations on procedure calls and actual parameters defined above
-- may be used to get more information about an entry calls.
```





```
---------------------------------------------------------------------
-- ACCEPT STATEMENTS - LRM 9.5

function Accepted_Entry (Accept_Statement : Statement)
                        return Declarations.Entry_Declaration;
-- Returns the declaration of the entry accepted in this statement.

function Accept_Body_Statements (Accept_Statement : Statement)
                                return Statement_Part_Iterator
-- Returns a list of the statements and pragmas in the body part of
-- the accept statement.


---------------------------------------------------------------------
-- DELAY STATEMENTS - LRM 9.6

function Delay_Expression (Delay_Statement : Statement) return Expression;
-- Returns the expression for the time of the delay


---------------------------------------------------------------------
-- SELECT, CONDITIONAL ENTRY and TIMED ENTRY STATEMENTS - LRM 9.7

subtype Select_Alternative is Ada_Program.Element;
subtype Select_Alternative_Iterator is Ada_Program.Element_Iterator;

function Select_Alternatives (Selective_Wait : Statement)
                             return Select_Alternative_Iterator;
-- Returns a list of the alternatives in the selective_wait statement.

function Is_Guarded (Alternative : Select_Alternative) return Boolean;
-- Returns true if a select alternative has a guard.

function Guard (Alternative : Select_Alternative) return Expression;
-- Returns the conditional expression guarding the alternative.
-- May return a nil element if there is no guard.

type Select_Alternative_Kinds is (Accept_Alternative,
                                  Delay_Alternative,
                                  Terminate_Alternative,
                                  Not_A_Select_Alternative);

function Select_Alternative_Kind (Alternative : Select_Alternative)
                                 return Select_Alternative_Kinds;

function Select_Alternative_Statements
            (Accept_Or_Delay_Alternative : Select_Alternative)
            return Statement_Part_Iterator;
-- Returns a list of the statements and pragmas in the the accept or
```

```
    -- delay alternative including the accept statement and delay statements
    -- themselves.

    function Else_Statements
                (Selective_Wait_Or_Conditional_Entry_Call : Statement)
                return Statement_Part_Iterator;
    -- Returns a list of statements and pragmas contained in the else
    -- part of a selective_wait or conditional_entry_call.  If no else
    -- part exists, a "DONE" iterator is returned.

    function Timed_Statements (Timed_Entry_Call : Statement)
                              return Statement_Part_Iterator;
    -- Returns a list of statements and pragmas contained in the or
    -- part of a timed entry call, including the delay statement itself.

    function Entry_Call_Statements (Conditional_Or_Timed_Entry_Call : Statement)
                                   return Statement_Part_Iterator;
    -- Returns the statement list associated with the conditional or
    -- timed entry call, including the actual entry call statement.  Use
    -- the ELSE_STATEMENTS or TIMED_STATEMENTS selector functions the get
    -- the rest of the information about the call.


    -------------------------------------------------------------------
    -- ABORT STATEMENTS - LRM 9.10

    function Aborted_Tasks (Abort_Statement : Statement) return Name_Iterator;
    -- Returns a list of NAME_EXPRESSIONS for the aborted tasks.
    -- Use ADA_PROGRAM.DEFINITION to get to the task declaration, or
    -- the NAMES_AND_EXPRESSIONS package to decompose the names.

end Statements;
```





```
with Ada_Program;

package Type_Information is

    --
    --    The use of this system is subject to the software license terms and
    --    conditions agreed upon between Rational and the Customer.
    --
    --              Copyright 1987, 1988, 1989, 1990 by Rational.
    --
    --                        RESTRICTED RIGHTS LEGEND
    --
    --    Use, duplication, or disclosure by the Government is subject to
    --    restrictions as set forth in subdivision (b)(3)(ii) of the Rights in
    --    Technical Data and Computer Software clause at 52.227-7013.
    --
    --
    --              Rational
    --              3320 Scott Boulevard
    --              Santa Clara, California 95054
    --
    --   PROPRIETARY AND CONFIDENTIAL INFORMATION OF RATIONAL;
    --   USE OR COPYING WITHOUT EXPRESS WRITTEN AUTHORIZATION
    --   IS STRICTLY PROHIBITED.  THIS MATERIAL IS PROTECTED AS
    --   AN UNPUBLISHED WORK UNDER THE U.S. COPYRIGHT ACT OF
    --   1976.  CREATED 1987, 1988, 1989, 1990.  ALL RIGHTS RESERVED.
    --

    -- Local Renaming:

    subtype Declaration          is Ada_Program.Declaration;
    subtype Name_Expression      is Ada_Program.Expression;
    subtype Expression           is Ada_Program.Expression;
    subtype Identifier_Reference is Ada_Program.Identifier_Reference;
    subtype Statement            is Ada_Program.Statement;
    subtype Task_Specification   is Ada_Program.Type_Definition;
    subtype Type_Definition      is Ada_Program.Type_Definition;

    subtype Association_Iterator      is Ada_Program.Association_Iterator;
    subtype Choice_Iterator           is Ada_Program.Choice_Iterator;
    subtype Declarative_Part_Iterator is
       Ada_Program.
Declaration_Or_Context_Clause_Or_Representation_Clause_Or_Pragma_Iterator;
    subtype Name_Iterator             is Ada_Program.Name_Iterator;
    ---------------------------------------------------------------
```

```
type Type_Definition_Kinds is (A_Subtype_Indication,
                               An_Enumeration_Type_Definition,
                               An_Integer_Type_Definition,
                               A_Float_Type_Definition,
                               A_Fixed_Type_Definition,
                               An_Array_Type_Definition,
                               A_Record_Type_Definition,
                               An_Access_Type_Definition,
                               A_Derived_Type_Definition,
                               A_Task_Type_Definition,
                               A_Private_Type_Definition,
                               A_Limited_Private_Type_Definition,
                               Not_A_Type_Definition);

function Kind (A_Type_Definition : Type_Definition)
              return Type_Definition_Kinds;

function Parent_Declaration (Type_Def : Type_Definition) return Declaration;
-- Returns the declaration associated with the type definition, if
-- any is available.  Anonymous types for example have no associated
-- declaration.

function Base_Type (Type_Def : Type_Definition) return Type_Definition;
-- Returns the base type of the specified type definition as per LRM 3.3.
-- All subtypes are constraints applied to some base type.  This function
-- returns that base type.

function Type_Structure (Type_Def : Type_Definition) return Type_Definition;
-- Returns the type structure from which the specified type definition has
-- been derived.  This function will unwind recursive derivations until the
-- type definition derives a new representation or is no longer derived.
-- This function is different from GROUND_TYPE only for enumerations or
-- records that have derivations with rep specs.

function Ground_Type (Type_Def : Type_Definition) return Type_Definition;
-- This function recursively unwinds all type derivations and subtyping
-- to arrive at a type definition which is neither a derived type or a
-- subtype.

function Last_Constraint
            (Type_Def : Type_Definition) return Type_Definition;
-- This function recursively unwinds subtyping to arrive at a type
-- definition which is either the base_type or imposes constraints.

function Is_Predefined (Type_Def : Type_Definition) return Boolean;
-- returns true if the type definition is one of:
-- Boolean, Character, String, Integer, Natural, Positive, Float

function Is_Universal (Type_Def : Type_Definition) return Boolean;
```





```
-- returns true if the type definition is a universal integer,
-- universal fixed or universal float.

subtype Subtype_Indication           is Type_Definition;
subtype Enumeration_Type_Definition  is Type_Definition;
subtype Integer_Type_Definition      is Type_Definition;
subtype Float_Type_Definition        is Type_Definition;
subtype Fixed_Type_Definition        is Type_Definition;
subtype Array_Type_Definition        is Type_Definition;
subtype Record_Type_Definition       is Type_Definition;
subtype Access_Type_Definition       is Type_Definition;
subtype Derived_Type_Definition      is Type_Definition;
subtype Task_Type_Definition         is Type_Definition;


---------------------------------------------------------------
-- TYPE CONSTRAINTS:

subtype Type_Constraint                    is Ada_Program.Element;
subtype Discrete_Range_Iterator            is Ada_Program.Element_Iterator;
subtype Discriminant_Association_Iterator is Ada_Program.Element_Iterator;

type Type_Constraint_Kinds is (A_Simple_Range,
                               A_Range_Attribute,
                               A_Floating_Point_Constraint,
                               A_Fixed_Point_Constraint,
                               An_Index_Constraint,
                               A_Discriminant_Constraint,
                               Not_A_Constraint);

function Constraint_Kind (A_Constraint : Type_Constraint)
                         return Type_Constraint_Kinds;

function Discrete_Ranges (Of_Index_Constraint : Type_Constraint)
                         return Discrete_Range_Iterator;
-- Returns the list of Discrete_Range components of an Index_Constraint

function Discriminant_Associations
            (Of_Discriminant_Constraint : Type_Constraint)
            return Discriminant_Association_Iterator;
-- Returns the list of discriminant associations of
-- a Discriminant_Constraint.


-------------------------------------------------------------------
-- DISCRETE RANGES:

subtype Discrete_Range is Ada_Program.Element;  -- LRM 3.6
```

```
type Range_Kinds is (A_Simple_Range,
                     A_Range_Attribute,
                     A_Subtype_Indication,
                     Not_A_Range);

function Range_Kind (A_Discrete_Range : Discrete_Range) return Range_Kinds;

subtype Range_Info is Ada_Program.Element;

procedure Bounds (A_Range      :     Range_Info;
                  Lower, Upper : out Expression);
-- This procedure returns the simple expression for the
-- upper and lower bounds of a range if one is present
-- NIL_ELEMENTs otherwise.

-- Note that range attributes are expressions and can be analyzed
-- by using the attribute operations in NAMES_AND_EXPRESSIONS.


---------------------------------------------------------------
-- CHOICES:

subtype Choice is Ada_Program.Element;

type Choice_Kinds is (A_Simple_Expression,
                      A_Discrete_Range,
                      Others_Choice,
                      An_Identifier_Reference,
                      Not_A_Choice);

function Choice_Kind (A_Choice : Choice) return Choice_Kinds;

function Choice_Expression (A_Choice : Choice) return Expression;

function Choice_Range (A_Choice : Choice) return Discrete_Range;

function Choice_Identifier (A_Choice : Choice) return Identifier_Reference;


---------------------------------------------------------------
-- SUBTYPE_INDICATIONS & TYPE_MARKS - LRM 3.3.2

function Type_Mark (A_Subtype_Indication : Subtype_Indication)
                   return Name_Expression;
-- Returns the type mark of a subtype indication.
-- The NAMES_AND_EXPRESSION package provides selectors to do further
-- decomposition.
```





```
function Constraint (A_Subtype_Indication : Subtype_Indication)
                    return Type_Constraint;
-- Returns the constraint applied to the subtype indication.  A nil
-- element is returned if no constraint is present.


---------------------------------------------------------------
-- ENUMERATION TYPES - LRM 3.5.1

function Enumeration_Literals
            (Enumeration_Type : Enumeration_Type_Definition)
            return Name_Iterator;
-- Returns a list of the literals declared an enumeration type
-- declaration.  Each of these elements has a Name.


---------------------------------------------------------------
-- INTEGER TYPES - LRM 3.5.4

function Integer_Constraint
            (Integer_Type : Integer_Type_Definition) return Range_Info;
-- Returns the range constraint on the integer type declaration.


---------------------------------------------------------------
-- REAL TYPES - LRM 3.5.6

function Digits_Accuracy_Definition
            (Floating_Point_Type : Float_Type_Definition) return Expression;
-- Returns the digits accuracy definition of a floating point type
-- definition or floating point constraint.

function Floating_Point_Constraint
            (Floating_Point_Type : Float_Type_Definition) return Range_Info;
-- Returns the range constraint of a floating point type declaration.

function Delta_Accuracy_Definition
            (Fixed_Point_Type : Fixed_Type_Definition) return Expression;
-- Returns the delta accuracy definition of a fixed point type definition
-- or fixed point constraint.

function Fixed_Point_Constraint
            (Fixed_Point_Type : Fixed_Type_Definition) return Range_Info;
-- Returns the range constraint of the fixed point type definition.


---------------------------------------------------------------
-- ARRAY TYPES - LRM 3.6
```

```
function Is_Constrained_Array
           (Array_Type : Array_Type_Definition) return Boolean;

function Index_Constraints (Constrained_Array_Type : Array_Type_Definition)
                        return Discrete_Range_Iterator;
-- Returns a list of the discrete range constraints for an
-- constrained array type declaration.

function Index_Subtype_Definitions
           (Unconstrained_Array_Type : Array_Type_Definition)
           return Name_Iterator;
-- Returns a list of the Index_Subtypes (Type_Marks) for an
-- unconstrained array type declaration.

function Component_Type (Array_Type : Array_Type_Definition)
                      return Subtype_Indication;
-- Returns the specification of the array component type.


--------------------------------------------------------------
-- DISCRIMINANTS - LRM 3.7.1

subtype Type_Definition_Or_Declaration is Ada_Program.Element;
subtype Discriminant_Iterator          is Ada_Program.Element_Iterator;

function Is_Discriminated
           (A_Type : Type_Definition_Or_Declaration) return Boolean;
-- This function applies to private, limited private, incomplete or
-- record types.  It returns True if this type has discriminants.
-- It may be applied to type declaration to handle the case of
-- incomplete types.

function Discriminants (A_Type : Type_Definition_Or_Declaration)
                     return Discriminant_Iterator;
-- Returns a list of discriminants of the type.  These elements may
-- then be manipulated with the functions provided in package
-- Declarations.


--------------------------------------------------------------
-- RECORD TYPES - LRM 3.7

subtype Record_Component                   is Ada_Program.Element;
subtype Record_Component_Or_Pragma_Iterator is Ada_Program.Element_Iterator;
subtype Variant                            is Ada_Program.Element;
subtype Variant_Or_Pragma_Iterator         is Ada_Program.Element_Iterator;
```





```
function Record_Components (Record_Type : Record_Type_Definition)
                        return Record_Component_Or_Pragma_Iterator;
-- Returns a list of the record components of the record declaration.

type Component_Kinds is (A_Null_Component,
                         A_Variable_Component,
                         A_Variant_Part_Component,
                         Not_A_Component);
-- A component can be a Variable, NULL or Variant_Part.
-- A Null_Component is a NIL_ELEMENT.
-- The operations on Variables can be used to decompose Variable_Components.

function Component_Kind
           (Component : Record_Component) return Component_Kinds;

function Associated_Discriminant
           (Variant_Part : Record_Component) return Identifier_Reference;

function Variant_Item_List (Variant_Part : Record_Component)
                        return Variant_Or_Pragma_Iterator;
-- Returns a list of variants that make up the record component.

function Variant_Choices (Variant_Item : Variant) return Choice_Iterator;
-- Returns a list of the 'WHEN <choice> | <choice>' choices.
-- Use the above CHOICES queries to extract further information.

function Inner_Record_Components (Variant_Item : Variant)
                              return Record_Component_Or_Pragma_Iterator;
-- Returns a list of the record components of the inner record declaration.
-- Use Component_Kind to analyze further.


--------------------------------------------------------------
-- ACCESS TYPES - LRM 3.8

function Access_To (Access_Type : Access_Type_Definition)
                 return Subtype_Indication;
-- Returns the subtype indication associated with the access type.


--------------------------------------------------------------
-- DERIVED TYPES - LRM 3.4

function Derived_From (Derived_Type : Derived_Type_Definition)
                    return Subtype_Indication;
-- Returns the subtype indication associated with the derived type.
```

```
    ----------------------------------------------------------------
    -- TASK TYPE DEFINITIONS
    -- LRM Chapter 9

    function Task_Components (Task_Spec : Task_Specification)
                             return Declarative_Part_Iterator;
    -- Returns a list of entry declarations, representation clauses
    -- and pragmas in a task specification. The list is in order of appearance.
    -- The operations on subprogram declarations can be used to
    -- decompose task entries.

end Type_Information;
```





```
generic
    Size : Integer;
    -- number of buckets

    type Domain_Type is private;
    type Range_Type  is private;
    -- both types are pure values
    -- no initialization or finalization of values of either
    --     domain_type or range_type is necessary
    -- = and := can be used for equality and copy

    with function Hash (Key : Domain_Type) return Integer is <>;
    -- efficiency => spread hash over an interval at least as great as size

    pragma Must_Be_Constrained (Yes => Domain_Type, Range_Type);

package Map_Generic is

    type Map is private;

    type Pair is
        record
            D : Domain_Type;
            R : Range_Type;
        end record;

    function  Eval (The_Map : Map; D : Domain_Type) return Range_Type;
    procedure Find (The_Map :        Map;
                    D       :        Domain_Type;
                    R       : in out Range_Type;
                    Success : out    Boolean);
    procedure Find (The_Map :        Map;
                    D       :        Domain_Type;
                    P       : in out Pair;
                    Success : out    Boolean);

    procedure Define   (The_Map        : in out Map;
                        D              :        Domain_Type;
                        R              :        Range_Type;
                        Trap_Multiples :        Boolean := False);
    procedure Undefine (The_Map : in out Map; D : Domain_Type);

    procedure Initialize (The_Map : out Map);
    function  Is_Empty   (The_Map : Map) return Boolean;
    procedure Make_Empty (The_Map : in out Map);

    procedure Copy (Target : in out Map; Source : Map);
```

```
    type Iterator is private;

    procedure Init  (Iter : out Iterator; The_Map : Map);
    procedure Next  (Iter : in out Iterator);
    function  Value (Iter : Iterator) return Domain_Type;
    function  Done  (Iter : Iterator) return Boolean;

    Undefined : exception;
    -- raised by eval if the domain value in not in the map

    Multiply_Defined : exception;
    -- raised by define if the domain value is already defined and
    -- the trap_multiples flag has been specified (ie. is true)

    function Cardinality (The_Map : Map) return Natural;
    function Nil                         return Map;
    function Is_Nil      (The_Map : Map) return Boolean;

    ------------------------------------------------------
    -- Implementation Notes and Non-Standard Operations --
    ------------------------------------------------------

    -- := and = operate on references
    --       := implies sharing (introduces an alias)
    --        =  means is the same set, not the same value of type set
    -- Initializing a map also makes it empty
    -- Accessing an uninitialized map will raise CONSTRAINT_ERROR.

    -- garbage may be generated

    -- Concurrent Properties
    --    any number of find/eval/is_empty may be safely done while one
    --        define/undefine is taking place.  If the define is redefining an
    --        existing element in the domain of the map, concurrrent reading is
    --        safe if and only if := on range_type is atomic.

private
    type Node;
    type Set is access Node;

    type Node is
        record
            Value : Pair;
            Link  : Set;
        end record;

    subtype Index is Integer range 0 .. Size - 1;

    type Table is array (Index) of Set;
```





```
    type Map_Data is
        record
            Bucket : Table;
            Size   : Integer := 0;
        end record;

    type Map is access Map_Data;

    type Iterator is
        record
            The_Map     : Map;
            Index_Value : Index;
            Set_Iter    : Set;
            Done        : Boolean;
        end record;

end Map_Generic;
```

```
with System;

package Byte_Defs is

    -- This package defines bytes, byte_strings, and their operators.
    -- They are defined here to facilitate portability: the base types
    -- are target-dependent, and renaming them here localizes this
    -- target dependency in a single place.

    subtype Byte is System.Byte;
    -- An unsigned 8-bit value, range 0 .. 255.

    function "="  (X, Y : Byte) return Boolean renames System."=";
    function ">"  (X, Y : Byte) return Boolean renames System.">";
    function "<"  (X, Y : Byte) return Boolean renames System."<";
    function ">=" (X, Y : Byte) return Boolean renames System.">=";
    function "<=" (X, Y : Byte) return Boolean renames System."<=";

    function "+"   (X, Y : Byte) return Byte renames System."+";
    function "-"   (X, Y : Byte) return Byte renames System."-";
    function "*"   (X, Y : Byte) return Byte renames System."*";
    function "/"   (X, Y : Byte) return Byte renames System."/";
    function "mod" (X, Y : Byte) return Byte renames System."mod";
    function "rem" (X, Y : Byte) return Byte renames System."rem";

    subtype Byte_String is System.Byte_String;
    -- An array (integer range <>) of Bytes.

    function "="  (X, Y : Byte_String) return Boolean renames System."=";
    function ">"  (X, Y : Byte_String) return Boolean renames System.">";
    function "<"  (X, Y : Byte_String) return Boolean renames System."<";
    function ">=" (X, Y : Byte_String) return Boolean renames System.">=";
    function "<=" (X, Y : Byte_String) return Boolean renames System."<=";

    function "&" (X, Y : Byte_String) return Byte_String renames System."&";
    function "&" (X : Byte; Y : Byte_String) return Byte_String
        renames System."&";
    function "&" (X : Byte_String; Y : Byte) return Byte_String
        renames System."&";
    function "&" (X : Byte; Y : Byte) return Byte_String renames System."&";

end Byte_Defs;
```





```
with Byte_Defs;
with Text_Io;

package Byte_String_Io is

    subtype Byte_String is Byte_Defs.Byte_String;
    subtype File_Type   is Text_Io.File_Type;

    function Image (Item : Byte_String) return String;

    procedure Put (Item : Byte_String);
    procedure Put (File : File_Type; Item : Byte_String);

    procedure Get (Item : out Byte_String);
    procedure Get (File : File_Type; Item : out Byte_String);

    function Get                      return Byte_String;
    function Get (File : File_Type) return Byte_String;

end Byte_String_Io;
```

```
with Ftp_Defs;
with Transport_Defs;
with Machine;

package File_Transfer is

    type Connect_Id is private;

    --- handle for all communications to ftp transfer service


    Socket_21           : constant                           := 21;
    Default_Ftp_Socket : constant Transport_Defs.Socket_Id :=
       (1 => 0, 2 => Socket_21);


    procedure Open (Connection : out Connect_Id;
                    Status     : out Transport_Defs.Status_Code);

    ---- allocate a connection and initialize it;
    ---- May fail due to resource limitations and/or networking
    ---- problems


    procedure Close (Connection : Connect_Id);

    ---- Release connection.
    ---- Returned to pool for others requesters


    procedure Get_Owner (Connection : Connect_Id; Owner : out Machine.Job_Id);

    ---- return the current owner of the connection


    procedure Set_Owner (Connection : in Connect_Id; Owner : Machine.Job_Id);
    ---- change the owner of the connection

    ---- Connection with FTP have knowledge of the job which opened
    ---- the connect.  It detects the death of the job and reclaims the
    ---- resources.  If a user wishes to pass the connect_id between
    ---- jobs and have it be used then it should update the owner
    ---- the the jobs or a job which will be running the full time
    ---- the connection is in use.
```





```
    procedure Connect (Connection     : Connect_Id;
                       To_Remote_Host : Transport_Defs.Host_Id;
                       Remote_Socket  : Transport_Defs.Socket_Id :=
                          Default_Ftp_Socket);

    ---- Initiate a connection to the host and socket specified
    ---- The outcome of this operations is checked by using the
    ---- COMMAND_STATUS and READ_RESPONSE procedures


    procedure Disconnect (Connection : Connect_Id);

    ---- Break the current Ethernet connection
    ---- Used in case of a problem otherwise the SEND_QUIT command
    ---- should be sent.


    procedure Read_Response (Connection :     Connect_Id;
                             Message    : out String;
                             Count      : out Natural);

    ---- Return the text messages recieved from the FTP server.  The
    ---- messages will still contain the code values found at the start of
    ---- the messages.  The messages are line oriented.  Count is the number
    ---- of characters in the message.
    ---- Read_response will block waiting for the next message or
    ---- the end of the command.
    ----    Note: A string length of 80 or more
    ---- is recommended to get full lines of the message;
    ---- If no command is active then a message of length zero is returned


    function End_Of_Line (Connection : Connect_Id) return Boolean;

    ---- Returns true when the end of a given message line is reached.
    ---- If no command is active and the fucntion is called true
    ---- is returned.


    function End_Of_Response (Connection : Connect_Id) return Boolean;

    ---- When all response messages for a FTP request have been read
    ---- END_OF_RESPONSE  will be TRUE.  If no command is active and
    ---- the function is called True is again returned.


    procedure Command_Status (Connection :     Connect_Id;
                              Status     : out Ftp_Defs.Status_Code);
```

```
---- Return the final status of the last operation performed.
---- If COMMAND_STATUS is called before all responses have been
---- read they will be discarded.


procedure File_Transfer_Status (Connection :      Connect_Id;
                                Status      : out Ftp_Defs.Transfer_Status);

---- Return status the current status of the most recent
---- file transfer


procedure Send_Username (Connection : Connect_Id; Username : String);

---- Send username information to the FTP server.
---- The command should be the first command issued after a successful
---- connect.  (ie REQUEST_STATUS returns a status of successful);


procedure Send_Password (Connection : Connect_Id; Password : String);

---- Send the password associated with the username previously sent.
---- A command status of NEED_PASSWORD returned after the user command
---- denotes the the password information is needed.


procedure Send_Account (Connection : Connect_Id; Account : String);

---- Some FTP servers require Account information for some file
---- operations.


procedure Send_Quit (Connection : Connect_Id);

---- Send to command to the server informing it that the session is
---- logging off. This will cause the connection to be disconnected.


procedure Set_Type (Connection : Connect_Id; New_Type : Ftp_Defs.Type_Code);

---- Send request for transfer type setting to be change to
---- specified value.  First the local server checks to see that
---- it supports the TYPE if it does the request is sent.  If a
---- positive response is recieved then the local setting is updated.


procedure Set_Mode (Connection : Connect_Id; New_Mode : Ftp_Defs.Mode_Code);

---- Send request for transfer mode setting to be change to
```





```
---- specified value.  First the local server checks to see that
---- it supports the MODE if it does the request is sent.  If a
---- positive response is recieved then the local setting is updated.


procedure Set_Structure (Connection    : Connect_Id;
                         New_Structure : Ftp_Defs.Structure_Code);

---- Send request for transfer mode setting to be change to
---- specified value.  First the local server checks to see that
---- it supports the MODE if it does the request is sent.  If a
---- positive response is recieved then the local setting is updated.


procedure Set_Allocation
           (Connection : Connect_Id; Pages : Natural; Records : Natural);

---- Some machines will require that space be preallocated to transfer
---- data.


procedure Send_Delete (Connection : Connect_Id; Remote_Filename : String);

---- Send command across link requesting the remote machine to
---- delete the named file.


procedure Send_Cwd (Connection : Connect_Id; Remote_Pathname : String);

---- Send command across link requesting the remote machine to
---- update its current working directory to the pathname specified


function Pio_File_Of (Connection : Connect_Id) return Ftp_Defs.Pio_Pointer;

function Dio_File_Of (Connection : Connect_Id) return Ftp_Defs.Dio_Pointer;

---- Return to pointer for the files used by FTP.  The FTP
---- transfer routines handle control of the file handle for the
---- local files.  The functions above give the user visibility to
---- these handles so that their program can also work with the
---- file without having to pass the name around.


procedure Start_Store (Connection      : Connect_Id;
                       Remote_Filename : String;
                       Append          : Boolean := False);

---- Start the process for sending a file across the network data link
```

```
---- This form assumes the user has already opened the file for reading
---- (Using either poly_io for binary transfers or device_independent_io
---- for character and byte transfers) on the file_pointers supplied by
---- the pio_of
---- Append specifies that the data sent is to be appened to the
---- contents of the remote file if it exist.


procedure Start_Store (Connection      : Connect_Id;
                       Local_Filename  : String;
                       Remote_Filename : String;
                       Append          : Boolean := False);

---- Similiar to the previous store, only this version opens the file
---- for the user using the name supplied in local_filename.
---- the current transfer type setting (binary,ascii ...etc) determines
---- which type of open will be used.


procedure Start_Retrieve (Connection      : Connect_Id;
                          Remote_Filename : String);

---- Request that the remote server send a copy of the file specified
---- by remote_filename be sent across the data connect and placed
---- in the local file which the use has opened using the appropriate
---- open operations with the file handle made visible by the functions
---- above


procedure Start_Retrieve (Connection      : Connect_Id;
                          Local_Filename  : String;
                          Remote_Filename : String;
                          Append          : Boolean := False);

---- Request that the remote server send a copy of the file specified
---- by remote_filename be sent across the data connect and placed
---- in the local file.  The local file is opened be the FTP transfer
---- worker using the local_filename supplied by the user.
---- Append specifies that the data transfered is to be appended to
---- the contents of the local file.


procedure Start_Directory_List (Connection      : Connect_Id;
                                Remote_Pathname : String;
                                Verbose         : Boolean := False);

---- Request that the remote server send directory/file information
---- for the directory/file specified by remote_pathname.
---- The list is placed in the local file which the user has
```





```
---- opened via the dio_file_of.
---- The parameter verbose if true specifies is list of files and
---- file information is to be sent.  If verbose is false only a
---- list of filename matching the remote_pathname is sent.


procedure Start_Directory_List (Connection      : Connect_Id;
                                Local_Filename  : String;
                                Remote_Pathname : String;
                                Verbose         : Boolean := False);

---- Request that the remote server send directory/file information
---- for the directory/file specified by remote_pathname.
---- The list is placed in the local_filename.
---- The parameter verbose if true specifies is list of files and
---- file information is to be sent.  If verbose is false only a
---- list of filename matching the remote_pathname is sent.


procedure Send_Data_Port (Connection : Connect_Id;
                          Host       : Transport_Defs.Host_Id;
                          Socket     : Transport_Defs.Socket_Id);

---- Send information to the FTP server telling it what port
---- it should be sending the data it transfers to.


function Data_Host_Id (Connection : Connect_Id)
                      return Transport_Defs.Host_Id;

function Data_Socket_Id (Connection : Connect_Id)
                        return Transport_Defs.Socket_Id;

---- Returns the host and socket id for that data connect being used
---- on the current connection.  These should be used to get
---- the information to be sent in the SEND_DATA_PORT command


procedure Send_Help_Request
             (Connection : Connect_Id; Help_On : String := "");

---- Request the remote ftp server to return help information cross the
---- command link.  The response is read as usual with the read_response
---- procedure.


procedure Send_Status_Request
             (Connection : Connect_Id; Status_On : String := "");
```

```
    ---- Request the remote ftp server to return status information for
    ---- the specified argument.  For the null argument the server should
    ---- send back general status.  A none null argument the server will
    ---- try to send back file/directory information for the specified
    ---- argument.


    procedure Send_Verbatim (Connection : Connect_Id; Command : String);

    ---- Send the arguement COMMAND across the command connect unchanged.
    ---- This comand must be a simple command which has no side affects,
    ---- such as causing the server to try to transfer a file.


    procedure Send_Site_Command (Connection : Connect_Id; Argument : String);

    ---- Some servers have site specific options.  Allow user
    ---- to send site command. The local server will register positive
    ---- or negative completion but as in the SEND_VERBATIM command
    ---- no local processing from the response occurs. (ie. local parameters
    ---- remain unchanged.


    procedure Send_Pasv (Connection : Connect_Id);

    ---- Request remote to go into pasv mode on its data connect
    ---- Remote will return identity of this port and this information
    ---- the local server will attempt to read this information and
    ---- it is queried using functions PASV_DATA_HOST and PASV_DATA_SOCKET
    ---- Since the response varies betweenimplementations the
    ---- local server may not be able to interpret the response.
    ---- It may be necessary for the user to visually inspect the
    ---- response.


    function Pasv_Data_Host (Connection : Connect_Id)
                            return Transport_Defs.Host_Id;

    function Pasv_Data_Socket (Connection : Connect_Id)
                              return Transport_Defs.Socket_Id;

    --- host and socket information for last SEND_PASV request
    --- if the response could not be parsed TRANSPORT_DEFS.NULL_HOST_ID,
    --- and TRANSPORT_DEFS.NULL_SOCKET_ID will be returned.


    function Is_Open (Connection : Connect_Id) return Boolean;

    function Is_Connected (Connection : Connect_Id) return Boolean;
```





```
    function Is_Logged_In (Connection : Connect_Id) return Boolean;

    function Current_Type (Connection : Connect_Id) return Ftp_Defs.Type_Code;

    function Current_Mode (Connection : Connect_Id) return Ftp_Defs.Mode_Code;

    function Current_Structure (Connection : Connect_Id)
                               return Ftp_Defs.Structure_Code;

    function Command_Is_Active (Connection : Connect_Id) return Boolean;

    function Most_Recent_Command (Connection : Connect_Id)
                                 return Ftp_Defs.Ftp_Commands;

    function Most_Recent_Command_Status
                (Connection : Connect_Id) return Ftp_Defs.Status_Code;

    function Most_Recent_Response_Code (Connection : Connect_Id) return Natural;

    function Transfer_Is_Active (Connection : Connect_Id) return Boolean;

    function Most_Recent_Transfer_Status
                (Connection : Connect_Id) return Ftp_Defs.Transfer_Status;

    function Last_Transfer_Length (Connection : Connect_Id) return Natural;

    function Last_Transfer_Time (Connection : Connect_Id) return Duration;

    function Remote_Host_Id (Connection : Connect_Id)
                            return Transport_Defs.Host_Id;

    Null_Connect_Id : constant Connect_Id;

end File_Transfer;
```

```
with Polymorphic_Io;
with Device_Independent_Io;

package Ftp_Defs is

    type Pio_Handle is
        record
            File_Handle : Polymorphic_Io.Handle;
            Position    : Polymorphic_Io.File_Position;
        end record;

    type Pio_Pointer is access Pio_Handle;

    type Dio_Pointer is access Device_Independent_Io.File_Type;

    type Mode_Code is (Stream, Block, Compressed);

    type Structure_Code is (File, Recrd, Page);

    type Type_Code is (Ascii, Ebcdic, Image, Binary, Local_Binary, Local_Byte,
                       Ascii_Cc, Ebcdic_Cc, Ascii_Telnet, Ebcdic_Telnet);

    Default_Mode      : Mode_Code      := Stream;
    Default_Structure : Structure_Code := File;
    Default_Type      : Type_Code      := Ascii;

    type Ftp_Commands is

       (Login,             -- forming initial connection
        Set_User,          -- set user information
        Set_Pass,          -- set passwork information
        Set_Account,       -- set account information
        Set_Type,          -- set transfer type information
        Set_Stru,          -- set transfer structure information
        Set_Mode,          -- set transfer mode information
        Set_Allocation,    -- set space allocation
        Send_File,         -- send file to remote
        Send_File_Append,  -- send file and append to remote file
        Retr_File,         -- retrieve file from remote
        List_Directory,    -- get verbose directory listing
        Nlst_Directory,    -- get name list of directory
        Do_Cwd,            -- change remote current working directory
        Do_Delete,         -- delete file at remote host
        Do_Help,           -- get help from remote
        Do_Stat,           -- get status from remote
        Do_Port,           -- send data port information
        Do_Pasv,           -- request pasv data opeation
```





```
        Do_Site,           -- sent site command
        Verbatim,          -- command sent no local processing of response
        Do_Quit,           -- log off
        Noop);             -- no affect

    type Status_Code is

       (Command_In_Progress,      -- Command active pending response

        Successful,               -- command completed successfully

        Need_Password,            -- command successful need password next

        Need_Account,             -- need account info to procede

        Not_Used,                 -- command not needed (ie no account needed)

        Transfer_Started,         -- transfer has started

        Transfer_Complete,        -- transfer has completed successfully

        Transfer_Failed,          -- transfer failed refer to transfer
                                  -- status for reason

        Not_Logged_In,            -- command failed user not logged in.

        Syntax_Error,             -- command syntax error (ie line too long)

        Bad_Sequence,             -- sequenced commands (ie user - password)
                                  -- were done out of sequence

        No_Local_Support,         -- The local transfer worker does not
                                  -- support command parameter

        Command_Not_Implemented,  -- remote server does not implement the
                                  -- command

        Param_Not_Implemented,    -- remote server does not support given
                                  -- parameter

        Local_Pasv_Error,         -- a send_pasv request got a successful
                                  -- response from the remote server but
                                  -- the local server could not interpret
                                  -- the returned port information

        Remote_Directory_Error,   -- file or directory access error
```

```
        Timed_Out,                -- No response was recieved in a
                                  -- appropriate time

        Network_Error,            -- Some problem has occured on network
                                  -- The connection may have been lost
        Invalid_Use,              -- The connection is not open or the
                                  -- caller is not a valid user

        Unknown_Error);           -- An error response which could not
                                  -- be clasified was recieved

    type Transfer_Status is

       (In_Progress,             -- transfer is currently active

        Ok,                       -- transfer completed successfully

        Local_Error,              -- transfer failed due to local error

        File_Error,               -- transfer failed due to local file error

        Line_Error,               -- problem with data line,
                                  -- transfer could not complete.

        Open_Failed,              -- could not get data connect open
                                  -- to start transfer

        Transfer_Abort,           -- file transfer has been aborted

        Remote_File_Unavailable,  -- file unavailable at remote site

        Remote_File_Error,        -- remote file error

        Storage_Error,            -- Storage error at remote

        Unknown_Page_Type,        -- remote did not recognize page type

        Filename_Bad,

        Comm_Line_Error,          -- Command link lost during transfer

        Unknown_Error);           -- An error response which could not be
                                  -- classified was recieved

    function Ftp_Product_Is_Installed return Boolean;
    Ftp_Product_Is_Not_Installed : exception;

end Ftp_Defs;
```





```
package Ftp_Name_Map is

    type Machine_Type is new String;

    function Local_To_Remote (File_Name   : String;
                              Local_Roof  : String;
                              Remote_Roof : String;
                              Remote_Type : Machine_Type) return String;

    function Remote_To_Local (File_Name   : String;
                              Local_Roof  : String;
                              Remote_Roof : String;
                              Remote_Type : Machine_Type) return String;

    Rational : constant Machine_Type := "Rational";
    Unix     : constant Machine_Type := "Unix";
    Aos      : constant Machine_Type := "AOS";
    Vms      : constant Machine_Type := "VMS";
    Mvs      : constant Machine_Type := "MVS";

end Ftp_Name_Map;



package Ftp_Product is

    function Is_Installed return Boolean;

    Is_Not_Installed : exception;

end Ftp_Product;
```

```
with Ftp_Defs;
with Ftp_Name_Map;
with Profile;

package Ftp_Profile is

    function Remote_Machine   return String;
    function Remote_Type      return Ftp_Name_Map.Machine_Type;
    function Remote_Directory return String;
    function Remote_Roof      return String;

    function Auto_Login return Boolean;

    function Username return String;
    function Password return String;
    function Account  return String;

    function Transfer_Type      return Ftp_Defs.Type_Code;
    function Transfer_Mode      return Ftp_Defs.Mode_Code;
    function Transfer_Structure return Ftp_Defs.Structure_Code;

    function Send_Port_Enabled return Boolean;
    -- Always tell the remote server what port to use.

    function Prompt_For_Password return Boolean;
    function Prompt_For_Account  return Boolean;

    function Current_Remote_Type return Ftp_Name_Map.Machine_Type;
    function Current_Remote_Roof return String;

    function Profile_Get return Profile.Response_Profile renames Profile.Get;
    -- This declaration is introduced to avoid a name collision.

    procedure Show (Response : Profile.Response_Profile := Profile.Get);

end Ftp_Profile;
```





```
with Transport_Defs;
with File_Transfer;

package Ftp_Server is

    procedure Start (Network : Transport_Defs.Network_Name := "tcp/ip";
                     Socket  : Transport_Defs.Socket_Id    :=
                        File_Transfer.Default_Ftp_Socket);

    -- Start a new Ftp server.  Network identifies the specific network
    -- it is to use.  Socket  identifies the connection on which to
    -- accept requests.

end Ftp_Server;
```

```
with Text_Io;
with Transport_Defs;

package Host_Id_Io is

    procedure Put (Item : Transport_Defs.Host_Id);
    procedure Put (File : Text_Io.File_Type; Item : Transport_Defs.Host_Id);

    function Get                             return Transport_Defs.Host_Id;
    function Get (File : Text_Io.File_Type) return Transport_Defs.Host_Id;

    function Image (Item : Transport_Defs.Host_Id) return String;

end Host_Id_Io;
```

```
with Byte_Defs;
with Calendar;
with Interchange_Defs;

package Interchange is

    -- This package defines a collection of types for data interchange.
    -- Each type has some local representation, which is dependent on
    -- the characteristics of the local architecture/compiler; and also
    -- an interchange representation (a canonical byte sequence), which
    -- is machine-independent.  This package provides operations for
    -- converting between the local and interchange representations.

    -- The types provided are similar to types in the Ada predefined
    -- language environment, i.e. packages STANDARD and CALENDAR.

    -- The conversion operations are generic on a procedural byte
    -- stream, so that they may be used on a variety of interchange
    -- media, e.g. communication channels, tapes, and disks.

    -- The algorithms for conversion to and from the interchange form
    -- are extensible to any Ada type except access values and task
    -- types.  The rules for forming new type conversion algorithms
    -- are given below.

    -- The interchange representation does not carry type information,
    -- i.e. the types of the interchanged data must be known a priori
    -- in order to convert them back to the local representation.

    -- The interchange representation mostly follows the rules of
    -- Courier, the Xerox System Integration Standard for a remote
    -- procedure call protocol.  In particular, all interchange
    -- values are a multiple of two bytes (16 bits) in length.


    Constraint_Error : exception;

    -- Raised when an interchange conversion procedure encounters
    -- an out-of-range value.


    subtype Byte        is Byte_Defs.Byte;
    subtype Byte_String is Byte_Defs.Byte_String;


    type    Short_Integer  is range -(2 ** 14) - (2 ** 14) ..
                                     (2 ** 14) + ((2 ** 14) - 1);
```





```
    subtype Short_Natural  is Short_Integer range 0 .. Short_Integer'Last;
    subtype Short_Positive is Short_Integer range 1 .. Short_Integer'Last;


    type    Integer  is new Interchange_Defs.Longest_Integer
                        range -(2 ** 30) - (2 ** 30) ..
                               (2 ** 30) + ((2 ** 30) - 1);
    subtype Natural  is Integer range 0 .. Integer'Last;
    subtype Positive is Integer range 1 .. Integer'Last;


    type    Long_Integer  is new Interchange_Defs.Longest_Integer;
    subtype Long_Natural  is Long_Integer range 0 .. Long_Integer'Last;
    subtype Long_Positive is Long_Integer range 1 .. Long_Integer'Last;

    -- The range of long_integer is machine-dependent.  Each
    -- implementation should choose the largest possible range.
    -- The interchange form (which is machine-independent),
    -- can express values up to +- 2 ** (2 ** 16 - 1) - 1.


    type Float is new Interchange_Defs.Float;

    type Long_Float is new Interchange_Defs.Long_Float;

    subtype Nanosecond_Count is Interchange.Natural range 0 .. 10 ** 9 - 1;

    type Duration is
        record
            Seconds     : Interchange.Integer;
            Nanoseconds : Interchange.Nanosecond_Count;
        end record;

    function Convert (X : Standard.Duration) return Interchange.Duration;

    function Convert (X : Interchange.Duration) return Standard.Duration;

    subtype Year_Number  is Interchange.Short_Integer;
    subtype Month_Number is Interchange.Short_Integer range 1 .. 12;
    subtype Day_Number   is Interchange.Short_Integer range 1 .. 31;
    subtype Day_Duration is Interchange.Duration;

    type Time is
        record
            Year    : Year_Number;
            Month   : Month_Number;
            Day     : Day_Number;
            Seconds : Day_Duration;
        end record;
```

```
    function Convert (X : Calendar.Time) return Interchange.Time;

    function Convert (X : Interchange.Time) return Calendar.Time;


    generic
        type Stream_Id is limited private;

        -- Identifies an interchange medium.

        with procedure Put (Into : Stream_Id; Data : Byte_String) is <>;

        -- Put the given DATA into the given stream.

        with procedure Get (From : Stream_Id; Data : out Byte_String) is <>;

        -- Get the first DATA'LENGTH bytes from the given stream.

        -- The interchange medium must not lose bytes, and must
        -- preserve byte ordering.  Also, it must fragment and
        -- reassemble.  That is, the sequence of calls:
        --
        --     PUT (STREAM, DATA (1 .. N));
        --     PUT (STREAM, DATA (N + 1 .. LAST));
        --
        -- must have the same effect as the single call
        --
        --     PUT (STREAM, DATA (1 .. LAST));
        --
        -- for all values of N in the range 1 .. LAST.
        -- Similarly, the sequence of calls
        --
        --     GET (STREAM, DATA (1 .. N));
        --     GET (STREAM, DATA (N + 1 .. LAST));
        --
        -- must have the same effect as the single call
        --
        --     GET (STREAM, DATA (1 .. LAST));
        --
        -- for all values of N in the range 1 .. LAST.

        -- Note that a STREAM_ID is passed as an 'in' parameter.
        -- This simplifies the interchange of unconstrained types
        -- (since they can be returned as the value of a function),
        -- but may require the instantiator to invent a level of
        -- indirection.
```





```
    package Operations is

        -- Given a facility for interchanging bytes, this package
        -- provides a facility for interchanging values of other
        -- types.  For each type, PUT and GET operations are
        -- defined: PUT converts any value to a sequence of
        -- bytes, and GET reconstructs the original value.

        -- Exceptions raised by the primitive operations are
        -- propagated to the caller of the derived operations.


        -- A short_integer is represented in 16-bit two's complement,
        -- split into two bytes, most significant byte first.

        procedure Put (Into : Stream_Id; Data : Interchange.Short_Integer);
        procedure Get (From : Stream_Id; Data : out Interchange.Short_Integer);


        -- An integer is represented in 32-bit two's complement,
        -- split into four bytes, most significant byte first.

        procedure Put (Into : Stream_Id; Data : Interchange.Integer);
        procedure Get (From : Stream_Id; Data : out Interchange.Integer);


        -- A long_integer is represented in arbitrary - precision two's
        -- complement, that is, the two's complement representation,
        -- split into as many bytes as needed, represented as a Byte_String,
        -- most significant byte first.  See the Put & Get routines for
        -- Byte_Strings, below.

        -- GET may raise CONSTRAINT_ERROR if the local implementation
        -- of long_integer cannot represent a gotten value.

        procedure Put (Into : Stream_Id; Data : Interchange.Long_Integer);
        procedure Get (From : Stream_Id; Data : out Interchange.Long_Integer);


        -- Floating point types are represented in IEEE format.
        -- A Float (single-precision) occupies 4 bytes, and
        -- a Long_Float (double-precision) occupies 8 bytes.
        -- The sign bit goes in the MSB of the first byte, followed by
        -- the exponent, followed by the fraction, with the LSB of the
        -- fraction going in the LSB of the last byte.

        procedure Put (Into : Stream_Id; Data : Interchange.Float);
        procedure Get (From : Stream_Id; Data : out Interchange.Float);
```

```
procedure Put (Into : Stream_Id; Data : Interchange.Long_Float);
procedure Get (From : Stream_Id; Data : out Interchange.Long_Float);


-- The types for time are represented using the rules
-- for records, described below.

procedure Put (Into : Stream_Id; Data : Interchange.Time);
procedure Get (From : Stream_Id; Data : out Interchange.Time);

procedure Put (Into : Stream_Id; Data : Interchange.Duration);
procedure Get (From : Stream_Id; Data : out Interchange.Duration);


-- A Boolean is represented by the value 0 for false and
-- 1 for true, converted to an interchange.short_natural.

procedure Put (Into : Stream_Id; Data : Standard.Boolean);
procedure Get (From : Stream_Id; Data : out Standard.Boolean);


-- A Byte is represented by its 'pos, converted to an
-- interchange.short_natural (or, if you like, padded
-- on the left with a 0 byte).

procedure Put (Into : Stream_Id; Data : Byte);
procedure Get (From : Stream_Id; Data : out Byte);


-- A CHARACTER is represented like a byte.

procedure Put (Into : Stream_Id; Data : Standard.Character);
procedure Get (From : Stream_Id; Data : out Standard.Character);


-- The GET operation for an unconstrained type is a function,
-- rather than a procedure.  This is because Ada rules don't
-- allow the caller of GET to declare an unconstrained
-- variable of the type.  Casting GET as a function allows
-- the caller to declare a constant of the type, and
-- initialize it with the value of the GET function, e.g.:
--
-- declare
--     S : constant STRING := GET_STRING (STREAM);
-- begin
--     ...
```





```
-- A STRING is represented by its 'LENGTH, followed by the
-- 'POS of each of its elements (in ascending index order),
-- followed by a 0 byte if required to pad the whole thing to
-- an even number of bytes.  The 'LENGTH is represented as a
-- INTERCHANGE.NATURAL.  Each element is represented as one
-- byte.  This representation loses the 'FIRST of the STRING,
-- much as does slice assignment.

procedure Put_String (Into : Stream_Id; Data : Standard.String);
function  Get_String (From : Stream_Id) return Standard.String;


-- A Byte_String is represented like a String.

procedure Put_Byte_String (Into : Stream_Id; Data : Byte_String);
function  Get_Byte_String (From : Stream_Id) return Byte_String;


-- Interchange forms for other types may be defined using the
-- following rules.  In some cases, the rules are implemented
-- by generics.

-- A record type is represented by the sequence of its fields,
-- in the order of their declaration.  In discriminated records,
-- fields which are inaccessible are omitted.

-- A discrete type (integer subtype or enumeration type)
-- is represented by its 'POS, converted to a short_integer.

generic
    type Discrete_Type is (<>);

package Discrete is

    procedure Put (Into : Stream_Id; Data : Discrete_Type);

    procedure Get (From : Stream_Id; Data : out Discrete_Type);

end Discrete;


-- A vector (one-dimensional array) is represented by its
-- 'LENGTH, followed by its elements in index order.  The
-- 'LENGTH is represented as an Interchange.Natural.  This
-- representation loses information about the 'FIRST of the
-- vector, much like slice assignment.
```

```
        generic
            type Element_Type is private;

            with procedure Put (Into : Stream_Id; Data : Element_Type) is <>;
            with procedure Get (From :     Stream_Id;
                                Data : out Element_Type)              is <>;

            type Index_Type is (<>);
            with procedure Put (Into : Stream_Id; Data : Index_Type)     is <>;
            with procedure Get (From : Stream_Id; Data : out Index_Type) is <>;

            type Vector_Type is array (Index_Type range <>) of Element_Type;
        package Vector is

            procedure Put (Into : Stream_Id; Data : Vector_Type);

            function Get (From : Stream_Id) return Vector_Type;

        end Vector;

    end Operations;

end Interchange;
```





```
package Interchange_Defs is

    -- This package defines fundamental types and operators
    -- for data interchange.  When porting this package, you
    -- should modify it to indicate the appropriate types on
    -- your system.

    subtype Longest_Integer is Standard.Long_Integer;
    -- The longest (highest-precision) supported signed integer.
    -- If you've got an arbitrary-precision package, that's ideal.
    -- See INTERCHANGE.LONG_INTEGER.

    function "=" (X, Y : Longest_Integer) return Boolean renames Standard."=";
    function ">" (X, Y : Longest_Integer) return Boolean renames Standard.">";
    function "<" (X, Y : Longest_Integer) return Boolean renames Standard."<";
    function ">=" (X, Y : Longest_Integer) return Boolean renames Standard.">=";
    function "<=" (X, Y : Longest_Integer) return Boolean renames Standard."<=";
    function "+" (X, Y : Longest_Integer) return Longest_Integer
        renames Standard."+";
    function "-" (X, Y : Longest_Integer) return Longest_Integer
        renames Standard."-";
    function "*" (X, Y : Longest_Integer) return Longest_Integer
        renames Standard."*";
    function "/" (X, Y : Longest_Integer) return Longest_Integer
        renames Standard."/";
    function "mod" (X, Y : Longest_Integer) return Longest_Integer
        renames Standard."mod";
    function "rem" (X, Y : Longest_Integer) return Longest_Integer
        renames Standard."rem";

    type Float      is new Standard.Float
                              --       digits 8
                              range -2.0 * (2.0 ** 127) .. 2.0 * (2.0 ** 127);
    type Long_Float is new Standard.Float;
    --       digits 15
    --       range - 2.0  * (2.0 ** 1023) .. 2.0 * (2.0 ** 1023);
    -- These definitions are intended to represent IEEE floating
    -- point single- and double-precision formats, respectively.
    -- Float'last is IEEE infinity, float'first is IEEE -infinity.

    function Duration_Magnitude return Standard.Integer;
    -- This should be the largest integral power of 10
    -- which is <= Standard.Duration'Last, <= 10 ** 9,
    -- and <= Integer'Last.  It may be a constant.
    -- It is used by the body of INTERCHANGE.CONVERT.

end Interchange_Defs;
```

```
with Transport_Defs;

package Network_Product is

    function Is_Installed (Network : Transport_Defs.Network_Name := "TCP/IP")
                          return Boolean;

    Is_Not_Installed : exception;

end Network_Product;
```





```
with Interchange;
with Transport_Stream;

package Rpc is

    type Version_Number is new Interchange.Short_Integer;

    type Version_Range is
        record
            First, Last : Version_Number;
        end record;

    type Transaction_Id is new Interchange.Short_Integer;

    type Program_Number is new Interchange.Integer;

    type Procedure_Number is new Interchange.Short_Integer;

    type Error_Type is (Error_Other, Error_Constraint, Error_Numeric,
                        Error_Program, Error_Storage, Error_Tasking,
                        Status_Error, Mode_Error, Name_Error, Use_Error,
                        Device_Error, End_Error, Data_Error, Layout_Error,
                        Error_Server_Defined, Error_Username_Or_Password);

    -- for Error_Type use
    --     (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15);

    type Reject_Kind is (Rej_No_Such_Program, Rej_No_Such_Version,
                         Rej_No_Such_Procedure, Rej_Invalid_Argument);

    -- for Reject_Kind use (0, 1, 2, 3);

    type Reject_Details (Kind : Reject_Kind := Rej_Invalid_Argument) is
        record
            case Kind is
                when Rej_No_Such_Version =>
                    Supported : Version_Range;
                when others =>
                    null;
            end case;
        end record;

    type Message_Kind is (Call_Message, Reject_Message,
                          Return_Message, Abort_Message);

    -- for Message_Kind use (0, 1, 2, 3);
```

```
type Message_Header (Kind : Message_Kind := Return_Message) is
    record
        Id : Transaction_Id := 0;
        case Kind is
            when Call_Message =>
                Program : Program_Number;
                Version : Version_Number;
                Proc    : Procedure_Number;
                -- argument values follow
            when Reject_Message =>
                Details : Reject_Details;
            when Return_Message =>
                null;
                -- result values follow
            when Abort_Message =>
                Error : Error_Type;
        end case;
    end record;


-- Interchange operations on the above types:

subtype Stream_Id is Transport_Stream.Stream_Id;

procedure Put (Into : Stream_Id; Data : Version_Range);
procedure Get (From : Stream_Id; Data : out Version_Range);

procedure Put_Message (Into : Stream_Id; Data : Message_Header);
function  Get_Message (From : Stream_Id) return Message_Header;

-- This last procedure is a little unusual.  If the gotten value
-- is OK, it returns it, otherwise it raises the corresponding
-- exception.  In a sense, an exception (or lack thereof) is a
-- possible return value.

Protocol_Error       : exception;
No_Such_Program      : exception;
No_Such_Version      : exception;
No_Such_Procedure    : exception;
Other_Error          : exception;
Invalid_Argument     : exception;
Server_Defined_Error : exception;

type Exception_Number is new Interchange.Integer;

procedure Put (Into : Stream_Id; Data : Exception_Number);
procedure Get (From : Stream_Id; Data : out Exception_Number);
```





```
    function Overlaps (X, Y : Version_Range) return Boolean;
    -- Return true if X and Y have some versions in common.

    function Max (X, Y : Version_Range) return Version_Number;
    -- Return the largest version which is common to both X & Y.

    Defined_Versions : constant Version_Range := (3, 5);
    -- RPC protocol versions which have been defined.

    Exception_Versions : constant Version_Range := (4, Version_Number'Last);
    -- RPC protocol versions which support server-defined exceptions.

    Username_Versions : constant Version_Range := (5, Version_Number'Last);
    -- RPC protocol versions which support passing Username & Password
    -- information with each call.

    Username_Or_Password_Error : exception;
    -- The Username & Password supplied with a remote procedure call
    -- were rejected by the server machine, either because there is
    -- no such Username or because the Password is incorrect.

end Rpc;
```

```
with Profile;
with Rpc;
with Transport_Defs;
with Transport_Stream;

package Rpc_Access_Utilities is

    pragma Consume_Offset (3);

    function Remote_Username
                (Host_Name : String;
                 Response  : Profile.Response_Profile := Profile.Get)
                return String;

    function Remote_Password
                (Host_Name : String;
                 Response  : Profile.Response_Profile := Profile.Get)
                return String;

    function Remote_Session (Host_Name : String;
                             Response : Profile.Response_Profile := Profile.Get)
                            return String;

    generic
        Default_Host_Name : String;
        Default_Socket    : Transport_Defs.Socket_Id;
        Default_Program   : Rpc.Program_Number;
        Default_Version   : Rpc.Version_Number;
        Default_Username  : String := "";
        Default_Password  : String := "";
    procedure Start_Request_Generic
                 (Stream    : out Transport_Stream.Stream_Id;
                  Proc      :     Rpc.Procedure_Number;
                  Host_Name :     String                    := Default_Host_Name;
                  Socket    :     Transport_Defs.Socket_Id := Default_Socket;
                  Program   :     Rpc.Program_Number        := Default_Program;
                  Version   :     Rpc.Version_Number        := Default_Version;
                  Username  :     String                    := Default_Username;
                  Password  :     String                    := Default_Password;
                  Response  :     Profile.Response_Profile := Profile.Get);
    -- Like Rpc_Client.Start_Request_with_Username, with the addition
    -- of host name resolution and Remote_Password file processing.
    -- The given Host_Name is resolved to a Network and Host_Id by
    -- Transport_Name.  If Username = "", then it is replaced by
    -- Remote_Username (Host_Name, Reponse).  If Password = "",
    -- then it is replaced by Remote_Password (Host_Name, Response).
end Rpc_Access_Utilities;
```





```
with Rpc;
with Transport_Defs;
with Transport_Stream;

package Rpc_Client is

    generic
        Default_Network : Transport_Defs.Network_Name;
        Default_Host    : Transport_Defs.Host_Id;
        Default_Socket  : Transport_Defs.Socket_Id;
        Default_Program : Rpc.Program_Number;
        Default_Version : Rpc.Version_Number;

    procedure Start_Request_Generic
                 (Stream  : out Transport_Stream.Stream_Id;
                  Proc    :     Rpc.Procedure_Number;
                  Network :     Transport_Defs.Network_Name := Default_Network;
                  Host    :     Transport_Defs.Host_Id      := Default_Host;
                  Socket  :     Transport_Defs.Socket_Id    := Default_Socket;
                  Program :     Rpc.Program_Number          := Default_Program;
                  Version :     Rpc.Version_Number          := Default_Version);

    -- Allocate a stream from the pool.  Transmit a call message
    -- header with the given program, version, and proc values.


    -- The following procedure is defunct: it is here for
    -- backward-compatibility.  Use the previous procedure.

    generic
        Pool    : in out Transport_Stream.Pool_Id;
        Program :        Rpc.Program_Number;
        Version :        Rpc.Version_Number;

    procedure Begin_Request_Generic (Stream : out Transport_Stream.Stream_Id;
                                     Proc   :     Rpc.Procedure_Number);

    -- Allocate a stream from the pool.  Transmit a call message
    -- header with the given program, version, and proc values.


    procedure End_Request (Stream : Transport_Stream.Stream_Id);

    -- Flush the transmit buffer.  Get the response header.
    -- If it is not OK, deallocate the stream.
```

```
    procedure End_Response (Stream : Transport_Stream.Stream_Id);

    -- Deallocate the stream.


    generic
        with procedure Raise_Exception (Excep : Rpc.Exception_Number);

    procedure End_Request_With_Exception (Stream : Transport_Stream.Stream_Id);

    -- Like End_Request (above), except that it also checks
    -- for a server-defined exception, and, if there is one,
    -- raises it (using the Raise_Exception procedure).


    generic
        Default_Network  : Transport_Defs.Network_Name;
        Default_Host     : Transport_Defs.Host_Id;
        Default_Socket   : Transport_Defs.Socket_Id;
        Default_Program  : Rpc.Program_Number;
        Default_Version  : Rpc.Version_Number;
        Default_Username : String := "";
        Default_Password : String := "";

    procedure Start_Request_With_Username
                 (Stream   : out Transport_Stream.Stream_Id;
                  Proc     :     Rpc.Procedure_Number;
                  Network  :     Transport_Defs.Network_Name := Default_Network;
                  Host     :     Transport_Defs.Host_Id := Default_Host;
                  Socket   :     Transport_Defs.Socket_Id := Default_Socket;
                  Program  :     Rpc.Program_Number := Default_Program;
                  Version  :     Rpc.Version_Number := Default_Version;
                  Username :     String := Default_Username;
                  Password :     String := Default_Password);

    -- Like Start_Request_Generic, above, with the addition of
    -- support for passing a username and password to the server.
    -- This is useful when the server must assume an identity in
    -- the access control system of the serving machine.
    -- This feature is supported only in versions 5 and higher
    -- of the RPC protocol.

end Rpc_Client;
```





```
package Rpc_Product is

    function Is_Installed return Boolean;

    Is_Not_Installed : exception;

end Rpc_Product;
```

```
with Rpc;
with Transport;
with Transport_Stream;

package Rpc_Server is

    procedure Begin_Response (Stream : Transport_Stream.Stream_Id;
                              Id     : Rpc.Transaction_Id);

    generic
        Program   : Rpc.Program_Number;
        Supported : Rpc.Version_Range := (0, Rpc.Version_Number'Last);

        with procedure Process_Call (Stream  : Transport_Stream.Stream_Id;
                                     Id      : Rpc.Transaction_Id;
                                     Version : Rpc.Version_Number;
                                     Proc    : Rpc.Procedure_Number) is <>;

        -- Process one procedure call: get the arguments from
        -- the STREAM, make the call, call BEGIN_RESPONSE
        -- (STREAM, ID), and put the results into the stream.
        -- If an exception is raised, call RETURN_EXCEPTION.
        -- If an unexpected exception is raised, simply let it
        -- propagate.  The caller will catch it.  The caller
        -- will also take care of flushing the stream transmit
        -- buffer on return.

    procedure Serve (Connection : Transport.Connection_Id);

    -- Serve an incoming RPC connection: Allocate a transport
    -- stream.  Check the incoming package and version.  If
    -- they don't match, transmit an exception.  If they
    -- match, process calls until the connection is
    -- disconnected.  On each call, catch any propagated
    -- exceptions and transmit them, and flush the transmit
    -- buffer.  When the connection is disconnected, or a
    -- protocol error occurs, deallocate the transport stream
    -- and return.

    procedure Return_Exception (Stream : Transport_Stream.Stream_Id;
                                Id     : Rpc.Transaction_Id;
                                Excep  : Rpc.Exception_Number);

    -- Like Begin_Response (above) except that it returns an exception.
    -- The server must NOT return any data following the exception.
```





```
    generic
        Program   : Rpc.Program_Number;
        Supported : Rpc.Version_Range := (0, Rpc.Version_Number'Last);

        with procedure Process_Call (Stream   : Transport_Stream.Stream_Id;
                                     Id       : Rpc.Transaction_Id;
                                     Version  : Rpc.Version_Number;
                                     Proc     : Rpc.Procedure_Number;
                                     Username : String;
                                     Password : String) is <>;

        -- Process one procedure call: assume the identity
        -- of the given Username, get the arguments from
        -- the STREAM, make the call, call BEGIN_RESPONSE
        -- (STREAM, ID), and put the results into the stream.
        -- If an exception is raised, call RETURN_EXCEPTION.
        -- If an unexpected exception is raised, simply let it
        -- propagate.  The caller will catch it.  The caller
        -- will also take care of flushing the stream transmit
        -- buffer on return.

    procedure Serve_With_Username (Connection : Transport.Connection_Id);

    -- Serve an incoming RPC connection: Allocate a transport
    -- stream.  Check the incoming package and version.  If
    -- they don't match, transmit an exception.  If they
    -- match, process calls until the connection is
    -- disconnected.  On each call, catch any propagated
    -- exceptions and transmit them, and flush the transmit
    -- buffer.  When the connection is disconnected, or a
    -- protocol error occurs, deallocate the transport stream
    -- and return.

end Rpc_Server;
```

```
procedure Show_Trace (Entries : Natural := 8);

pragma Loaded_Main;
```





```
procedure Tcp_Ip_Boot
              (Use_Arp            : Boolean := True;
               Enable_Link_Level  : Boolean := True;
               Exos_Prefix        : String  := "!tools.networking.";
               Host_Id_File       : String  := "!machine.tcp_ip_host_id";
               Ether_Id_File      : String  := "!machine.ethernet_host_id";
               Use_Checksums      : Boolean := True;
               Diagnostic         : Boolean := False);

-- Download the Ethernet controller with TCP/IP networking software,
-- and start it.  The controller may be an Excelan EXOS-204 with
-- EXOS 8010 software, or a CMC ENP-100i with CMC TCP/IP software.

-- Software for the Excelan EXOS-204 controller is stored in files
-- named EXOS_PREFIX & "EXOS_p_x_y"; where p is the Excelan product
-- number, and x and y are the software version number.  For example,
-- the file "EXOS_8010_3_2" contains the 8010 software, version 3.2.
-- Software for the CMC ENP-100i controller is stored in files named
-- EXOS_PREFIX & "CMC_TCP_IP_x_y_z"; where x, y and z are the software
-- version number.  For example, the file "CMC_TCP_IP_2_6_1" contains
-- software version 2.6.1.

-- Tcp_Ip_Boot loads the most recent version of the software that is
-- present in the EXOS_PREFIX directory.  So, if both versions 3.2 and
-- 3.1 are present, 3.2 will be loaded.

-- HOST_ID_FILE is the name of a text file which begins with the
-- Internet address of this machine, in decimal dotted notation.  This
-- value is used to initialize the TCP/IP software, so that it will
-- respond to ARP queries and IP datagrams directed to this address.

-- The file named "TCP_IP_Subnet_Mask", if it exists in the same
-- directory as HOST_ID_FILE, begins with the subnet mask, in decimal
-- dotted notation.  Each non-zero bit of the IP subnet mask indicates
-- that the corresponding bit of this machine's IP address is part of
-- the network or subnetwork number.

-- ETHER_ID_FILE is the name of a text file which, if it exists,
-- begins with the Ethernet address of this machine, in decimal dotted
-- notation.  If this file does not exist or is illegible, the address
-- will be taken from PROM on the Ethernet controller.  Ordinarily
-- this file does not exist; the use of the PROM value is recommended.

-- The HOST_ID_FILE, TCP_IP_Subnet_Mask and/or ETHER_ID_FILE may
-- optionally contain a Machine.Id, written as a decimal number, after
-- the address (and some blank space).  If the Machine.Id is present,
-- but does not match Machine.Get_Id, the file contents will not be
```

```
-- used.  In the case of HOST_ID_FILE, Tcp_Ip_Boot fails, and the
-- controller is not started.  In the case of ETHER_ID_FILE, the
-- address in PROM on the controller will be used.

-- The HOST_ID_FILE, TCP_IP_Subnet_Mask and/or ETHER_ID_FILE may
-- optionally contain comments, at the end of a line, marked by --.

-- Decimal dotted notation means the form nn.nn.nn.nn; where each nn
-- is the (decimal) representation of one byte of the address.  The
-- most significant byte comes first.  For example, network number 89
-- is commonly used for private IP networks: such addresses will have
-- the form "89.nn.nn.nn" in decimal dotted notation.

-- USE_ARP determines whether the TCP/IP software will use ARP
-- (the Address Resolution Protocol) to find the Ethernet addresses
-- of other hosts it wants to talk to, and to advertise its own
-- Ethernet address to other hosts that want to connect to it.

-- ENABLE_LINK_LEVEL, if true, enables the use of Ethernet link level
-- I/O.  This form of I/O allows application programs to transmit
-- arbitrary frames on the Ethernet, and to receive frames that do
-- not contain TCP/IP data.

-- USE_CHECKSUMS, if true, enables the calculation and checking
-- of IP header checksums and TCP checksums.

-- DIAGNOSTIC, if true, causes the procedure to do a dry run; that is,
-- scan the code file, calculate memory allocation and initial data
-- settings, and print out information about the results, all without
-- actually affecting the Ethernet controller.  This is intended for
-- debugging Tcp_Ip_Boot.
```





```
procedure Tcp_Ip_Dump (Dump_File : String := "");



package Telnet_Product is

    function Is_Installed return Boolean;

    Is_Not_Installed : exception;

end Telnet_Product;
```

```
package Telnet_Profile is

    -- This package supplies default values for Telnet commands.
    -- You can set your defaults by editing your session switch
    -- file.  In the switch file, a switch named Telnet.xxx sets
    -- the default values returned by function xxx below.

    function Escape return String;
    -- If Escape is non-null, and the Escape string is received
    -- from the terminal, then the session will be suspended
    -- and the terminal will be reconnected to the Environment.

    function Escape_On_Break return Boolean;
    -- If Escape_On_Break is true, and a BREAK signal is received
    -- from the terminal, then the session will be suspended
    -- and the terminal will be reconnected to the Environment.

    function Remote_Machine return String;
    -- The name of the machine you want to communicate with.

end Telnet_Profile;
```





```
with System_Utilities;
with Telnet_Protocol;
with Terminal_Specific;

package Telnet_Port is

    function Name (Port : Port_Type) return String
        renames System_Utilities.Terminal_Name;
    -- A port may be read or written by opening a File_Type object on
    -- it, with Open.Name => Name (Port); and using the Read and Write
    -- operations defined on File_Type (see package Terminal_Specific).

    function Port (File : File_Type) return Port_Type;
    -- Return the port on which the given File was opened.
    -- If File was not opened on a port, raise Not_A_Telnet_Port.

    function Is_A_Telnet_Port (Port : Port_Type) return Boolean;
    function Is_A_Telnet_Port (File : File_Type) return Boolean;
    -- Return true iff the given object identifies a Telnet port.
    -- Is_A_Telnet_Port (File) = Is_A_Telnet_Port (Port (File)).

    Not_A_Telnet_Port : exception;
    -- Raised if any of the operations below are attempted on
    -- a File that was not opened on a Telnet port.

    -- Although the operations below take File_Type parameters, they
    -- actually manipulate underlying permanent Port data structures.
    -- One may open and close File_Type handles repeatedly on a given
    -- port, without changing the state associated with that port.
    -- In particular, closing a File does NOT disconnect the port, nor
    -- reset its Convert_Received_New_Line_to_CR switch to the default.
    -- design note: File_Type is used instead of Port_Type to force
    -- the client of this package to obtain a lock on the port before
    -- manipulating its state; if the lock cannot be obtained then the
    -- state cannot be manipulated.  This is a feature: it allows a job
    -- (for example the login manager or core editor) to prevent other
    -- jobs from messing with the state of its port.

    -- For each Telnet port there is a Boolean that, if True, causes each
    -- received CRLF (Telnet new_line) sequence to be read as CR alone.
    -- The purpose of this switch is to support Telnet terminal servers
    -- that transmit the single keystroke [return] as a Telnet new_line.
    -- At system boot time, every port's switch is set to True.  It may
    -- be changed at any time.
```

```
    procedure Set_Convert_Received_New_Line_To_Cr
                 (File : File_Type; Enabled : Boolean := True);
    -- Set the switch to the given value.

    function Get_Convert_Received_New_Line_To_Cr
                (File : File_Type) return Boolean;
    -- Return the present value of the switch.

    procedure Connect (File       : File_Type;
                       Connection : Telnet_Protocol.Connection_Id);
    -- If Is_Connected (File) then Disconnect (File);
    -- Bind the Telnet port identified by File to the given Connection.

    procedure Connect (File     : File_Type;
                       Max_Wait : Duration := Duration'Last);
    -- If Is_Connected (File) then do nothing and return immediately.
    -- Otherwise, wait for an incoming connection on TCP/IP socket 23.
    -- If it arrives within Max_Wait, open a Telnet connection, and
    -- bind it to the Telnet port identified by the given File.
    -- Otherwise, raise Not_Connected.

    function Is_Connected (File : File_Type) return Boolean;
    -- Return true iff the given File identifies a Telnet port that
    -- is bound to a Telnet_Protocol.Connection_Id that is connected.

    -- If a Telnet port is connected, then data written to it will
    -- be transmitted on the associated Telnet connection, and
    -- data received on that connection may be read from the port.

    -- If a Telnet port is not connected, then data written to it
    -- will be discarded (ignored), and any attempt to read from it
    -- will first execute Connect (File, Max_Wait).

    procedure Disconnect (File : File_Type);
    -- if Is_Connected (File) then Telnet_Protocol.Close (Connection (File)).

    function Connection (File : File_Type) return Telnet_Protocol.Connection_Id;
    -- Return the Connection that is currently bound to the Telnet port
    -- that is identified by File.  If there is none, raise Not_Connected.

    Not_Connected : exception;

end Telnet_Port;
```





```
with Byte_Defs;
with Transport;
with Transport_Defs;

package Telnet_Protocol is

    type Connection_Id is private;

    pragma Consume_Offset (5);

    procedure Open (Connection : out Connection_Id;
                    Remote     :     Transport.Connection_Id);
    -- Allocate Telnet resources (including space to store
    -- option values), and bind them to the given Remote.
    -- Initially, all options for both owners are disabled.
    -- Does not affect the Transport connection.

    procedure Close (Connection : Connection_Id);
    -- Release Telnet resources.
    -- Closes the Transport connection.

    function Remote (Connection : Connection_Id) return Transport.Connection_Id;
    -- Return the ID of the associated Transport connection.

    function Image (Connection : Connection_Id) return String;
    -- Return a unique name for a connection.


    ---- OPTION NEGOTIATION ----
    type Option is (Transmit_Binary, Echo, Suppress_Go_Ahead);

    type Owner is (Local, Remote);

    type    Desired_Option_States is (On, Off, Dont_Care);
    subtype Current_Option_States is Desired_Option_States range On .. Off;

    procedure Offer (Connection : Connection_Id;
                     For_Option : Option;
                     For_Owner  : Owner;
                     To_State   : Desired_Option_States := Off);
    -- Initiate negotiation of an option value.
    -- To_State indicates the desired option value.
```

```
    procedure Request (Connection :     Connection_Id;
                       For_Option :     Option;
                       For_Owner  :     Owner;
                       To_State   :     Desired_Option_States := Off;
                       Success    : out Boolean;
                       Max_Wait   :     Duration              := Duration'Last);
    -- Initiate negotiation of an option value.
    -- To_State indicates the desired option value.
    -- Wait until the remote Telnet implementation either
    -- accepts or rejects the option, or Max_Wait expires.
    -- If the option is accepted, return Success = True,
    -- otherwise return Success = False.

    function State_Of (Connection : Connection_Id;
                       For_Option : Option;
                       For_Owner  : Owner) return Current_Option_States;
    -- Return the current state of the option.


    ---- DATA I/O ----

    type Signal_Type is (None, New_Line, Synch, Break,
                         Interrupt_Process, Abort_Output, Are_You_There,
                         Erase_Character, Erase_Line, Go_Ahead);

    procedure Transmit (Connection :     Connection_Id;
                        Status     : out Transport_Defs.Status_Code;
                        Data       :     Byte_Defs.Byte_String;
                        Count      : out Natural;
                        Signal     :     Telnet_Protocol.Signal_Type := None;
                        Max_Wait   :     Duration := Duration'Last);

    procedure Transmit (Connection :     Connection_Id;
                        Status     : out Transport_Defs.Status_Code;
                        Data       :     String;
                        Count      : out Natural;
                        Signal     :     Telnet_Protocol.Signal_Type := None;
                        Max_Wait   :     Duration := Duration'Last);

    procedure Transmit (Connection :     Connection_Id;
                        Status     : out Transport_Defs.Status_Code;
                        Signal     :     Telnet_Protocol.Signal_Type;
                        Max_Wait   :     Duration := Duration'Last);
    -- Transmit the given Data, followed by the given Signal.
    -- Stuff NUL's after CR's, and double 255's.
    -- Shave off high-order bits if not transmitting binary.
```





```
    procedure Receive (Connection :     Connection_Id;
                       Status     : out Transport_Defs.Status_Code;
                       Data       : out Byte_Defs.Byte_String;
                       Count      : out Natural;
                       Signal     : out Telnet_Protocol.Signal_Type;
                       Max_Wait   :     Duration := Duration'Last);

    procedure Receive (Connection :     Connection_Id;
                       Status     : out Transport_Defs.Status_Code;
                       Data       : out String;
                       Count      : out Natural;
                       Signal     : out Telnet_Protocol.Signal_Type;
                       Max_Wait   :     Duration := Duration'Last);
    -- Receive some information.  Stop when either
    -- (1) a non-OK Status is received, or
    -- (2) a non-None Signal is received, or
    -- (3) some Data are received, or
    -- (4) Max_Wait expires.
    -- Return the number of Data bytes received in Count.
    -- Strip NUL's after CR's, and un-double 255's.
    -- Echo data if the Echo option is enabled.

    -- Receive will not return both data (Count > 0)
    -- and a signal (Signal /= None) in the same call.

    -- NOTE: only one Receive can be made at a time and
    --       that at one Receive must be pending for forward
    --       progress.


    ---- EXCEPTIONS ----
    Not_Connected      : exception;
    Receive_Data_Error : exception;

    pragma Consume_Offset (4);

    procedure Caller_Will_Take_Responsibility_For_Echo
                (Connection                      : Connection_Id;
                 Caller_Will_Take_Responsibility : Boolean := True);

end Telnet_Protocol;
```

```
with Telnet_Port;
with Telnet_Protocol;
with Transport_Defs;

package Telnet_Tools is

    -- Some commonly-used operations derived from Telnet_Port, etc.

    Telnet_Network : constant Transport_Defs.Network_Name :=
       Transport_Defs.Normalize ("TCP/IP");
    Telnet_Socket  : constant Transport_Defs.Socket_Id    := (0, 23);

    procedure Connect
                  (Connection    : out Telnet_Protocol.Connection_Id;
                   Status        : out Transport_Defs.Status_Code;
                   Remote_Host   : Transport_Defs.Host_Id;
                   Network       : Transport_Defs.Network_Name := Telnet_Network;
                   Remote_Socket : Transport_Defs.Socket_Id := Telnet_Socket;
                   Max_Wait      : Duration := Duration'Last;
                   Max_Retries   : Natural := 2);
    -- Form a Telnet connection to the specified Remote_Host:
    -- Transport.Open (X, Status, Network);
    -- Transport.Connect (X, Status, Remote_Host, Remote_Socket, Max_Wait);
    -- if Status = Connection_Refused then retry, up to Max_Retries times;
    -- Telnet_Protocol.Open (Connection, X).

    procedure Connect
                  (Connection       : out Telnet_Protocol.Connection_Id;
                   Status           : out Transport_Defs.Status_Code;
                   Remote_Host_Name : String;
                   Remote_Socket    : Transport_Defs.Socket_Id := Telnet_Socket;
                   Max_Wait         : Duration := Duration'Last;
                   Max_Retries      : Natural := 2);
    -- Form a Telnet connection to the specified Remote_Host:
    -- Resolve Remote_Host_Name to a Host_Id and Network_Name,
    -- using package Transport_Name;
    -- Connect (Connection, etc..);

    procedure Connect
                  (File          : Telnet_Port.File_Type;
                   Status        : out Transport_Defs.Status_Code;
                   Remote_Host   : Transport_Defs.Host_Id;
                   Network       : Transport_Defs.Network_Name := Telnet_Network;
                   Remote_Socket : Transport_Defs.Socket_Id := Telnet_Socket;
                   Max_Wait      : Duration := Duration'Last;
                   Max_Retries   : Natural := 2);
```





```
    procedure Connect
                  (File             : Telnet_Port.File_Type;
                   Status           : out Transport_Defs.Status_Code;
                   Remote_Host_Name : String;
                   Remote_Socket    : Transport_Defs.Socket_Id := Telnet_Socket;
                   Max_Wait         : Duration := Duration'Last;
                   Max_Retries      : Natural := 2);
    -- Form a Telnet connection to the specified Remote_Host:
    -- Connect (Telnet_Protocol.Connection_Id'(X), etc...);
    -- Telnet_Port.Connect (File, X);

end Telnet_Tools;
```

```
with Ftp_Defs;
with Ftp_Name_Map;
with Ftp_Profile;
with Profile;

generic
    with function Local_To_Remote
                    (File_Name   : String;
                     Local_Roof  : String;
                     Remote_Roof : String;
                     Remote_Type : Ftp_Name_Map.Machine_Type) return String;

    -- Return the remote file name corresponding
    -- to the given local File_Name.

    with function Remote_To_Local
                    (File_Name   : String;
                     Local_Roof  : String;
                     Remote_Roof : String;
                     Remote_Type : Ftp_Name_Map.Machine_Type) return String;

    -- Return the local file name corresponding
    -- to the given remote File_Name.

package Transfer_Generic is

    -- This package helps you to create 'customized' versions of
    -- commands in package Ftp.  Specifically, you can replace
    -- the standard Ftp name mapping functions (from Ftp_Name_Map)
    -- with your own, customized name mapping functions.

    -- The multi-file Ftp operations (Put_Set, Get_Set, Get_List,
    -- Store_Set, Retrieve_Set, Retrieve_List) use name mapping
    -- functions for converting between remote and local filenames.
    -- The single-file operations (Put, Get, Store, Retrieve) also
    -- use these mappings if you omit the target file name.  These
    -- name mapping functions are defined in package Ftp_Name_Map.

    -- If the standard name mapping functions don't do what you
    -- need, you can supply your own (as the parameters to this
    -- generic) and use the resulting transfer operations instead
    -- of the ones in package Ftp.

    procedure Put
                (From_Local_File    : String := "<IMAGE>";
                 To_Remote_File     : String :=
                    "";       -- default => map local_name
```





```
                 Remote_Machine     : String := Ftp_Profile.Remote_Machine;
                 Username           : String := Ftp_Profile.Username;
                 Password           : String := Ftp_Profile.Password;
                 Account            : String := Ftp_Profile.Account;
                 Remote_Directory   : String := Ftp_Profile.Remote_Directory;
                 Remote_Type        : Ftp_Name_Map.Machine_Type :=
                    Ftp_Profile.Remote_Type;
                 Append_To_File     : Boolean := False;
                 Transfer_Type      : Ftp_Defs.Type_Code :=
                    Ftp_Profile.Transfer_Type;
                 Transfer_Mode      : Ftp_Defs.Mode_Code :=
                    Ftp_Profile.Transfer_Mode;
                 Transfer_Structure : Ftp_Defs.Structure_Code :=
                    Ftp_Profile.Transfer_Structure;
                 Send_Port          : Boolean := Ftp_Profile.Send_Port_Enabled;
                 Response           : Profile.Response_Profile := Profile.Get);

    procedure Get
                (From_Remote_File   : String := "";
                 To_Local_File      : String := "";
                 Remote_Machine     : String := Ftp_Profile.Remote_Machine;
                 Username           : String := Ftp_Profile.Username;
                 Password           : String := Ftp_Profile.Password;
                 Account            : String := Ftp_Profile.Account;
                 Remote_Directory   : String := Ftp_Profile.Remote_Directory;
                 Remote_Type        : Ftp_Name_Map.Machine_Type :=
                    Ftp_Profile.Remote_Type;
                 Append_To_File     : Boolean := False;
                 Transfer_Type      : Ftp_Defs.Type_Code :=
                    Ftp_Profile.Transfer_Type;
                 Transfer_Mode      : Ftp_Defs.Mode_Code :=
                    Ftp_Profile.Transfer_Mode;
                 Transfer_Structure : Ftp_Defs.Structure_Code :=
                    Ftp_Profile.Transfer_Structure;
                 Send_Port          : Boolean := Ftp_Profile.Send_Port_Enabled;
                 Response           : Profile.Response_Profile :=
                    Ftp_Profile.Profile_Get);

    procedure Store (From_Local_File : String := "<IMAGE>";
                     To_Remote_File  : String := "";
                     Remote_Type     : Ftp_Name_Map.Machine_Type;
                     Append_To_File  : Boolean := False;
                     Response        : Profile.Response_Profile := Profile.Get;
                     Account         : String := Ftp_Profile.Account);

    procedure Retrieve
                (From_Remote_File : String := "";
                 To_Local_File    : String := "";
                 Remote_Type      : Ftp_Name_Map.Machine_Type;
```

```
              Append_To_File    : Boolean := False;
              Response          : Profile.Response_Profile := Profile.Get;
              Account           : String := Ftp_Profile.Account);

   procedure Put_Set
             (From_Local_File_Set : String := "<IMAGE>";
              Local_Roof : String := "$";
              Remote_Roof : String := Ftp_Profile.Remote_Roof;
              Remote_Machine : String := Ftp_Profile.Remote_Machine;
              Username : String := Ftp_Profile.Username;
              Password : String := Ftp_Profile.Password;
              Account : String := Ftp_Profile.Account;
              Remote_Directory : String := Ftp_Profile.Remote_Directory;
              Remote_Type : Ftp_Name_Map.Machine_Type :=
                 Ftp_Profile.Remote_Type;
              Append_To_File : Boolean := False;
              Transfer_Type : Ftp_Defs.Type_Code :=
                 Ftp_Profile.Transfer_Type;
              Transfer_Mode : Ftp_Defs.Mode_Code :=
                 Ftp_Profile.Transfer_Mode;
              Transfer_Structure : Ftp_Defs.Structure_Code :=
                 Ftp_Profile.Transfer_Structure;
              Send_Port : Boolean := Ftp_Profile.Send_Port_Enabled;
              Response : Profile.Response_Profile := Profile.Get);

   procedure Get_Set
             (From_Remote_File_Set : String := "";
              Local_Roof : String := "$";
              Remote_Roof : String := Ftp_Profile.Remote_Roof;
              Remote_Machine : String := Ftp_Profile.Remote_Machine;
              Username : String := Ftp_Profile.Username;
              Password : String := Ftp_Profile.Password;
              Account : String := Ftp_Profile.Account;
              Remote_Directory : String := Ftp_Profile.Remote_Directory;
              Remote_Type : Ftp_Name_Map.Machine_Type :=
                 Ftp_Profile.Remote_Type;
              Append_To_File : Boolean := False;
              Transfer_Type : Ftp_Defs.Type_Code :=
                 Ftp_Profile.Transfer_Type;
              Transfer_Mode : Ftp_Defs.Mode_Code :=
                 Ftp_Profile.Transfer_Mode;
              Transfer_Structure : Ftp_Defs.Structure_Code :=
                 Ftp_Profile.Transfer_Structure;
              Send_Port : Boolean := Ftp_Profile.Send_Port_Enabled;
              Response : Profile.Response_Profile := Profile.Get);

   procedure Get_List
             (Remote_File_List   : String := "";
              Local_Roof         : String := "$";
```





```
              Remote_Roof        : String := Ftp_Profile.Remote_Roof;
              Remote_Machine     : String := Ftp_Profile.Remote_Machine;
              Username           : String := Ftp_Profile.Username;
              Password           : String := Ftp_Profile.Password;
              Account            : String := Ftp_Profile.Account;
              Remote_Directory   : String := Ftp_Profile.Remote_Directory;
              Remote_Type        : Ftp_Name_Map.Machine_Type :=
                 Ftp_Profile.Remote_Type;
              Append_To_File     : Boolean := False;
              Transfer_Type      : Ftp_Defs.Type_Code :=
                 Ftp_Profile.Transfer_Type;
              Transfer_Mode      : Ftp_Defs.Mode_Code :=
                 Ftp_Profile.Transfer_Mode;
              Transfer_Structure : Ftp_Defs.Structure_Code :=
                 Ftp_Profile.Transfer_Structure;
              Send_Port          : Boolean := Ftp_Profile.Send_Port_Enabled;
              Response           : Profile.Response_Profile := Profile.Get);

   procedure Store_Set
             (From_Local_File_Set : String := "<IMAGE>";
              Local_Roof : String := "$";
              Remote_Roof : String := Ftp_Profile.Current_Remote_Roof;
              Remote_Type : Ftp_Name_Map.Machine_Type :=
                 Ftp_Profile.Current_Remote_Type;
              Append_To_File : Boolean := False;
              Response : Profile.Response_Profile := Profile.Get;
              Account : String := Ftp_Profile.Account);

   procedure Retrieve_Set (From_Remote_File_Set : String := "";
                           Local_Roof : String := "$";
                           Remote_Roof : String :=
                              Ftp_Profile.Current_Remote_Roof;
                           Remote_Type : Ftp_Name_Map.Machine_Type :=
                              Ftp_Profile.Current_Remote_Type;
                           Append_To_File : Boolean := False;
                           Response : Profile.Response_Profile := Profile.Get;
                           Account : String :=
                              Ftp_Profile.Account);

   procedure Retrieve_List
             (Remote_File_List : String := "";
              Local_Roof       : String := "$";
              Remote_Roof      : String := Ftp_Profile.Current_Remote_Roof;
              Remote_Type      : Ftp_Name_Map.Machine_Type :=
                 Ftp_Profile.Current_Remote_Type;
              Append_To_File   : Boolean := False;
              Response         : Profile.Response_Profile := Profile.Get;
              Account          : String := Ftp_Profile.Account);
end Transfer_Generic;
```

```
with Byte_Defs;
with Machine;
with Transport_Defs;

package Transport is

    type Connection_Id is private;
    -- Identifies the local resources associated with a connection.

    Null_Connection_Id : constant Connection_Id;

    function Local_Host (Network : Transport_Defs.Network_Name)
                    return Transport_Defs.Host_Id;
    -- The ID which identifies your machine in the given NETWORK.


    -- All defined network_names can be scanned:

    type Network_Name_Iterator is limited private;
    procedure Init  (Iter : in out Network_Name_Iterator);
    procedure Next  (Iter : in out Network_Name_Iterator);
    function  Done  (Iter : Network_Name_Iterator) return Boolean;
    function  Value (Iter : Network_Name_Iterator)
                return Transport_Defs.Network_Name;


    -- All open connection_id's can be scanned:

    type Connection_Id_Iterator is limited private;
    procedure Init  (Iter : in out Connection_Id_Iterator);
    procedure Next  (Iter : in out Connection_Id_Iterator);
    function  Done  (Iter : Connection_Id_Iterator) return Boolean;
    function  Value (Iter : Connection_Id_Iterator) return Connection_Id;


    procedure Open (Connection   : out Transport.Connection_Id;
                    Status       : out Transport_Defs.Status_Code;
                    Network      :     Transport_Defs.Network_Name;
                    Local_Socket :     Transport_Defs.Socket_Id :=
                       Transport_Defs.Null_Socket_Id);
    -- Allocate the local resources required to establish a
    -- connection.  Any subsequent connection which may be
    -- established will be via the given NETWORK, using the given
    -- LOCAL_SOCKET.  If LOCAL_SOCKET = NULL_SOCKET_ID, the
    -- transport service will invent a socket_id which is not
    -- currently in use, and assign it to this connection.
```





```
    procedure Close (Connection : Transport.Connection_Id);
    -- Deallocate local resources.
    -- If the connection is connected, disconnect it.
    -- If the connection is not open, do nothing.

    procedure Connect (Connection    :     Transport.Connection_Id;
                       Status        : out Transport_Defs.Status_Code;
                       Remote_Host   :     Transport_Defs.Host_Id;
                       Remote_Socket :     Transport_Defs.Socket_Id;
                       Max_Wait      :     Duration := Duration'Last);
    -- Initiate a connection to the specified remote host and socket.
    -- The CONNECTION must be open.

    procedure Connect (Connection :     Transport.Connection_Id;
                       Status     : out Transport_Defs.Status_Code;
                       Max_Wait   :     Duration := Duration'Last);
    -- Wait for a connection initiated by some other task.
    -- The CONNECTION must be open.

    procedure Disconnect (Connection : Transport.Connection_Id);
    -- Break a connection.
    -- If the CONNECTION is not connected, do nothing.

    function Is_Open      (Connection : Transport.Connection_Id) return Boolean;
    function Is_Connected (Connection : Transport.Connection_Id) return Boolean;
    --  function is_connecting_passive
    --                        (connection : transport.connection_id)
    --                          return Boolean;
    --  function is_connecting_active
    --                        (connection : transport.connection_id)
    --                          return Boolean;
    function Network       (Connection : Transport.Connection_Id)
                  return Transport_Defs.Network_Name;
    function Local_Host    (Connection : Transport.Connection_Id)
                     return Transport_Defs.Host_Id;
    function Local_Socket  (Connection : Transport.Connection_Id)
                       return Transport_Defs.Socket_Id;
    function Remote_Host   (Connection : Transport.Connection_Id)
                      return Transport_Defs.Host_Id;
    function Remote_Socket (Connection : Transport.Connection_Id)
                        return Transport_Defs.Socket_Id;


    procedure Transmit (Connection :     Transport.Connection_Id;
                        Status     : out Transport_Defs.Status_Code;
                        Data       :     Byte_Defs.Byte_String;
                        Count      : out Natural;
                        Max_Wait   :     Duration := Duration'Last;
                        More       :     Boolean  := False);
```

```
-- Transmit some data.  The CONNECTION must be connected.
-- STATUS is returned OK, unless the connection is broken.
-- DATA is the data to be transmitted.  COUNT is returned the
-- number of bytes actually transmitted.  This may differ
-- from DATA'LENGTH if the connection breaks, or if the
-- operation times out.

-- MAX_WAIT is the maximum amount of time to spend trying to
-- transmit the data.  The operation completes when all the
-- DATA bytes have been transmitted, or when MAX_WAIT
-- expires, whichever comes first.

-- MORE indicates that the service may hold the data in its
-- local buffers, to be combined with more data which the
-- client is about to transmit.  This is a performance hint
-- only: the service is free to ignore it and transmit all
-- data as soon as it gets it.

-- This operation may simply store data in a local buffer,
-- and actually transmit it at some future opportunity.  If
-- buffering is involved, the service assumes responsibility
-- for transmitting the buffered data.


procedure Receive (Connection :     Transport.Connection_Id;
                   Status     : out Transport_Defs.Status_Code;
                   Data       : out Byte_Defs.Byte_String;
                   Count      : out Natural;
                   Max_Wait   :     Duration := Duration'Last);

-- Receive some data.  The CONNECTION must be connected.
-- STATUS is returned OK, unless the connection is broken.
-- DATA is the buffer into which to store received data.
-- COUNT is returned the number of bytes actually stored.
-- This may be less than DATA'LENGTH.

-- MAX_WAIT is the maximum amount of time to spend waiting
-- to receive some data.  The operation completes when one
-- or more DATA bytes have been received, or when MAX_WAIT
-- expires, whichever comes first.

-- This procedure does not wait to fill up the DATA buffer.
-- As soon as ANY data are received, it returns them.

function Hash (Connection : Transport.Connection_Id) return Natural;
-- Calculate a value suitable for hashing.
```





```
-- Each connection is owned by some job.
-- By default, a connection is owned by the job which opens it.
-- Whenever a job terminates, all connections owned by it are
-- automatically closed.

procedure Set_Owner (Connection : Transport.Connection_Id;
                     Owner      : Machine.Job_Id);
-- Set the owner of the connection.  This procedure allows
-- a connection to be given away to some other job.

function Get_Owner (Connection : Transport.Connection_Id)
                   return Machine.Job_Id;
-- Return the owner of the connection.

procedure Close_All (Owner : Machine.Job_Id);
-- Close all connections owned by the given Owner.
-- This happens automatically when the Owner terminates.

function Is_Connecting_Passive
             (Connection : Transport.Connection_Id) return Boolean;
function Is_Connecting_Active
             (Connection : Transport.Connection_Id) return Boolean;

pragma Consume_Offset (4);


-- The following 3 operations were invented to support AX25.
-- Set_ and Get_Options allow the user to select values of
-- various protocol parameters (for example, packet size,
-- window size, timeouts).  Get_Statistics allows the user
-- to monitor the operation of the protocol.

-- For AX25, the Context string is the 'image (ASCII decimal)
-- of the port number of a port used for AX25 communication.

function Set_Options (Network : Transport_Defs.Network_Name;
                      Context : String := "";
                      Options : String := "") -- new option values
                     return String;
-- The Options string is a sequence of option name/value
-- pairs, using the option parser syntax "<name>=<value>,...".
-- The return value is "" if the operation succeeded.
-- A non-null value is an error message, which may contain
-- ASCII.LF's to indicate line breaks.
```

```
    function Get_Options (Network : Transport_Defs.Network_Name;
                          Context : String := "")
                         return String; -- all option values
    -- The return value is a list of all current options
    -- and their current values, in the same syntax as
    -- Set_Options.Options.  A null value indicates that
    -- the Network/Context pair is not defined or has no
    -- options.

    function Get_Statistics
                (Network : Transport_Defs.Network_Name; Context : String := "")
               return String;
    -- The return value is a list of all current statistics and
    -- their values, in a syntax similar to Set_Options.Options.
    -- A null value indicates that the Network/Context pair is
    -- not defined or has no statistics.

    package Route is
        -- A table of 4-tuples, used for Transport level routing.
        -- See package Transport_Route for background information.

        procedure Define (Network     :     Transport_Defs.Network_Name;
                          Destination :     Transport_Defs.Host_Id;
                          Subnet_Mask :     Transport_Defs.Host_Id;
                          Route       :     Transport_Defs.Host_Id;
                          Status      : out Transport_Defs.Status_Code);

        procedure Undefine (Network     :     Transport_Defs.Network_Name;
                            Destination :     Transport_Defs.Host_Id;
                            Subnet_Mask :     Transport_Defs.Host_Id;
                            Route       :     Transport_Defs.Host_Id;
                            Status      : out Transport_Defs.Status_Code);

        type Iterator is private;
        procedure Init (Iter : out Iterator);
        procedure Next (Iter : in out Iterator);
        function  Done (Iter : Iterator) return Boolean;
        function  Network (Iter : Iterator) return Transport_Defs.Network_Name;
        function  Destination (Iter : Iterator) return Transport_Defs.Host_Id;
        function  Subnet_Mask (Iter : Iterator) return Transport_Defs.Host_Id;
        function  Route (Iter : Iterator) return Transport_Defs.Host_Id;

    end Route;

end Transport;
```





```
with Byte_Defs;

package Transport_Defs is

    type Network_Name is new String;
    -- Identifies one of the several networks to which the system
    -- may be connected, e.g.  "Milnet", "Telenet", "Bitnet", etc.

    Null_Network_Name : constant Network_Name := "";

    function Normalize (Value : Network_Name) return Network_Name;
    -- Strips leading/trailing blanks, and maps to upper case.

    function Hash (Value : Network_Name) return Natural;

    type Host_Id is new Byte_Defs.Byte_String;
    -- Identifies a machine within a network.

    Null_Host_Id : constant Host_Id (1 .. 0) := (others => 0);

    function Normalize (Value : Host_Id) return Host_Id;
    -- Strips leading 0's.

    function Hash (Value : Host_Id) return Natural;

    type Socket_Id is new Byte_Defs.Byte_String;
    -- Identifies a program within a machine.

    Null_Socket_Id : constant Socket_Id (1 .. 0) := (others => 0);

    function Normalize (Value : Socket_Id) return Socket_Id;
    -- Strips leading 0's.

    function Hash (Value : Socket_Id) return Natural;

    type Status_Code is new Integer;
    -- An encoding of the outcome of a TRANSPORT operation.

    function Image (Status : Status_Code) return String;
    -- Returns a printable string, describing the error.

    Ok                     : constant Status_Code := 0;
    No_Local_Resources     : constant Status_Code := 1;
    No_Free_Sockets        : constant Status_Code := 2;
    No_Free_Memory         : constant Status_Code := 3;
    Not_Open               : constant Status_Code := 4;
    Not_Connected          : constant Status_Code := 5;
```

```
   Too_Many_Clients        : constant Status_Code := 6;
   Timed_Out               : constant Status_Code := 7;
   No_Such_Host            : constant Status_Code := 8;
   Connection_Refused      : constant Status_Code := 9;
   Disconnected            : constant Status_Code := 10;
   Connection_Broken       : constant Status_Code := 11;
   No_Hardware             : constant Status_Code := 12;
   No_Such_Network         : constant Status_Code := 13;
   Not_Initialized         : constant Status_Code := 14;
   Not_Downloaded          : constant Status_Code := 15;
   Socket_In_Use           : constant Status_Code := 16;
   Access_Denied           : constant Status_Code := 17;
   No_Free_Connections     : constant Status_Code := 18;
   Not_Registered          : constant Status_Code := 19;
   Network_Unreachable     : constant Status_Code := 20;
   Host_Unreachable        : constant Status_Code := 21;
   Protocol_Not_Supported  : constant Status_Code := 22;
   No_Such_Socket          : constant Status_Code := 30;
   No_Response             : constant Status_Code := 31;
   Data_Too_Long           : constant Status_Code := 32;

end Transport_Defs;
```





```
with Interchange;
with Transport_Stream;

package Transport_Interchange is
   new Interchange.Operations (Stream_Id => Transport_Stream.Stream_Id,
                               Put       => Transport_Stream.Transmit,
                               Get       => Transport_Stream.Receive);
```

```
with Profile;
with Transport;
with Transport_Defs;
with Text_Io;

package Transport_Name is

    function Host_To_Network_Name (Host_Name : String)
                                   return Transport_Defs.Network_Name;

    function Host_To_Host_Id (Host_Name : String) return Transport_Defs.Host_Id;

    function Host_Id_To_Host (Host : Transport_Defs.Host_Id) return String;

    -- function Host_Id_To_Host (Network : Transport_Defs.Network_Name;
    --                           Host : Transport_Defs.Host_Id) return String;
    -- ... at end for upward-compatibility.

    Undefined : exception;

    type Host_Iterator is limited private;
    procedure Init  (Iter : in out Host_Iterator);
    procedure Next  (Iter : in out Host_Iterator);
    function  Value (Iter : Host_Iterator) return String;
    function  Done  (Iter : Host_Iterator) return Boolean;

    function Host_To_Machine_Type (Host_Name : String) return String;

    -- Host_to_Network_Name and Host_to_Host_Id begin by attempting to
    -- query a TCP/IP name server.  The object !Machine.Tcp_Ip_Name_Server
    -- is read as text: it must contain the Internet address of the server
    -- in decimal dotted notation.  The address is used to send a query in
    -- a UDP datagram.  If an answer is received, Host_to_Host_Id returns
    -- it, and Host_to_Network_Name returns "TCP/IP".  If any step of this
    -- process fails (for example, there is no Tcp_Ip_Name_Server file),
    -- the function instead attempts to resolve the name as follows:
    --
    -- Other name resolution operations are driven by a text file, namely
    -- !Machine.Transport_Name_Map.  The mapping from host_name to other
    -- values is changed by changing this file.  Each line of the file
    -- must contain the following, in the order given, separated by spaces:
    --
    --     a network_name,
    --     a host_id, in decimal notation xx.yy.zz, and
    --     a host_name, which must be an Ada simple identifier.
    --
    -- After the host name, each line may contain an optional machine_type:
```





```
    -- this identifies the operating system running on that host.
    -- See Ftp_Name_Map.Machine_Type.  If the machine_type is omitted,
    -- Host_to_Machine_Type will return the null string.
    --
    -- Comments, beginning with "--", may be added to the end of any line.

    function Local_Host_Name
                (Network : Transport_Defs.Network_Name) return String;

    -- Each user may keep maps from Host_Name to username, password
    -- and session for that host.  The maps are pointed to by the
    -- profile values Remote_Passwords (for usernames and passwords)
    -- and Remote_Sessions (for sessions).  See the documentation of
    -- package Profile for further details.

    function Host_To_Remote_Username
                (Host_Name : String;
                 Response  : Profile.Response_Profile := Profile.Get)
                return String;
    function Host_To_Remote_Password
                (Host_Name : String;
                 Response  : Profile.Response_Profile := Profile.Get)
                return String;
    function Host_To_Remote_Session
                (Host_Name : String;
                 Response  : Profile.Response_Profile := Profile.Get)
                return String;

    function Host_Id_To_Host (Network : Transport_Defs.Network_Name;
                              Host    : Transport_Defs.Host_Id) return String;

    -- A network object name consists of "!!", followed by a host
    -- name, followed by either "." or "!", followed by the name
    -- of an object within that host.

    function Is_Network_Object_Name (Name : String) return Boolean;
    -- Return true iff the name is a syntactically correct
    -- network object name.  The host_name need not be defined.

    function Network_Object_To_Host (Name : String) return String;
    -- If not Is_Network_Object_Name (Name) then raise Constraint_Error.
    -- Otherwise, return the host name part of a network object name,
    -- without the leading "!!" or trailing punctuation "!" or ".".
    -- The returned name is not neccessarily defined.

    function Network_Object_To_Rest (Name : String) return String;
    -- if not Is_Network_Object_Name (Name) then raise Constraint_Error.
    -- Otherwise, return the object name part of a network object name,
    -- beginning with "!" (even if the original punctuation was ".").
```

```
    -- The returned name does not neccessarily denote an extant object.

    package Service is
        -- A mapping from the name of a service to where it is available;
        -- that is, the Network(s) and Socket_Id(s) where its servers wait.

        -- This mapping is stored in the text object named
        File_Name : constant String := "!Machine.Transport_Services";

        -- Within this object, each non-empty text line contains
        --    * a Transport_Defs.Network_Name,
        --    * a Transport_Defs.Socket_Id,
        --    * a service name, and
        --    * optionally a machine name;
        -- in that order, separated by white space (blanks or tabs).
        -- Characters from "--" to the end of the line are a comment.  A
        -- Network_Name, Socket_Id, service name or machine name must not
        -- contain any white space or the character sequence "--".  The
        -- Transport_Defs.Socket_Id is in decimal dotted notation; that is, a
        -- sequence of non-negative decimal integers separated by periods.
        -- Each integer represents one or more bytes, as minimally required to
        -- represent that integer in binary form, most significant byte first.
        -- For example, 258 and 1.2 represent the same Socket_Id, which is
        -- expressed in Ada as (1,2).  A service name or a machine name should
        -- be a valid Ada identifier.  The case (upper or lower) of a service
        -- name or a machine name is not significant.

        -- A line signifies that the named service is available via the given
        -- network at the given socket in the given machine; if no machine
        -- name is given, the line signifies that the named service is
        -- available via the given network at the given socket in all machines
        -- not specifically named in other lines.

        function Normalize (Service_Name : String) return String;
        -- change to lower case, and strip leading or trailing white space.

        function Socket  (Service_Name, Host_Name : String)
                        return Transport_Defs.Socket_Id;
        function Network (Service_Name, Host_Name : String)
                         return Transport_Defs.Network_Name;
        -- Return the first defined Socket or Network associated with
        -- the given Service_Name and Host_Name.  If no such specific
        -- association is defined, return Socket (Service_Name) or
        -- Network (Service_Name), below.

        function Socket (Service_Name : String) return Transport_Defs.Socket_Id;
        function Network (Service_Name : String)
                         return Transport_Defs.Network_Name;
        -- Return the first defined Socket or Network associated with
```





```
        -- the given Service_Name, but no specific Host_Name.
        -- If no such association is defined, raise

        Undefined : exception;

        function Local_Socket
                    (Service_Name : String;
                     Network      : Transport_Defs.Network_Name := "TCP/IP")
                    return Transport_Defs.Socket_Id;
        -- Return the socket at which the given service is available in
        -- this machine as identified in the given Network.  That is:
        -- begin
        --     return Socket (Service_Name,
        --                    Transport_Name.Local_Host_Name (Network));
        -- exception
        --     when Transport_Name.Undefined =>
        --         return Socket (Service_Name);
        -- end;
        -- This function may raise Transport_Name.Service.Undefined.

        function Local_Network
                    (Service_Name : String;
                     Network      : Transport_Defs.Network_Name := "TCP/IP")
                    return Transport_Defs.Network_Name;
        -- Return the network via which the given service is available in
        -- this machine as identified in the given Network.  That is:
        -- begin
        --     return Network (Service_Name,
        --                     Transport_Name.Local_Host_Name (Network));
        -- exception
        --     when Transport_Name.Undefined =>
        --         return Network (Service_Name);
        -- end;
        -- This function may raise Transport_Name.Service.Undefined.

        -- A service may be available on multiple Socket/Network pairs;
        -- they may be found by iterating through all service definitions
        -- searching for definitions that match the service and host name:

        type Iterator is limited private;
        procedure Init   (Iter : in out Iterator);
        procedure Next   (Iter : in out Iterator);
        procedure Finish (Iter : in out Iterator); -- Done (Iter) becomes True.
        -- You are encouraged to Finish an Iterator when you're done using it,
        -- to release any global resource (actions, file handles) it may hold.

        function Done (Iter : Iterator) return Boolean;
        function Network (Iter : Iterator) return Transport_Defs.Network_Name;
        function Socket (Iter : Iterator) return Transport_Defs.Socket_Id;
```

```
        function Service_Name (Iter : Iterator) return String;
        function Host_Name (Iter : Iterator) return String;

        function Matches (Iter : Iterator; Service_Name, Host_Name : String)
                         return Boolean;
        -- Return true iff Service_Name (Iter) = Normalize (Service_Name)
        -- and Host_Name (Iter) = Normalize (Host_Name).

        function Matches (Iter : Iterator; Service_Name : String)
                         return Boolean;
        -- Return true iff Service_Name (Iter) = Normalize (Service_Name)
        -- and Host_Name (Iter) = "".

    private
        type Iterator is
            record
                File : Text_Io.File_Type;
                Last : Natural;
                Line : String (1 .. 2048);
            end record;
    end Service;

end Transport_Name;
```





```
with Transport;
with Transport_Defs;

generic
    with procedure Serve (Connection : Transport.Connection_Id) is <>;

    -- Service an incoming connection.  The connection is already
    -- open and connected when passed, and will be closed on return.

package Transport_Server is

    type Pool_Id is private;

    function Create (Network      : Transport_Defs.Network_Name;
                     Local_Socket : Transport_Defs.Socket_Id;
                     Max_Servers  : Natural := Natural'Last) return Pool_Id;

    -- Create a pool of server tasks, which may expand to have
    -- as many as MAX_SERVERS tasks in it.  Tasks are created
    -- in response to incoming connections on the given NETWORK
    -- and LOCAL_SOCKET.


    procedure Set_Max_Servers (Pool : Pool_Id; Max_Servers : Natural);

    -- Set the maximum number of server tasks which may be
    -- created for the pool.  If more tasks currently exist,
    -- they will continue to exist until they are done serving
    -- their connections, but no new tasks will be created.

    function Network      (Pool : Pool_Id) return Transport_Defs.Network_Name;
    function Local_Socket (Pool : Pool_Id) return Transport_Defs.Socket_Id;
    function Max_Servers  (Pool : Pool_Id) return Natural;
    function Servers      (Pool : Pool_Id) return Natural;


    procedure Finalize (Abort_Servers : Boolean := False);

    -- Terminate the tasks which depend on this instantiation,
    -- and close the transport.connections which they have open.
    -- By default, existing server tasks will continue to run
    -- until their clients (somewhere else in the network) close
    -- their connections.  If ABORT_SERVERS => TRUE, then the
    -- servers will be aborted immediately.
```

```
    -- Procedure finalize must be called before leaving a scope
    -- in which this package is instantiated (because of the Ada
    -- rules for task termination: see LRM section 9.4).

    -- In the cross-compiled version of this package (i.e. the
    -- one that uses shared elaboration), Finalize does nothing.

end Transport_Server;
```





```
with Transport;
with Transport_Defs;

package Transport_Server_Job is

    generic
        Network      : Transport_Defs.Network_Name;
        Local_Socket : Transport_Defs.Socket_Id;
        -- Where to wait for incoming calls.

        with procedure Server_Start;
        -- Start another server job running.

        with procedure Serve (Connection : Transport.Connection_Id);
        -- Process incoming requests from the Connection.

    procedure Server_Generic;
    -- Be a server, that is:
    -- Wait for an incoming connection on the given Network & Local_Socket.
    -- When a connection arrives, execute Server_Start; Serve (Connection).
    -- Repeat this process until the Local_Socket is in use, then return.

    procedure Change_Identity (Username, Password : String);
    -- Call Program.Change_Identity (Username, Password).
    -- That is, set the calling job's access control identity
    -- to the given Username.  If this fails, then raise
    -- Rpc.Username_or_Password_Error.  Otherwise return.

end Transport_Server_Job;
```

```
with Byte_Defs;
with Transport;
with Transport_Defs;

package Transport_Stream is

    -- A transport stream combines a transport.connection with
    -- some buffering, and provides a simplified transmit/receive
    -- interface.

    -- A facility is provided for creating pools of streams.
    -- Allocating and deallocating streams is usually fast.
    -- Unused streams are scavenged periodically.

    -- This package is useful for programs which want to communicate
    -- with some other machine, but don't want to be bothered with
    -- setting up and tearing down transport.connections, or don't
    -- want to deal with transport status codes.

    type Stream_Id is private;

    procedure Allocate (Stream     : out Stream_Id;
                        Connection :     Transport.Connection_Id);
    -- Create a stream around the given connection.  Such a
    -- stream has no pool, and will be closed as soon as it
    -- is deallocated.  The caller is responsible for opening
    -- and connecting the given connection.

    procedure Allocate (Stream  : out Stream_Id;
                        Is_New  : out Boolean;
                        Network :     Transport_Defs.Network_Name;
                        Host    :     Transport_Defs.Host_Id;
                        Socket  :     Transport_Defs.Socket_Id);
    -- Find or create a stream to the given network/host/socket.
    -- Is_New is returned true iff a new connection had to be
    -- built: i.e. there was not already a stream in the pool.

    procedure Deallocate (Stream : Stream_Id);
    -- Return a stream to its pool.  It will not be
    -- disconnected immediately, but may be scavenged.

    procedure Disconnect (Stream : Stream_Id);
    -- Disconnect the transport connection.

    procedure Transmit (Into : Stream_Id; Data : Byte_Defs.Byte_String);
    procedure Receive  (From : Stream_Id; Data : out Byte_Defs.Byte_String);
```





```
    procedure Flush_Transmit_Buffer (Stream : Stream_Id);
    function  Flush_Receive_Buffer
                 (Stream : Stream_Id) return Byte_Defs.Byte_String;
    -- If anything goes wrong with the above operations, the
    -- stream is disconnected, any associated buffers are
    -- deallocated, and the following exception is raised:

    Not_Connected : exception;
    -- Each stream is unsynchronized, that is, if two tasks call
    -- TRANSMIT at the same time, or two tasks call RECEIVE at the
    -- same time, the stream state will get screwed up.  Don't.

    type Pool_Id is private;

    function Create (Network       : Transport_Defs.Network_Name;
                     Remote_Host   : Transport_Defs.Host_Id;
                     Remote_Socket : Transport_Defs.Socket_Id;
                     Local_Socket  : Transport_Defs.Socket_Id :=
                        Transport_Defs.Null_Socket_Id) return Pool_Id;
    -- Create a pool of active streams.

    procedure Destroy (Pool : Pool_Id);
    -- Disconnect all streams in the pool,
    -- and terminate all associated tasks.

    procedure Allocate
                 (Stream : out Stream_Id; Pool : Pool_Id; Is_New : out Boolean);
    -- Get an idle stream from the pool.  If no idle stream
    -- is available, create one, open and connect a
    -- transport connection, and return IS_NEW = TRUE.
    -- If this fails, raise NOT_CONNECTED.

    procedure Scavenge (Pool : Pool_Id);
    -- Disconnect any streams which have been deallocated
    -- for a while.

    procedure Scavenge;
    -- Scavenge all pools.
    -- This procedure is called periodically (every minute)
    -- by a scavenger task inside the service.

    procedure Finalize;
    -- Destroy all pools, and terminate all tasks.

    -- Like all pools with explicit allocate and deallocate
    -- operations, this service is vulnerable to clients
    -- which allocate a resource and then fail to deallocate
    -- it, e.g. because they terminate abnormally.  This
```

```
    -- problem isn't easily solved in standard Ada.  If your
    -- system has some way of dealing with it, you might
    -- consider extending this service to do the right thing.

    function Connection (Stream : Stream_Id) return Transport.Connection_Id;

    subtype Unique_Id is Integer;

    function Unique (Stream : Stream_Id) return Unique_Id;

    procedure Set_User_Id (Stream : Stream_Id; User_Id : Integer := 0);
    function  Get_User_Id (Stream : Stream_Id) return Integer;

    -- This package keeps a map from Stream_Id to integer,
    -- which you can set and query using these subprograms.
    -- The User_Id is initialized to 0 when Is_New := True.
    -- The User_Id is not used by Transport_Stream: it is
    -- provided so that clients can associate higher level
    -- information with each stream.  For example, RPC uses
    -- it to record the RPC protocol version in use on the
    -- stream.

private

    type Pool_Type (Network_Length       : Natural;
                    Remote_Host_Length   : Natural;
                    Remote_Socket_Length : Natural;
                    Local_Socket_Length  : Natural);

    type Pool_List is access Pool_Type;

    type Pool_Id is new Pool_List;

    type Stream_Type;

    type Stream_List is access Stream_Type;

    Null_Unique_Id : constant Unique_Id := Unique_Id'First;

    type Stream_Id is
        record
            Stream : Stream_List;
            Unique : Unique_Id;
        end record;

end Transport_Stream;
```





```
with Default;
with Directory;
with Machine;

package Object_Editor is

    -- Detached jobs and jobs initiated via the program package have
    -- no associated image.

    type Focus is (Selection, Cursor, Image, Region);

    -- If not named specifically, an object (or collection of objects) can
    -- be designated by a highlighted selection on an image or by the
    -- location of the cursor within an image.  Different algorithms
    -- for determining such designated objects are implemented by this
    -- package to satisfy the differing requirements of environment
    -- commands (taking one argument denoted in this way)

    -- Selection

    --      An object is selected in this mode only if its declaration is
    --      highlighted and the cursor is within or immediately adjacent to
    --      that highlighted region.

    --      This is the most restrictive selection mode and would be used
    --      by commands, such as delete and demote, that must have a
    --      strong indication from the user of the intended object.


    -- Cursor

    --      An object is selected in this mode if its declaration is
    --      selected as described in the preceding case or, if there is no
    --      highlighted selection on the image of the cursor, the cursor
    --      is on a portion of the object's declaration.

    --      This is less restrictive than the preceding case in that it
    --      accepts any placement of the cursor within an image.  Commands
    --      such as definition or help would use this mode because it
    --      reduces the number of key strokes required by the user.


    -- Image

    --      An object is selected in this mode if its declaration is
    --      selected as described in the Selection mode or, if there is no
    --      highlighted selection on the image of the cursor, the cursor is
```

```
--     on the image associated with the object.

--     This mode accepts the same configurations of cursor and
--     selection as in the Cursor mode, but treats the cursor less
--     specifically by selecting the object that is represented by the
--     entire image rather than the object that is represented by the
--     portion of the image that the cursor is on.


-- Region

--     An object is selected in this mode if its declaration is
--     highlighted. The cursor does NOT have to be in the region.

--     This is less restrictive than Selection focus and would be
--     used by commands such as copy to obtain their source object.


-- Some object editors will behave the same in the Cursor and
-- Image modes because they do not support the notion of designations
-- nested within an image. For such editors, the Selection mode will
-- always fail unless the selection constitutes the entire image.

procedure Get_Declaration
            (Decl      : out Directory.Declaration;
             Status    : out Directory.Naming.Name_Status;
             Precision : Object_Editor.Focus := Object_Editor.Selection;
             Job       : Default.Process_Id := Default.Process);

-- Returns the declaration for the object designated by the current
-- selection according to the specified algorithm.

procedure Get_Object (Object    : out Directory.Object;
                      Status    : out Directory.Naming.Name_Status;
                      Class     : Directory.Class := Directory.Nil;
                      Precision : Object_Editor.Focus :=
                         Object_Editor.Selection;
                      Job       : Default.Process_Id := Default.Process);

-- Returns the Directory object designated by the current selection
-- according to the specified algorithm.


function Get_Text
           (Precision : Object_Editor.Focus := Object_Editor.Region;
            Job : Default.Process_Id := Default.Process) return String;
-- Returns the text contained in the current focus.  Region selections
-- are allowed.  Text may include Ascii.LF's to indicate end of line.
```





```
function Get_Name
            (Precision : Object_Editor.Focus := Object_Editor.Image;
             Job : Default.Process_Id := Default.Process) return String;
-- Return the name of the image, which may not be a valid directory name,
-- or the name of the declaration/object associated with the
-- current cursor/selection/region/window.


-- IMAGE MANIPULATION OPERATIONS

procedure Release_Access (The_Object : Directory.Object);
-- acquire rights to the designated object if they are held by the current
-- session; allows tools to guard against being locked out by other windows

--  procedure Display_Declaration (For_Object : Directory.Object;
--                                 In_Library : Boolean := False;
--                                 In_Place : Boolean := False);
--  procedure Display_Declaration (For_Version : Directory.Version;
--                                 In_Library : Boolean := False;
--                                 In_Place : Boolean := False);
--
--  procedure Display (The_Object : Directory.Object;
--                     In_Place : Boolean := False);
--  procedure Display (The_Version : Directory.Version;
--                     In_Place : Boolean := False);
--
--  -- In_Library will cause ada subunits to have their enclosing
--  -- library displayed instead of their stub declaration in their
--  -- parent ada unit.
pragma Consume_Offset (4);

procedure Update_Changed_Image (The_Object : Directory.Object);
-- notify the editor that changes have been made to the object that it
-- doesn't know about yet

procedure Update_Changed_Images;
-- non-specific (and slower) version of the above


subtype Editor_Name is String;

function Name return Editor_Name;
-- Returns the name of the object editor associated with the current image.

function Supports_Declarations
            (Editor : Editor_Name := Name) return Boolean;
function Supports_Subobjects (Class  : Directory.Class := Directory.Nil;
                              Editor : Editor_Name := Name) return Boolean;
-- Indicates whether the named editor can ever associate
```

```
    -- objects/declarations with the cursor or selection (other than the
    -- object/declaration associated with the containing image).

    -- Iterate over all object editors.  Primarily for completeness, but
    -- allows tool writers to determine the set of applicable OE's.

    type Iterator is private;
    procedure Init  (Iter : out Iterator);
    function  Done  (Iter : Iterator) return Boolean;
    function  Value (Iter : Iterator) return Editor_Name;
    procedure Next  (Iter : in out Iterator);


    -- Returns the job that the editor regards as current
    function Current_Job return Machine.Job_Id;

    procedure Display_Declaration (For_Object : Directory.Object;
                                   In_Library : Boolean := False;
                                   In_Place   : Boolean := False);
    procedure Display_Declaration (For_Version : Directory.Version;
                                   In_Library  : Boolean := False;
                                   In_Place    : Boolean := False);

    procedure Display (The_Object : Directory.Object;
                       In_Place   : Boolean := False);
    procedure Display (The_Version : Directory.Version;
                       In_Place    : Boolean := False);

end Object_Editor;
```





```
with Directory;

-- All options appear in Adaesque aggregate notation, with appropriate
-- relaxations of the rules.  Switch values and switch names, where the set
-- of choices is static (i.e. fixed set of switches, but not for User
-- names), should recognize unique prefix.  A package is available for
-- parsing parameter lists that adhere to this convention.

-- Options are formed from the following lexical components:
--
--     Punctuation ::=  '=>' | '=' | ':=' | '|' | '..'
--                 ::=  Separator
--
--     Separator   ::=  ',' | ';'
--
--     Value       ::=  Directory-String-Name
--                 ::=  Integer-Literal
--                 ::=  Float-Literal
--                 ::=  Literal
--                 ::=  '<>'             --  The default value defined for an
--                                       --  option.
--                 ::=  Other-Value      --  Any sequence of contiguous char-
--                                       --  acters not including separators;
--                                       --  leading and trailing blanks are
--                                       --  removed.
--                 ::=  '(' Balanced ')' --  Any sequence of contiguous char-
--                                       --  acters, balanced with respect to
--                                       --  parentheses, nested within
--                                       --  parentheses. The outer-most
--                                       --  enclosing parentheses are not part
--                                       --  of the value, but all contained
--                                       --  characters, including blanks, are.
--
--     Name        ::=  Simple-Ada-Name
--                 ::=  Other-Name       --  Any sequence of contiguous char-
--                                       --  acters not including punctuation
--                                       --  or blanks
--
--     Literal     ::=  Simple-Ada-Name

--     Any two-character sequence begining with \ is interpreted as a single,
--     non-special occurence of the second character.  Thus, "\," is NOT a
--     separator character, but a benign ocurrence of ','.
--
--     Blanks are allowed around the special characters:
--
--     Since Blanks are not allowed in a Name, a blank following a Name may be
```

```
--    used as a separator.
--
-- The syntax is (enclosing quotes not included):
--
--    Options     ::=  [Option {Separator Option}]
--
--    Option      ::=  Range {'|' Range} [('=>' | ':=' | '=') Value]
--                ::=  ['~'] Range {'|' Range}
--                ::=  Literal
--                ::=  '_' File-Name
--
--    Range       ::=  Name ['..' Name]
--                ::=  'others'         --  Denotes names not otherwise
--                                      --  specified.
--
--
-- General semantics:

-- A Name denotes an option. A Range denotes the options in a predefined
-- sequence of options from the option denoted by the first name to and
-- including the option denoted by the last name of the range.  The
-- specified Value is associated with each option denoted by the Names and
-- Ranges of the Option.

-- If a Value clause is omitted, the options denoted by the Names and
-- Ranges of the Option must be Boolean-valued.  If '~' preceeds
-- the Names of an Option, the options assume the value False, otherwise they
-- assume the value True.

-- A Literal denotes a value of a specific option.  When it appears as an
-- Option, it denotes both the Name and the Value of that option.

-- If a name appears more than once, the value associated with the leftmost
-- instance of the name is the one used.

-- Examples:

-- Access_List.Set ("Public=>RW");
-- Profile.Set  ("+++,---,++*,!!!,l=>80");
-- Profile.Set ("Persevere, +++ | --- | ++* | !!! => true,w => 80");
-- Source_Archive.Restore (..., Form => "New_Units, Acl => (Public=>RW)");
-- Switches.set ("Semantics.Ignore_Minor_Errors := True");


generic
    type Option_Id is (<>);

    -- This discrete type defines the ordering of option names used to
    -- define ranges.  Its values are used in the programmatic interface to
```





```
-- identify options.  For a static collection of options, such as in
-- Profile, Option_Id would probably be an enumerated type; its
-- enumeration ids would define the names of the options.  For
-- non-static options, such as access lists, Option_Id would be an
-- integer type and most names would be defined using the define
-- procedure exported by the generic.

Nil   : in Option_Id := Option_Id'First;
First : in Option_Id := Option_Id'Succ (Nil);
Last  : in Option_Id := Option_Id'Last;

-- Nil is an Option_Id that represents no option name.  Only option_id's
-- in the range First .. Last are definable; Nil should not be in that
-- range;

Option_Kinds : in String := "others => Unspecified";

-- Specification of the kind of each option.  The string must satisfy
-- the syntax for a forms parameter in which Names are taken from the
-- set of Option_Id images and Values are the enumeration ids of the
-- type Option_Kind, defined below.  (The Option_Kind 'Literal' may not
-- be specified in this string; use the Define function.) The default
-- kind for all options is 'Unspecified'. Options that are not
-- Boolean-Valued must be followed by a Value clause when they are used
-- in a parameter string. If the Kind of an option is other than
-- 'Unspecified', the parser will verify that the associated value is of
-- the proper form for the specified Kind.

Default_Values : in String := "";

-- Default values for the options.  The string must satisfy the syntax
-- for a forms parameter in which Names are taken from the set of
-- Option_Id images.  Not all Option_Ids need to have Default_Values.
-- Default values are substituted for the special symbol '<>' when it
-- appears in a Value clause.  If no default value has been specified
-- for an option and the option appears with a '<>' value, the reference
-- to the option is deleted by the option parser.

Alternate_Names : in String := "";

-- Alternate names for the options.  The string must satisfy the syntax
-- for a forms parameter in which Names are taken from the set of
-- Option_Id images, and the Values obey the syntax for Other_Names.
-- Not all Option_Ids need to have Alternate_Names.  The standard name
-- for an option is the Image of the Option_Id.  The Undefine function
-- may be used to remove the standard name from the set of permitted
-- names. All names for the same option_id value have the same kind and
-- default value.
```

```
    From : Option_Id := First;
    To   : Option_Id := Last;

    -- From and To define the range of Option_Id's that make up the
    -- initial set of defined options.  This set can be expanded or
    -- reduced using the Define and Undefine procedures defined in the
    -- package.

package Parameter_Parser is

    type Option_Kind is (Unspecified, Directory_String_Name, Boolean_Valued,
                         Integer_Valued, Float_Valued, Literal);

    procedure Define (Option            : Option_Id;
                      Name              : String      := "";
                      Kind              : Option_Kind := Unspecified;
                      Default_Value     : String      := "";
                      Allow_Name_Prefix : Boolean     := False);

    -- Defines a new Name to be associated with the given Option_Id.  The
    -- default Name is the Option_Id'image of the Option.  Any number of
    -- names may be associated with an Option_Id value.  The parameter
    -- specification may use any of these names to set the option.

    -- Allow_Name_Prefix allows a unique prefix of the Name to be used in
    -- place of the Name in a parameter specification.

    -- The Default_Value string is parsed as if it were a Value
    -- specification; a balanced string or '\' must be used to protect
    -- separators in the default value.  If Default_Value is the null
    -- string, no default value is assigned to the option.  If you want the
    -- default value to be a null string, use "()".

    -- If Kind is 'Literal', Name must be non-null.  The given Name must
    -- never appear with a Value clause. When it appears by itself, without
    -- a Value clause, the implied value is the Name itself.  (The generic
    -- package Enumerated_Value, defined below, provides a convenient way to
    -- define all enumeration ids of an option as literals.)

    procedure Undefine (Name : String);

    -- Removes the Name (and its prefixes) as a possible option name.

    procedure Undefine (From : Option_Id; To : Option_Id := Nil);

    -- Removes all names and prefixes that denote an Option_Id in the given
    -- range as possible option names.

    procedure Allow_Name_Prefix (Name : String; Value : Boolean := True);
```





```
    procedure Allow_Name_Prefix (Value : Boolean   := True;
                                 From  : Option_Id := Nil;
                                 To    : Option_Id := Nil);

    -- The Allow_Name_Prefix flag for a name, when set, allows a unique
    -- prefix of the name to be used in place of the name in a parameter
    -- specification.  The default setting of this flag for the initial set
    -- of Option_Ids is true.

    -- The first procedure sets the Allow_Name_Prefix flag for the named
    -- option.

    -- The second procedure sets the Allow_Name_Prefix flag for all defined
    -- Names that map to an Option_Id in the specified range.  The range
    -- implied by the default values is the full set of Option_Id's defined
    -- at the time of call.  If From is non-Nil and To is Nil, the single
    -- Option_Id From is implied.  If From and To are both non-Nil, all
    -- Option_Id's in the range From .. To are implied.

    type Iterator is private;

    procedure Parse (Parameter :     String;
                     Options   : out Iterator;
                     Success   : out Boolean);

    function Parse (Parameter : String) return Iterator;

    function Is_Successful (Iter : Iterator) return Boolean;

    -- Success is True, iff all options were parsed correctly.  When no
    -- options parsed correctly, a Done iterator is returned, which may be
    -- passed to the Diagnosis function to obtain more information.  If
    -- some, but not all, options were parsed correctly the returned
    -- iterator will be non-null.  Iterations (positions in the iterator)
    -- are allocated for erroneous specifications as well as for correctly
    -- parsed specifications.  The Is_Ok() predicate distinguishes between
    -- them.

    -- The iterator returned by Parse represents an expanded, unfactored
    -- specification, equivalent to the input specification; each iteration
    -- represents a simple specification of the form, "Name [=> Value]." All
    -- Names are returned with their full spelling.  Ranges and the reserved
    -- name 'others' are expanded so that there is one iteration for each
    -- option_id covered by the Range or 'others.' Duplicate specifications
    -- have been removed, leaving the last specification at its point of
    -- occurrence.  Except for the deleted duplications all specifications
    -- of the input string are present in the iterator in the same order as
    -- in the input string.
```

```
-- In the following subprograms, the optional Name parameter is used
-- to interrogate the iterator as a set.  The default value, Nil,
-- addresses the current iteration of the iterator.

function Is_Ok (Iter : Iterator; Name : Option_Id := Nil) return Boolean;

-- Indicates whether the designated option was syntactically correct in
-- the specification; If the named option was not specified, Is_Ok()
-- returns False;

function Is_Present (Iter : Iterator; Name : Option_Id) return Boolean;

-- Indicates whether the indicated option was present in the option
-- parameter string.  An option is present if its name was
-- parsable.  It may otherwise be in error.

function Diagnosis (Iter : Iterator; Name : Option_Id := Nil) return String;

-- Returns text for a message that describes what was wrong with the
-- option specification, if anything.  If the named option was not
-- specified, Diagnosis returns a message to this effect.  If an
-- option Is_Ok(), Diagnosis returns the null string.

function  Done  (Iter : Iterator) return Boolean;
procedure Next  (Iter : in out Iterator);
procedure Reset (Iter : in out Iterator);

-- Advances the iterator to the next iteration.

function Name (Iter : Iterator) return Option_Id;

-- Returns Nil if the iteration corresponds to an unparsable
-- specification.

function Name (Iter : Iterator; Name : Option_Id := Nil) return String;

-- Returns the name that was used in the specification to denote the
-- indicated Option_Id.  The full name is returned even if a prefix was
-- used.

function Has_Value
            (Iter : Iterator; Name : Option_Id := Nil) return Boolean;

-- Indicates whether the Value clause for the indicated option was
-- specified or not.

function Get_Image (Iter    : Iterator;
                    Name    : Option_Id := Nil;
                    Default : String    := "") return String;
```





```
-- Get_Image returns the uninterpreted image of the Value associated
-- with the indicated option.  If Is_OK() or Has_Value() is false,
-- the null string is returned. If Is_Present() is false, the
-- default value associated with the named option is returned.

-- In the returned value, the two-character sequences begining with \
-- have been reduced to a single character.

-- The Get_xxx functions defined below use Get_Image to obtain a
-- string to interpret.  Thus they operate on the default value
-- when the option has not been named.

function Kind (Iter : Iterator; Name : Option_Id := Nil) return Option_Kind;

-- The value of Get_Image() on the specified option is inspected to
-- determine its type. Kind returns Unspecified if the kind cannot be
-- determined.

function Get_Object (Iter    : Iterator;
                     Name    : Option_Id        := Nil;
                     Default : Directory.Object := Directory.Nil)
                    return Directory.Object;

-- The value of Get_Image() is evaluated by Directory.Naming.-
-- Resolve. Directory.Nil is returned if it cannot return a
-- Directory.Object value.  The results of the attempt to resolve
-- the directory name will also be reflected in Is_Ok() and
-- Diagnosis() after the call to Get_Object.

function Get_Objects (Iter         : Iterator;
                      Name         : Option_Id := Nil;
                      Deleted_Ok   : Boolean   := False;
                      Objects_Only : Boolean   := False)
                     return Directory.Naming.Iterator;

-- The value of Get_Image() is evaluated by Directory.Naming.-
-- Resolve. A Done iterator is returned if it cannot be resolved.
-- The results of the attempt to resolve the directory name will
-- also be reflected in Is_Ok() and Diagnosis() after the call to
-- Get_Objects.

function Get_Boolean (Iter    : Iterator;
                      Name    : Option_Id := Nil;
                      Default : Boolean   := False) return Boolean;

-- If Get_Image() is non-null, Get_boolean tries to interpret it as
-- a Boolean_Literal and returns the denoted value. If the it is not
-- a Boolean_Literal, False is returned, and Is_Ok() and Diagnosis()
-- will indicate an error and the nature of the error. A
```

```
-- Boolean_Literal may be any prefix of True or False.

-- If Get_Image() is null, Get_Boolean returns false if the name
-- appeared with a '~' and it returns true otherwise.

function Get_Integer (Iter    : Iterator;
                      Name    : Option_Id := Nil;
                      Default : Integer   := Integer'Last) return Integer;

-- Get_Integer tries to parse the Get_Image() value as an
-- integer and returns the denoted value.  If Integer'Value fails,
-- Integer'last is returned and Is_OK() and Diagnosis() will
-- identify the error.

function Get_Float (Iter    : Iterator;
                    Name    : Option_Id := Nil;
                    Default : Float     := Float'Safe_Large) return Float;

-- Get_Float tries to parse the Get_Image() value as a float
-- literal and returns the denoted value.  If it cannot parse the
-- value as a float value, Float'large is returned and Is_OK() and
-- Diagnosis() will identify the error.

generic
    type T is (<>);
    Nil               : in T         := T'First;
    Id                :    Option_Id := Parameter_Parser.Nil;
    Allow_Name_Prefix :    Boolean   := True;
package Enumerated_Value is
    function Get_Enumeration (Iter : Iterator;
                              Name : Option_Id := Parameter_Parser.Nil;
                              Allow_Value_Prefix : Boolean := True;
                              Default : T := Nil) return T;
end Enumerated_Value;

-- Get_Enumeration tries to interpret the Get_Image() value as the
-- unique prefix of an image of a component of T.  It returns Nil and
-- preps Is_Ok() and Diagnosis() if no such interpretation is possible.
-- If Allow_Value_Prefix is false, only full spellings of values of type
-- T are recognized.

-- If the Id formal parameter is not Nil, the values of type T will be
-- defined as legal option names.  If one of these values is found in an
-- option list in the place of a name, it will be treated as a value of
-- the option denoted by Id (i.e., "Id =>" is implicitly inserted before
-- the value).   If Allow_Name_prefix is True, unique prefixes of the
-- values of T will be recognized.
```





```
    generic
        type T is private;
        Nil : in T;
        with function Value     (S : String) return T is <>;
        with function Diagnosis (S : String) return String;
    function Get_Value (Iter    : Iterator;
                        Name    : Option_Id := Parameter_Parser.Nil;
                        Default : T         := Nil) return T;

    -- Get_Value applies the formal Value function to the Get_Image()
    -- value.  Value should return Nil if the passed value is
    -- unacceptable.  If Nil is returned, the next call to Diagnosis
    -- will return the value returned by Get_Value.Diagnosis.

private
    type Iterator_Data;
    type Iterator is access Iterator_Data;
    pragma Segmented_Heap (Iterator);

end Parameter_Parser;
```

```
with Directory;
with Simple_Status;

package Profile is

    -- A collection of job-related utilities for control of log generation,
    -- Activity files, and error reactions.  These facilities are used by
    -- most packages in !Commands, and may be used by user-written procedures.

    type Response_Profile is private;
    -- The aggregate of all components of the job response profile

    function Get return Response_Profile;
    -- Profile for the current job

    function Get_Default return Response_Profile;
    -- Profile for the Session (which is the default for a job that
    -- does not specify one)

    procedure Set (Profile : Response_Profile);
    -- Set profile for rest of current job

    procedure Set_Default (Profile : Response_Profile);
    -- Set profile for session; this is the value used for the job
    -- response profile if none is otherwise specified.

    function Default_Profile return Response_Profile;
    -- If the user has established no response profile for his session,
    -- this profile is used; it is the aggregate of all the Default_xxx
    -- constants defined in this package.

    subtype Name is String;  -- an unambiguous string name

    -- A map is maintained between a job id and the following values:

    -- The ERROR REACTION specifies which of several options a command is
    -- to follow in reacting to error situations.

    -- The LOG_FILTER specifies in some detail, the desired content of the
    -- log that is generated by the command.

    -- The LOG_PREFIXES specifies the format of messages entered into the log.

    -- The WIDTH specifies the number of columns in the log display.

    -- The LOG_FILE specifies the predefined file to be used by the Log
    -- package to generate output.
```





```
    -- The ACTIVITY specifies the activity file used for loading subsystems.

    -- The REMOTE_PASSWORDS specifies the file in which usernames and
    -- passwords for remote machines are stored.

    -- The REMOTE_SESSIONS specifies the file in which session names
    -- for remote machines are stored.

    -- Procedures are provided for setting each of these values into
    -- the map on a job or session basis, and functions are provided
    -- for retrieving the current value from these maps.

    -- Error_Reaction specifies how a command is to respond to errors
    -- along two dimensions:

    -- Perseverance    whether to continue processing or stop at the first
    --                 error.  (If a log is to be generated, the process
    --                 perseveres long enough to print an error message
    --                 regardless of the setting of this option.)

    -- Exception       whether or not a process is to propagate an
    -- Propagation     exception to its caller when it terminates a run in
    --                 which errors occurred. (If processing has
    --                 persevered, ERROR is raised.)

    type Error_Reaction is (Quit, Propagate, Persevere, Raise_Error);

    -- Quit        Command terminates at the first error. It may log
    --             the error in the job error map, but may not
    --             propagate an exception to its caller

    -- Propagate   Command terminates at the first error by raising
    --             an exception.  An entry should be made to the job
    --             error map. Profile.Error may be raised if no other
    --             exception is appropriate.

    -- Persevere   Command continues after errors at its discretion.
    --             Exceptions may not be propagated to its caller.

    -- Raise_Error Command continues after errors at its discretion.
    --             The command must raise an exception if it was unable
    --             to complete successfully.

    Default_Reaction : constant Error_Reaction := Persevere;

    procedure Set_Reaction (Reaction : Error_Reaction);

    procedure Set_Default_Reaction (Reaction : Error_Reaction);
```

```
function Reaction (Response : Response_Profile := Profile.Get)
                  return Error_Reaction;

function Persevere
            (Response : Response_Profile := Profile.Get) return Boolean;
-- true if error reaction is Persevere or Raise_Error; i.e. if command
-- is to continue after an error

function Propagate
            (Response : Response_Profile := Profile.Get) return Boolean;
-- true if error reaction is Propagate or Raise_Error; i.e., if a
-- command is to raise an exception when it is done.

-- Log Filter

type Msg_Kind is (Auxiliary_Msg, Debug_Msg, Note_Msg, Positive_Msg,
                  Position_Msg, Negative_Msg, Warning_Msg, Error_Msg,
                  Exception_Msg, Sharp_Msg, At_Msg, Dollar_Msg);

-- Log messages of any class can be filtered out of the log as it is
-- being generated using the procedures defined below.
type Log_Filter is array (Msg_Kind) of Boolean;

-- The filter specifies what types of messages are to be printed.

Quiet   : constant Log_Filter := Log_Filter'(others => False);
Full    : constant Log_Filter := Log_Filter'
                                  (Debug_Msg => False, others => True);
Terse   : constant Log_Filter := Log_Filter'
                                  (False, False, False, others => True);
Errors  : constant Log_Filter :=
   Log_Filter'(Negative_Msg .. Exception_Msg => True, others => False);
Summary : constant Log_Filter :=
   Log_Filter'(Positive_Msg | Negative_Msg => True, others => False);

Default_Filter : constant Log_Filter := Full;

function Filter (Response : Response_Profile := Profile.Get)
                return Log_Filter;

function Includes
            (Kind : Msg_Kind; Response : Response_Profile := Profile.Get)
            return Boolean;

-- iff includes (Kind, Response) is true then messages of that Kind are
-- sent to the log.

procedure Include            (Kind : Msg_Kind; Value : Boolean := True);
procedure Include_In_Default (Kind : Msg_Kind; Value : Boolean := True);
```





```
-- Change the filter value for the given message kind

procedure Set_Filter (Auxiliaries : Boolean := True;
                      Diagnostics : Boolean := True;
                      Notes       : Boolean := True;
                      Positives   : Boolean := True;
                      Negatives   : Boolean := True;
                      Positions   : Boolean := True;
                      Warnings    : Boolean := True;
                      Errors      : Boolean := True;
                      Exceptions  : Boolean := True;
                      Sharps      : Boolean := True;
                      Dollars     : Boolean := True;
                      Ats         : Boolean := True);

procedure Set_Default_Filter (Auxiliaries : Boolean := True;
                              Diagnostics : Boolean := True;
                              Notes       : Boolean := True;
                              Positives   : Boolean := True;
                              Negatives   : Boolean := True;
                              Positions   : Boolean := True;
                              Warnings    : Boolean := True;
                              Errors      : Boolean := True;
                              Exceptions  : Boolean := True;
                              Sharps      : Boolean := True;
                              Dollars     : Boolean := True;
                              Ats         : Boolean := True);

procedure Set_Filter (Filter : Log_Filter);
-- Establishes the given filter value(s) as the current filter value
-- for the job.

procedure Set_Default_Filter (Filter : Log_Filter);
-- Establishes the given filter value(s) as the default filter value
-- for the session.

-- Log Format
-- Specifies the prefixes desired for each message and the length of
-- the line used in formating messages

type Log_Prefix is (Nil, Time,       -- 11:00:00 PM
                    Hr_Mn_Sc,    -- 23:00:00
                    Hr_Mn,       -- 23:00
                    Date,        -- September 29, 1983
                    Mn_Dy_Yr,    -- 09/29/83
                    Dy_Mon_Yr,   -- 29-SEP-83
                    Yr_Mn_Dy,    -- 83/09/29
                    Symbols      -- +++, ++*, etc
                    );
```

```
-- Each prefix (except Nil) is followed by a single blank

type Log_Prefixes is array (1 .. 3) of Log_Prefix;

-- Any combination of up to three prefixes may be specified

Default_Prefixes : constant Log_Prefixes := (Yr_Mn_Dy, Hr_Mn_Sc, Symbols);

function Prefixes (Response : Response_Profile := Profile.Get)
                  return Log_Prefixes;

procedure Set_Prefixes (Prefixes : Log_Prefixes);
procedure Set_Prefixes (Prefix1, Prefix2, Prefix3 : Log_Prefix := Nil);

procedure Set_Default_Prefixes (Prefixes : Log_Prefixes);
procedure Set_Default_Prefixes
             (Prefix1, Prefix2, Prefix3 : Log_Prefix := Nil);

Default_Width : constant Natural := 77;  -- default width of log display

procedure Set_Width         (Width : Natural);
procedure Set_Default_Width (Width : Natural);

function Width (Response : Response_Profile := Profile.Get) return Natural;


-- ACTIVITY:

subtype Activity_Type is Directory.Object;

function Default_Activity return Activity_Type;

procedure Set_Activity         (Activity : Activity_Type);
procedure Set_Default_Activity (Activity : Activity_Type);

function Activity (Response : Response_Profile := Profile.Get)
                  return Activity_Type;


-- LOG_FILE

type Log_Output_File is (Use_Output, Use_Error,
                         Use_Standard_Output, Use_Standard_Error);

Default_Log_File : constant Log_Output_File :=
   Use_Output;                -- default file for log

procedure Set_Log_File         (Log_File : Log_Output_File);
procedure Set_Default_Log_File (Log_File : Log_Output_File);
```





```
function Log_File (Response : Response_Profile := Profile.Get)
                  return Log_Output_File;

function Response (Reaction : Error_Reaction  := Profile.Reaction;
                   Filter   : Log_Filter      := Profile.Filter;
                   Prefixes : Log_Prefixes    := Profile.Prefixes;
                   Width    : Natural         := Profile.Width;
                   Activity : Activity_Type   := Profile.Activity;
                   Log_File : Log_Output_File := Profile.Log_File)
                  return Response_Profile;

procedure Set_Response (Reaction : Error_Reaction  := Profile.Reaction;
                        Filter   : Log_Filter      := Profile.Filter;
                        Prefixes : Log_Prefixes    := Profile.Prefixes;
                        Width    : Natural         := Profile.Width;
                        Activity : Activity_Type   := Profile.Activity;
                        Log_File : Log_Output_File := Profile.Log_File);

procedure Set_Default_Response
             (Reaction : Error_Reaction :=
                 Profile.Reaction (Profile.Get_Default);
              Filter   : Log_Filter := Profile.Filter (Profile.Get_Default);
              Prefixes : Log_Prefixes :=
                 Profile.Prefixes (Profile.Get_Default);
              Width    : Natural := Profile.Width (Profile.Get_Default);
              Activity : Activity_Type :=
                 Profile.Activity (Profile.Get_Default);
              Log_File : Log_Output_File :=
                 Profile.Log_File (Profile.Get_Default));

function Ignore (Reaction : Error_Reaction  := Profile.Persevere;
                 Filter   : Log_Filter      := Profile.Quiet;
                 Prefixes : Log_Prefixes    := Profile.Prefixes;
                 Width    : Natural         := Profile.Width;
                 Activity : Activity_Type   := Profile.Activity;
                 Log_File : Log_Output_File := Profile.Log_File)
                return Response_Profile renames Response;

function Warn (Reaction : Error_Reaction  := Profile.Persevere;
               Filter   : Log_Filter      := Profile.Full;
               Prefixes : Log_Prefixes    := Profile.Prefixes;
               Width    : Natural         := Profile.Width;
               Activity : Activity_Type   := Profile.Activity;
               Log_File : Log_Output_File := Profile.Log_File)
              return Response_Profile renames Response;

function Verbose (Reaction : Error_Reaction  := Profile.Reaction;
                  Filter   : Log_Filter      := Profile.Full;
                  Prefixes : Log_Prefixes    := Profile.Prefixes;
```

```
                  Width    : Natural         := Profile.Width;
                  Activity : Activity_Type   := Profile.Activity;
                  Log_File : Log_Output_File := Profile.Log_File)
                 return Response_Profile renames Response;

    function Raise_Exception (Reaction : Error_Reaction  := Profile.Propagate;
                              Filter   : Log_Filter      := Profile.Filter;
                              Prefixes : Log_Prefixes    := Profile.Prefixes;
                              Width    : Natural         := Profile.Width;
                              Activity : Activity_Type   := Profile.Activity;
                              Log_File : Log_Output_File := Profile.Log_File)
                             return Response_Profile renames Response;

    No_Prefixes : constant Log_Prefixes := (Nil, Nil, Nil);

    function Nil (Reaction : Error_Reaction  := Profile.Quit;
                  Filter   : Log_Filter      := Profile.Quiet;
                  Prefixes : Log_Prefixes    := Profile.No_Prefixes;
                  Width    : Natural         := Profile.Default_Width;
                  Activity : Activity_Type   := Profile.Default_Activity;
                  Log_File : Log_Output_File := Profile.Default_Log_File)
                 return Response_Profile renames Response;

    Error : exception;


    ----------------------
    -- String Interface --
    ----------------------

    -- The strings accepted and returned by the following subprograms obey
    -- the syntax for form parameters. The Option names and expected values
    -- are as follows:

    -- Option Name              Value

    -- REACTION                 QUIT, PROPAGATE, PERSEVERE, RAISE_ERROR
    --
    -- AUXILIARY_MSG, :::       TRUE or FALSE
    -- DEBUG_MSG, ???                "
    -- NOTE_MSG, ---                 "
    -- POSITIVE_MSG, +++             "
    -- POSITION_MSG, >>>             "
    -- NEGATIVE_MSG, ++*             "
    -- WARNING_MSG, !!!              "
    -- ERROR_MSG, ***                "
    -- EXCEPTION_MSG, %%%            "
    -- SHARP_MSG, ###                "
    -- AT_MSG, @@@                   "
```





```
    -- DOLLAR_MSG, $$$               "
    --
    -- PREFIX                   (form[, form[, form]]) where form is one of
    --                          TIME, HR_MN_SC, HR_MN,
    --                          DATE, MN_DY_YR, DY_MON_YR, YR_MN_DY,
    --                          SYMBOLS
    --                          e.g. (TIME, DATE, SYMBOLS)
    --                               (HR_MN_SC, SYMBOLS)
    --
    -- LINE_WIDTH               positive integer in 1..1024
    --
    -- ACTIVITY                 name of activity file
    --
    -- LOG_FILE                 USE_OUTPUT, USE_ERROR, USE_STANDARD_OUTPUT, or
    --                          USE_STANDARD_ERROR


    -- The Options enclosed in "<>" brackets take no value, but denote the
    -- current value of the named profile.  These values may be used by
    -- themselves to denote all components of the named profile or as the value
    -- of one of the profile components to denote just that component of the
    -- named file, e.g. Activity=><DEFAULT>.  Where special symbols appear,
    -- e.g. +++, any symbols not mentioned are turned off.
    --
    -- EDITOR'S NOTE: THE FOLLOWING DESCRIPTIONS ARE *NOT* COMPLETELY
    -- ACCURATE. FOR ACCURATE DESCRIPTIONS, SEE THE PARAMETER-VALUE
    -- CONVENTIONS TABBED SECTION IN THIS REFERENCE SUMMARY.
    --
    -- <DEFAULT>                System default, ignores user profile.
    --                          PERSEVERE, :::, ~???, ---..$$$, Width=>77
    --                          Prefix=>YR_MN_DY, HR_MN_SC SYMBOLS, USE_OUTPUT
    --
    -- <ERRORS>                 ++*, ***..%%%, <PROFILE>
    --
    -- <IGNORE>                 ~::::..$$$. <PROFILE>
    --
    -- <NIL>                    Empty, ignores user profile.
    --                          Quit, ~::::..$$$, Prefix=>, Width=>77, USE_OUTPUT
    --
    -- <PROFILE>                Values set by job/session profile.
    --
    -- <QUIET>                  Equivalent of <NIL>
    --
    -- <PROGRESS>               +++, ++*, ***..%%%, <PROFILE>
    --
    -- <RAISE_EXCEPTION>        PROPAGATE, <PROFILE>
    --
    -- <SESSION_PROFILE>        Use session profile rather than job.
    --
```

```
-- <WARN>                    ++*..%%%, <PROFILE>
--
-- <VERBOSE>                 :::, ~???, ---..$$$, <PROFILE>

function Get return String;
-- Profile for the current job as a form parameter

function Get_Default return String;
-- Profile for the Session (which is the default for a job that
-- does not specify one) as a form parameter

function Image (Profile : Response_Profile := Standard.Profile.Get)
               return String;
function Value (Image : String := Profile.Get) return Response_Profile;

procedure Convert (Image    :        String;
                   Response : out    Response_Profile;
                   Status   : in out Simple_Status.Condition);

-- Convert between form parameter representation of a profile and the
-- internal representation.  The Value function ignores invalid options
-- in the form parameter.

procedure Set (Profile : String; Status : in out Simple_Status.Condition);

-- Set profile for rest of current job; An error Status is returned and
-- the profile is not changed if the profile string is invalid.

procedure Set_Default (Profile :        String;
                       Status  : in out Simple_Status.Condition);

-- Set profile for session; this is the value used for the job
-- response profile if none is otherwise specified.

procedure Get_Cached_Resolution
             (Name            :     String;
              The_Declaration : out Directory.Declaration;
              The_Object      : out Directory.Object;
              The_Version     : out Directory.Version;
              Status          : out Directory.Naming.Name_Status);

-- Retrieve the resolution of the Name as cached at job initiation. Only
-- resolution of <IMAGE>, <CURSOR>, <REGION>, and <SELECTION> are cached.

function Cached_Selected_Text return String;

-- Retrieve the Selected text at job initiation.
```





```
    -- REMOTE_PASSWORDS:

    subtype Remote_Passwords_Type is Directory.Object;
    function  Remote_Passwords (Response : Response_Profile := Profile.Get)
                          return Remote_Passwords_Type;
    function  Default_Remote_Passwords return Remote_Passwords_Type;
    procedure Set_Remote_Passwords (Passwords : Remote_Passwords_Type);
    procedure Set_Default_Remote_Passwords (Passwords : Remote_Passwords_Type);

    -- REMOTE_SESSIONS:

    subtype Remote_Sessions_Type is Directory.Object;  -- text file
    function  Remote_Sessions (Response : Response_Profile := Profile.Get)
                         return Remote_Sessions_Type;
    function  Default_Remote_Sessions return Remote_Sessions_Type;
    procedure Set_Remote_Sessions (Sessions : Remote_Sessions_Type);
    procedure Set_Default_Remote_Sessions (Sessions : Remote_Sessions_Type);

end Profile;
```

```
package Program_Library_Maintenance is

    procedure Verify (Worlds   : String := "<IMAGE>$$";
                      Options  : String := "";
                      Response : String := "<PROFILE>");

    function Verify (Worlds   : String := "<IMAGE>$$";
                     Options  : String := "";
                     Response : String := "<QUIET>") return Boolean;

    -- Check the consistency of the program libraries for the given
    -- set of worlds.  Each library is checked for internal consistency
    -- as well as for consistency with the Ada units in the world.
    -- (There are no valid Options at present, reserved for the future.)


    procedure Check_Code_Segment_Reference_Counts
                 (Worlds : String := "!??'C(WORLD)'T(R1000)";
                  Increase_Reference_Counts_That_Are_Too_Small :
                     Boolean := False;
                  Decrease_Reference_Counts_That_Are_Too_Large :
                     Boolean := False;
                  Options : String := "";
                  Response : String := "<PROFILE>");

    -- Check, and optionally correct, the reference counts of the R1000
    -- code segments referenced in the given set of worlds. Note that if a
    -- code segment is referenced from within the given set of worlds
    -- and also from outside the given set of worlds, then an
    -- incorrect error message complaining the the reference count
    -- is too large will be generated. Thus, it is safest to run this
    -- procedure over all R1000 worlds at once.
    -- (There are no valid Options at present, reserved for the future.)


    procedure Find_Unreferenced_Code_Segments
                 (Destroy_Unreferenced_Code_Segments : Boolean := False;
                  Options                            : String  := "";
                  Response                           : String  := "<PROFILE>");

    -- Exmaines all of the R1000 code segments on the machine and lists
    -- those that are not referenced by any program library.  Note that
    -- there is always a small pool of such code segments that are used
    -- for commands.
    -- (There are no valid Options at present, reserved for the future.)
```





```
    procedure Display_Referencers (R1000_Code_Segment_Name : Natural;
                                   Worlds : String := "$$";
                                   Options : String := "";
                                   Response : String := "<PROFILE>");

    -- Given the name of an R1000 code segment and a set of worlds (any or
    -- all of which may be code views or contain loaded main programs),
    -- displays the set of units in the given world which reference the
    -- given code segment. This set of units can include "units" within
    -- code views (which do not exist as Ada objects), loaded main programs,
    -- Delta1 code databases, and ordinary Ada units.
    -- (There are no valid Options at present, reserved for the future.)


    function Ada_Object_Instance_Name
                (R1000_Code_Segment_Name : Natural) return Natural;

    -- Given the name of an R1000 code segment, returns the instance
    -- component of the object_id of the corresponding Ada object.
    -- If a code view or a loaded main program is copied from one machine
    -- to another, this can be useful in mapping a code segment address
    -- on the second machine back to the corresponding Ada source location
    -- on the first machine.


    procedure Build (Worlds : String := "<SELECTION>";
                     Atomic : Boolean := False;
                     Preserve_Unreconstructable_Libraries : Boolean := True;
                     Options : String := "";
                     Response : String := "<PROFILE>");

    -- Build or rebuild the program library for one or more worlds.
    --     Worlds -
    --         The set of worlds for which program libraries are to be built.
    --     Atomic -
    --         Specifies whether all program libraries are to built on a
    --         single action, so that either the entire operation will
    --         succeed for every world or it will have no effect.
    --     Preserve_Unreconstructable_Libraries -
    --         If the program library for a code view or a world containing
    --         a loaded main program is rebuilt, then the code associated
    --         with the code view or loaded main program will be lost
    --         unless the Delta1_Code_View_Compatibility switch was set
    --         when the code view or loaded main program was created.
    --         The BUILD procedure will skip over such worlds unless
    --         the PRESERVE_UNRECONSTRUCTABLE_LIBRARIES is set to FALSE.
    -- (There are no valid Options at present, reserved for the future.)
```

```
    procedure Show_Dependent_Main_Programs (Code_Views : String := "<IMAGE>$$";
                                            Options    : String := "";
                                            Response   : String := "<PROFILE>");

    -- (There are no valid Options at present, reserved for the future.)


    function Dependent_Main_Programs
                (Code_Views : String := "<IMAGE>$$";
                 Options    : String := "";
                 Response   : String := "<QUIET>") return String;

    -- Given a code view, determine the set of coded main programs that
    -- would have to be demoted to installed before the given code view
    -- could be destroyed (in other words, the set of coded main programs
    -- whose closure includes units out of the given code view).
    -- (There are no valid Options at present, reserved for the future.)


    procedure Destroy_Delta1_Code_Databases
                (Code_Views_Or_Loaded_Main_Programs : String := "<IMAGE>";
                 Options                            : String := "";
                 Response                           : String := "<PROFILE>");

    -- Destroy the Delta1 code database associated with a code view or
    -- with a loaded main program.  This will not affect the ability to
    -- use the code view or loaded main program on a Delta2 machine,
    -- but it will make it impossible to move the code to any Delta1
    -- machines (using Archive.Copy or Archive.Save/Restore).  Code
    -- views and loaded main programs will require considerably less
    -- space if the Delta1 code databases are destroyed.
    -- (There are no valid Options at present, reserved for the future.)


    procedure Edit (For_World : String := "<IMAGE>$$");

    -- Display the contents of the program library for a given world.
    -- May be used to display the units in a code view; but this editor
    -- is mainly designed to be used as a debugging tool.

end Program_Library_Maintenance;
```





```
package Program_Library_Object_Editor is

    procedure Edit (The_Library : String  := "<CURSOR>";
                    In_Place    : Boolean := False);
    -- Invokes the Program_Library_Object_Editor to display the
    -- directory of units contained in the program library.  The
    -- object to be edited may be a program_library object, or a code
    -- view (in the latter case, the associated program library
    -- is displayed).

    procedure Ada_Edit (The_Unit    : String  := "<IMAGE>";
                        The_Library : String  := "<IMAGE>";
                        In_Place    : Boolean := False);
    -- Invokes the Ada_Object_Editor to display the ada source
    -- from which a unit in a program library was
    -- produced.  The original (coded) unit must still exist
    -- on this machine for the command to succeed.

end Program_Library_Object_Editor;
```

```
generic
    type Element is private;

    pragma Must_Be_Constrained (Yes => Element);

package Queue_Generic is

    type Queue is private;

    procedure Initialize (Q : out Queue);

    function  Is_Empty   (Q : Queue) return Boolean;
    procedure Make_Empty (Q : in out Queue);

    procedure Copy (Target : in out Queue; Source : Queue);

    procedure Add    (Q : in out Queue; X : Element);
    procedure Delete (Q : in out Queue);
    function  First  (Q : Queue) return Element;
    -- on calls to delete and first, not is_empty(q) is assumed
    -- constraint error will be raises is is_empty(q)

    type Iterator is private;

    procedure Init  (Iter : out Iterator; Q : Queue);
    procedure Next  (Iter : in out Iterator);
    function  Value (Iter : Iterator) return Element;
    function  Done  (Iter : Iterator) return Boolean;

    -------------------------------------------------------
    -- Implementation Notes and Non-Standard Operations --
    -------------------------------------------------------

    -- variables of type queue are initially empty
    --    therefore, the call to initialize is optional

    -- := and = are meaningless
    --   := implies sharing (introduces an alias) for sub-structures

    -- garbage may be generated

private

    type Node;
    type Pointer is access Node;
```





```
    type Node is
        record
            Value : Element;
            Link  : Pointer;
        end record;

    type Queue is
        record
            Head : Pointer;
            Tail : Pointer;
        end record;

    type Iterator is new Queue;

end Queue_Generic;
```

```
with System;

package Random is

    type Handle is private;

    function Float_Value (The_Handle : Handle) return Float;
    -- Will return values in the range [0.0 .. 1.0), but note that type
    -- conversion may cause rounding.

    function Natural_Value (The_Handle : Handle; Max : Natural) return Natural;
    -- Return a value in the range 0 .. Max with uniform distribution.

    generic
        type Result_Type is (<>);
    function Enumeration_Value_Generic (The_Handle : Handle) return Result_Type;
    -- Return a uniform distribution of enumeration literals.

    function String_Value (The_Handle : Handle;
                           Max_Length : Natural;
                           Min_Length : Natural := 0;
                           Anchored   : Boolean := False) return String;
    -- Return a random string.  If Anchored is true, the lower bound
    -- will always be 1.

    ----------------------------------------------------------------------

    subtype Seed_Type is Integer;

    function Generate_Seed return Seed_Type;
    -- Construct a Seed based on the current time of day.

    procedure Initialize (The_Handle : out Handle;
                          Seed       : Seed_Type := Generate_Seed;
                          Storage    : System.Segment := System.Null_Segment);
    -- Start the generator.

    function Initial_Seed (The_Handle : Handle) return Seed_Type;
    -- Return the seed used to initialize this handle.

    function Calls (The_Handle : Handle) return Natural;
    -- Return the number of calls to this handle.

end Random;
```





```
with Transport;
with Transport_Defs;

generic
    type Service_Type is limited private;
    with procedure Serve (Service    : in out Service_Type;
                          Connection :        Transport.Connection_Id) is <>;

procedure Transport_Server_Proc (Service      : in out Service_Type;
                                 Connection   : in out Transport.Connection_Id;
                                 Network      : Transport_Defs.Network_Name;
                                 Local_Socket : Transport_Defs.Socket_Id);

-- Wait for incoming calls on the given Connection, Network and
-- Local_Socket.  For each call, call Serve (Service, Connection).
-- Repeat until the socket is in use (presumably by another server).
-- If Serve raises an exception, call Transport.Close (Connection)
-- and then re-raise the exception.
```

```
with Transport_Defs;

package Queue_Service is

    procedure Start (Network : Transport_Defs.Network_Name := "TCP/IP");
    procedure Serve (Network : Transport_Defs.Network_Name := "TCP/IP");

end Queue_Service;
```

```
package Script is

    procedure Pretty_Print (Script_File : String; Command_File : String);

end Script;
```





```
generic
    type Element is private;
    -- must be a pure value
    -- ie. no initialization or finalization is necessary
    -- = and := are equality and copy

    pragma Must_Be_Constrained (Yes => Element);

package Set_Generic is

    type Set is private;

    procedure Initialize (S : out Set);

    function  Is_Empty   (S : Set) return Boolean;
    procedure Make_Empty (S : in out Set);

    procedure Copy (Target : in out Set; Source : Set);

    function Is_Member (S : Set; X : Element) return Boolean;

    procedure Add    (S : in out Set; X : Element);
    procedure Delete (S : in out Set; X : Element);
    -- X is (is not) in S then the operation add (delete) is a no op.

    type Iterator is private;

    procedure Init  (Iter : out Iterator; S : Set);
    procedure Next  (Iter : in out Iterator);
    function  Value (Iter : Iterator) return Element;
    function  Done  (Iter : Iterator) return Boolean;

    -----------------------------------------------------
    -- Implementation Notes and Non-Standard Operations --
    -----------------------------------------------------

    -- variables of type set are initially empty
    --    therefore, the call to initialize is optional
    --   initialize does make the set empty

    -- := and = operate on references
    --   := implies sharing (introduces an alias)
    --   = means is the same set, not the same value of type set

    -- garbage may be generated
```

```
    -- Concurrency Properties
    -- any number of read operations (is_empty,is_member) can procede
    -- concurrently with one write operations (add/delete/make_empty)

private

    type Node is
        record
            Value : Element;
            Link  : Set;
        end record;

    type Set is access Node;

    type Iterator is new Set;

end Set_Generic;
```





```
package Simple_Status is

    -- Error status reporting package

    -- A simple_status.condition can be used to return error information from
    -- procedure calls.  They are relatively large and should always
    -- be passed in out (by convention to avoid copies).

    -- A Condition consists of a Condition_Name and a Message.
    -- The Condition_Name indicates the type of error (if any) and how
    -- how serious the error is (or if completion was
    -- successful).  The Message provides additional information about
    -- the error.

    -- In simple applications, A Condition_Name alone can be used to
    -- indicate status.

    -- By convention, condition names in an application should be
    -- standardized so that error conditions can be tested
    -- programmaticly.


    type Condition_Name  is private;  -- A short name for the error type
    type Condition_Class is
       (Normal,   -- operation completed normally
        Warning,  -- operation completed, but something unexpected happened
        Problem,  -- operation did not complete, but no harm done
        Fatal);   -- operation did not complete.  Proceeding is dangerous.
    type Condition       is private;  -- Contains the above plus a message
    -- Conditions are self-initializing to
    -- severity Normal and null names


    procedure Initialize (Status : in out Condition);
    -- The empty condition has null name and severity normal
    -- a declared condition is initialized;  This procedure will set the
    -- Condition to be Normal (ie, successful).


    function Name (Error_Type : Condition_Name) return String;
    function Name (Status : Condition)          return String;
    -- get the human-readable name of this Condition_Name (Condition)
```

```
function Severity (Error_Type : Condition_Name) return Condition_Class;
function Severity (Status : Condition)          return Condition_Class;


function Error_Type (Status : Condition) return Condition_Name;
-- provide the Condition_Name on which a Condition is built


function Error (Error_Type : Condition_Name;
                Level      : Condition_Class := Warning) return Boolean;
function Error (Status : Condition; Level : Condition_Class := Warning)
               return Boolean;
-- True <=> Severity (Error_Type/Status) >= Level;
-- usage:
--    Do_Something (Status);
--    if Simple_Status.Error (Status) then
--           ... Put (Display_Message (Status));


function Display_Message (Status : Condition) return String;
-- given a condition that indicates and error, this function returns
-- a string suitable for display to users.  It includes the
-- string form of the condition name and any additional problem-
-- specific information recorded in the condition.


function Message (Status : Condition) return String;
-- return just the message part of the Condition.


procedure Create_Condition (Status     : in out Condition;
                            Error_Type :        String;
                            Message    :        String          := "";
                            Severity   :        Condition_Class := Problem);
procedure Create_Condition (Status     : in out Condition;
                            Error_Type :        Condition_Name;
                            Message    :        String := "");
-- Create a new error condition.  The Error_Type is intended to
-- specify the class of error (limited to 63 characters), generally
-- in a few words (eg, "Illegal name").  Message is intended to
-- supplement the error_type with more specific information
-- (eg, ": '#' is an illegal character").  Function Display_Message
-- would then return "Illegal name: '#' is an illegal character".


function Create_Condition_Name
           (Error_Type : String; Severity : Condition_Class := Problem)
           return Condition_Name;
```





```
    function Equal (Status : Condition; Error_Type : String) return Boolean;
    function Equal (Status : Condition; Error_Type : Condition_Name)
                   return Boolean;
    function Equal (Status : Condition_Name; Error_Type : String)
                   return Boolean;
    function Equal (Status : Condition_Name; Error_Type : Condition_Name)
                   return Boolean;
    -- return true if the error_type string of Status is equal to the
    -- right error_type string (the second parameter).  The severity
    -- does not participate in the comparison.  The strings must
    -- match exactly (except for index range).  Sample usage:
    --   Directory.Open(File, Status);
    --   if SS.Equal (Status, "Nonexistent file") then ...
    --   elsif SS.Equal (Status, "Internal error") then ...
    --   etc.
    -- The strings in the example should be constants, of course.

end Simple_Status;
```

```
generic
    type Element is private;

    pragma Must_Be_Constrained (Yes => Element);

package Stack_Generic is

    type Stack is private;
    Empty_Stack : constant Stack;
    -- It is expected that a declared stack is initialized to Empty_Stack

    procedure Make_Empty (S : in out Stack);
    procedure Pop        (S : in out Stack);
    procedure Push       (X : Element; S : in out Stack);
    function  Empty      (S : Stack) return Boolean;
    function  Top        (S : Stack) return Element;

    procedure Copy (Target : in out Stack; Source : Stack);

    Underflow : exception;

    type Iterator is private;

    procedure Init (Iter : out Iterator; S : Stack);
    procedure Next (Iter : in out Iterator);

    function Value (Iter : Iterator) return Element;
    function Done  (Iter : Iterator) return Boolean;

private
    type Stack_Node;
    type Stack is access Stack_Node;
    Empty_Stack : constant Stack := null;
    type Iterator is new Stack;

end Stack_Generic;
```





```
generic

    Size : Integer;

    type Range_Type is private;
    -- Range_Type is a pure value
    -- no initialization or finalization of values of range_type is
    -- necessary
    -- = and := can be used for equality and copy

    Ignore_Case : Boolean := True;

    pragma Must_Be_Constrained (Yes => Range_Type);

package String_Map_Generic is

    type Map is private;

    function Eval (The_Map : Map; D : String) return Range_Type;

    procedure Find (The_Map :        Map;
                    D       :        String;
                    R       : in out Range_Type;
                    Success : out    Boolean);

    procedure Define   (The_Map         : in out Map;
                        D               :        String;
                        R               :        Range_Type;
                        Trap_Multiples  :        Boolean := False);
    procedure Undefine (The_Map : in out Map; D : String);

    procedure Initialize (The_Map : out Map);
    function  Is_Empty   (The_Map : Map) return Boolean;
    procedure Make_Empty (The_Map : in out Map);

    procedure Copy (Target : in out Map; Source : Map);

    type Iterator is private;

    procedure Init  (Iter : out Iterator; The_Map : Map);
    procedure Next  (Iter : in out Iterator);
    function  Value (Iter : Iterator) return String;
    function  Done  (Iter : Iterator) return Boolean;

    Undefined : exception;
    -- raised by eval if the domain value in not in the map
```

```
    Multiply_Defined : exception;
    -- raised by define if the domain value is already defined and
    -- the trap_multiples flag has been specified (ie. is true)

    function Nil                        return Map;
    function Is_Nil (The_Map : Map) return Boolean;

    function Cardinality (The_Map : Map) return Natural;

    -------------------------------------------------------
    -- Implementation Notes and Non-Standard Operations --
    -------------------------------------------------------

    -- := and = operate on references
    --    := implies sharing (introduces an alias)
    --    = means is the same set, not the same value of type set
    -- Initializing a map also makes it empty
    -- Accessing an uninitialized map will raise CONSTRAINT_ERROR.

    -- garbage may be generated

private

    subtype Index is Natural range 0 .. Size - 1;
    type    Node (Size : Natural);
    type    Set   is access Node;
    type    Table is array (Index) of Set;
    type Map_Data is
        record
            Bucket : Table;
            Size   : Integer := 0;
        end record;
    type Map is access Map_Data;
    type Iterator is
        record
            The_Map     : Map;
            Index_Value : Index;
            Set_Iter    : Set;
            Done        : Boolean;
        end record;
    type Node (Size : Natural) is
        record
            Link  : Set;
            Value : Range_Type;
            Name  : String (1 .. Size);
        end record;

end String_Map_Generic;
```





```
package String_Table is

    type Item is private;

    type Table is private;

    Table_Full : exception;

    -- create a table for unique strings
    function New_Table (Minimum_Table_Size : Natural := 127) return Table;

    function Nil return Item;

    -- return unique item in table, ignore_case => upper_case storage
    function Unique (Source      : String;
                     In_Table    : Table;
                     Ignore_Case : Boolean := True) return Item;

    -- return item if present, otherwise Nil
    function Find (Source      : String;
                   In_Table    : Table;
                   Ignore_Case : Boolean := True) return Item;

    -- return an item without entering in table
    function Allocate (Source : String; In_Table : Table) return Item;

    -- compare strings for identity, then same contents
    function Equal (L, R : Item) return Boolean;

    -- representation of string, suitable for hashing
    function Unique_Index (U : Item) return Integer;

    -- value of character or entire string
    function Char_At (Source : Item; At_Pos : Natural) return Character;
    function Image   (Source : Item)                   return String;
    function Length  (Source : Item)                   return Natural;
    function Is_Nil  (Source : Item)                   return Boolean;

    type Iterator is private;

    procedure Init  (Iter : out Iterator; The_Table : Table);
    procedure Next  (Iter : in out Iterator);
    function  Value (Iter : Iterator) return Item;
    function  Done  (Iter : Iterator) return Boolean;

end String_Table;
```

```
package String_Utilities is

    function Hash_String (S : String) return Integer;

    procedure Upper_Case (C : in out Character);
    procedure Lower_Case (C : in out Character);

    function Upper_Case (C : Character) return Character;
    function Lower_Case (C : Character) return Character;

    procedure Upper_Case (S : in out String);
    procedure Lower_Case (S : in out String);

    -- string returned has same 'First and 'Last as S
    function Upper_Case (S : String) return String;
    function Lower_Case (S : String) return String;

    function Number_To_String (Value   : Integer;
                               Base    : Natural   := 10;
                               Width   : Natural   := 0;
                               Leading : Character := ' ') return String;
    function Number_To_String (Value   : Long_Integer;
                               Base    : Natural   := 10;
                               Width   : Natural   := 0;
                               Leading : Character := ' ') return String;

    procedure String_To_Number (Source :     String;
                                Target : out Integer;
                                Worked : out Boolean;
                                Base   :     Natural := 10);
    procedure String_To_Number (Source :     String;
                                Target : out Long_Integer;
                                Worked : out Boolean;
                                Base   :     Natural := 10);

    function Strip_Leading
                (From : String; Filler : Character := ' ')    return String;
    function Strip_Trailing
                (From : String; Filler : Character := ' ')    return String;
    function Strip (From : String; Filler : Character := ' ') return String;

    -- Searches and compares
    -- Locate returns the index value in Within if found, 0 otherwise
    function Locate (Fragment    : String;
                     Within      : String;
                     Ignore_Case : Boolean := True) return Natural;
```





```
    function Locate (Fragment    : Character;
                     Within      : String;
                     Ignore_Case : Boolean := True) return Natural;

    function Reverse_Locate (Fragment    : String;
                             Within      : String;
                             Ignore_Case : Boolean := True) return Natural;

    function Reverse_Locate (Fragment    : Character;
                             Within      : String;
                             Ignore_Case : Boolean := True) return Natural;

    function Equal (Str1 : String; Str2 : String; Ignore_Case : Boolean := True)
              return Boolean;

    function Less_Than
            (Str1 : String; Str2 : String; Ignore_Case : Boolean := True)
            return Boolean;

    function Greater_Than
            (Str1 : String; Str2 : String; Ignore_Case : Boolean := True)
            return Boolean;

    procedure Capitalize (S : in out String);
    function  Capitalize (S : String) return String;

end String_Utilities;
```

```
with Calendar;

package Accounting_Information is

    type Iterator is private;

    type Entry_Kind is (Session, -- Entry when a user logs out
                        Job);    -- Entry for a user Job (after logout)

-- Iterator Routines

    function Initialize (Accounting_Library : String := "!Machine.Accounting")
                        return Iterator;
    -- Utilizes filenames of the form <Accounting_Library>.Activity_@
    --

    procedure Next (Iter : Iterator);
    function  Done (Iter : Iterator) return Boolean;

-- Information obtainable for each entry
    function Kind           (Iter : Iterator) return Entry_Kind;
    function Username       (Iter : Iterator) return String;
    function Session        (Iter : Iterator) return String;
    function Login_Time     (Iter : Iterator) return Calendar.Time;
    function Logoff_Time    (Iter : Iterator) return Calendar.Time;
    function Login_Duration (Iter : Iterator) return Duration;
    function Cpu_Used       (Iter : Iterator) return Duration;
    function Total_Disk_Io  (Iter : Iterator) return Natural;
    function Total_Jobs_Run (Iter : Iterator) return Natural;
    function Image          (Iter : Iterator) return String;

private
    type Handle;
    type Iterator is access Handle;

end Accounting_Information;
```





```
procedure Accounting_Report
             (From_Date            : String := "";
              To_Date              : String := "";
              For_User             : String := "@";
              Accounting_Directory : String := "!Machine.Accounting");
--
-- This procedure will generate an accounting summary from the accounting files
-- located in !Machine.Accounting.  Parameters are:
--
--      From_Date, To_Date  : String values acceptable by Time_Utilities.
--                            This value MUST be the date only (no time value).
--      For_User            : Specific users to display accounting information
--                            for.  The default "@" specifies ALL users.
--
-- Output will go to the standard output window.
-- For example:
--
--     Accounting_Summary (From_Date => "87/11/04",
--                         To_Date => "87/11/18",
--                         For_User => "[smp,tom,cbh,phil]");
--
-- would generate the following output:
--
-- Accounting Summary for Period  4-NOV-87 to 18-NOV-87.
-- Total 'Work' Time for Period :  80 hours ( 10 days )
--
--     NAME      LOGINS  LOGIN_TIME  CPU_TIME  DISK_IO  JOBS_RUN  P_LOG  L_CPU
-- ===========   ======  ==========  ========  =======  ========  =====  =====
-- CBH               17     7/11:12  12:10:22   842020      3191    224    152
-- PHIL              28     5/01:57   6:06:42   393996      2439    152     76
-- SMP                9    22:44:02  9:42.306    19326       223     28      2
-- TOM                2     2:26:52  3:12.889     7513        29      3      1
--                <Avg>       <Avg>     <Avg>    <Avg>     <Avg>  <Avg>  <Avg>
-- Total ( 4 )       14     3/09:35   4:37:30   315713      1470    102     58
--
-- where 'Total Work Hours for Period' is calculated by multiplying the
-- number of week days (MON-FRI) in the period by 8 hours, and :
--     NAME            : User account name.
--     LOGINS          : Number of times user has logged in.
--     LOGIN_TIME      : Total amount of time user was logged in.
--     CPU_TIME        : Total amount of CPU time user has consumed.
--     DISK_IO         : Total number of disk I/O requests for user.
--     JOBS_RUN        : Total number of jobs run by user.
--     P_LOG           : Percent user was logged in of 'Total Work Hours'
--     L_CPU           : 'Load' CPU, weights user consumption of available
--                       CPU time from 'Total Work Hours'.
-- The summary line at the bottom totals the amounts in each column.
```

```
with Calendar;
with Time_Utilities;
with Simple_Status;

package Log_Reader is

-- Abstraction for reading the system error log.
--
-- Provides an Iterator that automatically crosses log files and also
-- extends into the current active error log.  Thus, log messages
-- can be read right up to the last one issued by the system.
--
-- The date part of the date/time for an entry is impossible to know for
-- the first few entries in the first available log file.  In this case,
-- 1/1/1901 is returned (or 1/2/1901, if midnight occurs in these few
-- date unknown entries).
--
-- Continuation lines are automatically incorporated into each entry so that
-- each call to Next moves to a complete new entry.  Continuation lines
-- are read as part of the Message field and are preceded by ASCII.lf
-- characters.
--
-- NOTE: This package makes an assumption about the Error Log files
-- that it processes.  It assumes that the log file names are in the
-- form of Log_YY_MM_DD_At_HH_MM_SS, this is the form that the error log
-- daemon creates file names as in the directory !Machine.Error_Logs,
-- and that the time embedded in the file name accurately represents the
-- times of the entries in the file.

    procedure Load_Logs (From_Directory : String := "!machine.error_logs");

    procedure Load_Logs (Start_Time     : Calendar.Time;
                         Stop_Time      : Calendar.Time;
                         From_Directory : String := "!machine.error_logs");

    -- Initialize the module.  Must be called before any other operations.
    -- Builds map of log files in the specified directory.  Specifying a
    -- start and stop time will reduce the number of log files examined to
    -- those created during the given period.  Otherwise, all log files in
    -- the given directory are examined.
    --
    -- NOTE: The Start_Time and Stop_Time entered by this procedure are
    -- used as the start date and stop date.  This means that the time
    -- part of the Calendar.Time variable is ignored.  This package will
    -- iterate over the log entries from 00:00:01 of the start date to
    -- 23:59:59 of the stop date (or as much as is available in the log
    -- files and the current day's log messages).
```





```
    type Iterator is private;
    procedure Initialize (I      : out    Iterator;
                          Status : in out Simple_Status.Condition);

    procedure Next (I      : in out Iterator;
                    Status : in out Simple_Status.Condition);

    function Done              (I : Iterator) return Boolean;
    function Current_Entry     (I : Iterator) return String;
    function Current_Time      (I : Iterator) return Time_Utilities.Time;
    function Current_Severity  (I : Iterator) return String;
    function Current_Client    (I : Iterator) return String;
    function Current_Condition (I : Iterator) return String;
    function Current_Message   (I : Iterator) return String;
    function Current_File      (I : Iterator) return String;

    function Get_Time (Log_Entry : String) return Time_Utilities.Time;
    -- Note that only the HH:MM:SS part of the time is set by this operation
    function Severity  (Log_Entry : String) return String;
    function Client    (Log_Entry : String) return String;
    function Condition (Log_Entry : String) return String;
    function Message   (Log_Entry : String) return String;

    function Number_Of_Log_Files return Natural;
    -- Returns number of log files that exist.  Defined only after call
    -- to Load_Logs.

private

    type Iterator_Data;
    type Iterator is access Iterator_Data;

end Log_Reader;
```

```
package Outage_Information is

    procedure Update_Log_From_Dfs;
    -- Load the crash cause of the most recent outage from the DFS
    -- error log and store into the system error log as a
    -- crash explanation.

    type Iterator is private;
    procedure Init  (I : out Iterator);
    procedure Next  (I : in out Iterator);
    function  Done  (I : Iterator) return Boolean;
    function  Value (I : Iterator) return String;

    -- Iterate over the dfs error log.

private
    type Iterator_Data;
    type Iterator is access Iterator_Data;

end Outage_Information;
```





```
procedure Availability_Report
             (Starting_Date          : String  := "<MINUS_30_DAYS>";
              Ending_Date            : String  := "<TODAY>";
              Display_Contract_Hours : Boolean := True;
              Log_Directory          : String  := "!Machine.Error_Logs");

-- This procedure is used to scan the error logs during the given period and
-- generate a System Availability Report.  This report displays system downtime
-- by shutdown code for the period in 2 groups, Rational specific and Customer
-- specific.  Customer specific shutdown codes are 'COPS', 'CSD', and 'Other'.
-- All other codes are Rational specific.  In addition, dowtime is grouped
-- according to Contract and Non-Contract hours.  The default contract hours
-- are Monday thru Friday from 8:00 to 18:00.  It is possible to modify the
-- contract hours by creating a file in !Machine called Contract_Hours, and
-- entering the new contract hours as a valid option string(s):
--
--  Day  => Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
--  Time => 0:00..24:00
--
-- Format:
--
--   Day[..Day] => (Time..Time)
--
-- If using the range form "Day..Day", the precedence of Days must follow
-- as defined above, e.g. Monday..Sunday.
--
-- For example, the contents of the Contract_Hours file:
--
-- Mon..Fri = (0:00..24:00)
-- Sat..Sun => (8:00..17:00)
--
-- would define contract hours to be:
--     Monday thru Friday, 24 hours a day
--     Saturday thru Sunday, 9 hours a day from 8am to 5pm
--
-- A sample output:
--
-- System  Availability Report from  2-FEB-88 to  3-MAR-88
--
-- Rational, Mt. View, CA [Gator:894757]
--
-- Total Contract Hours     : 397:59:56
-- Total Non Contract Hours : 322:00:04
--                            =========
-- Total Hours              : 720:00:00
--
--                  Contract     Non Con       Total
```

```
--
--         PMH :      00:00      00:00      00:00
--         PMS :      00:00      00:00      00:00
--         HWC :      54:05      00:00      54:05
--         SWC :      00:00      00:00      00:00
--       CRASH :      00:00      00:00      00:00
--        HANG :      00:00      00:00      00:00
--       MAINT :      00:00      00:00      00:00
--     RELEASE :      00:00      00:00      00:00
--         UCC :      00:00      00:00      00:00
--    ========   ========   ========   ========
--       Total :      54:05      00:00      54:05
--   Available :      99.8%     100.0%      99.9%
--
--        COPS :      13:58    1:23:02    1:37:00
--         CSD :      00:00      00:00      00:00
--       OTHER :      05:15      00:00      05:15
--    ========   ========   ========   ========
--       Total :      19:13    1:23:02    1:42:15
--   Available :      99.9%      99.6%      99.8%
--
--   Grand Total    1:13:18    1:23:02    1:42:15
--      Uptime :      99.7%      99.6%      99.6%
--
--
-- Starting_Date : Default of <MINUS_30_DAYS> uses a date 30 days previous from
--                 when this procedure is executed.  If not the default, a valid
--                 date must be entered.
-- Ending_Date   : Default of <TODAY> uses the current date of execution.  If
--                 not the default, a valid date must be entered.
-- Display_Contract_Hours :
--                 If true, displays the contract hours used in
--                 calculating the system availability.
```





```
procedure Display_Large_Error_Logs
             (Limit          : Long_Integer := 60_000;
              Start_Time     : String       := "<MINUS_7_DAYS>";
              Stop_Time      : String       := "<TODAY>";
              Log_Directory  : String       := "!Machine.Error_Logs");

-- This routine will list any system error log files located in
-- !Machine.Error_Logs that are larger than the limit specified.  Only
-- error logs created between the specified start and stop times are
-- considered.


procedure Error_Summary
             (Starting_Date          : String  := "<MINUS_7_DAYS>";
              Ending_Date            : String  := "<TODAY>";
              Log_Filename           : String  := "<CURRENT_OUTPUT>";
              Save_Error_Information : Boolean := False;
              Log_Directory          : String  := "!Machine.Error_Logs");

-- This procedure will generate a system error summary report for the
-- period between Starting_Date and Ending_Date, and, if
-- Save_Error_Information is true, will save all log entries dealing with
-- system errors to !Machine.Error_Logs.Error_Summary_<Date/Time>.  The
-- Error Summary totals are written to the Log_Filename, and if left as the
-- default will write to the "Current Output".
```

```
procedure Error_Summary_Server (Last_Summary_Filename : String := "<DEFAULT>");

-- This procedure is intended to be executed from the operator/system manager's
-- Login procedure to generate an error report.  This report is a summary of
-- system errors such as Disk and ECC.  If a large number of these errors are
-- detected, Rational should be notified.
--
-- The file Last_Summary_Filename is maintained to contain the time this
-- routine was last executed, and the summary information from that
-- execution. Each time this routine is executed, that file is examined and
-- a report is generated for the period from the last execution to the
-- current time, thus only displaying errors that have occurred since the
-- last report.
--
-- If the default value is used for Last_Summary_Filename, then a file in
-- the users home library called Last_Error_Summary_Time is created/used.
--
-- For comparison, the error report displayed by the last execution is
-- displayed after the current error report has been displayed.
--
-- In the event the Last_Error_Summary_Time file does not exist, a default
-- of -7 days from the current time is used as the starting date.
```





```
procedure Pm_Report (Starting_Date      : String  := "<MINUS_90_DAYS>";
                     Ending_Date        : String  := "<TODAY>";
                     Delete_Log_Files   : Boolean := False;
                     Print_Report       : Boolean := True;
                     Report_Filename    : String  := "<DEFAULT>";
                     Log_Directory      : String  := "!Machine.Error_Logs";
                     Accounting_Library : String  := "!Machine.Accounting");

-- This procedure will generate a Preventive Maintenance (PM) Report.  This
-- report provides useful system information for analysis and history for
-- later reference.  The period defined by Starting_Date to Ending_Date is
-- used when summarizing the log file entries.  If Delete_Log_Files is
-- TRUE, then those log files that fall between Starting_Date and
-- Ending_Date are deleted from !Machine.Error_Logs.  If Print_Report is
-- TRUE, then the report is printed via the system default print queue.
-- The default Report_Filename is a file created in the current context,
-- called PM_Report_<Date/Time>.  This procedure is designed to execute in
-- as a background job, and will cause this to occur immediately after it
-- begins to execute.
```

```
procedure Show_Error_Log (Start : Natural := 0; Count : Natural := 30);

-- Start = 0 => show end of log
```

```
with Simple_Status;
with Time_Utilities;
with Bounded_String;
with System; -- cmg 03/06/92
with Job_Segment; -- cmg 03/06/92

package System_Information is

-- Interfaces to extract information used to produce System_Report output.

    procedure Generate (Start_Time    :        String := "";
                        End_Time      :        String := "";
                        Log_Directory :        String := "!Machine.Error_Logs";
                        Log_Time      : out    Duration;
                        Status        : in out Simple_Status.Condition);

    -- Must be called prior to using the following operations.  They, then
    -- can be used to read the reduced data.  Log_Time indicates the
    -- actual duration between the first and last entries used in this
    -- report.

    -- Each iterator has a type and returns certain information.
    -- General paradign for each is:
    --
    --      I : xxx_Iterator;
    --      Info : xxx_Information;
    --
    --      Initialize (I);
    --      while not Done (I) loop
    --          Info := Value (I);
    --          -- Do something with Info
    --          Next (I);
    --      end loop;

    type Usage_Iterator  is private;
    type Outage_Iterator is private;
    type Event_Iterator  is private;
    type Device_Iterator is private;
    type Daemon_Iterator is private;
    type Mount_Iterator  is private;

    procedure Initialize (I : out Usage_Iterator);
    procedure Initialize (I : out Outage_Iterator);
    procedure Initialize (I : out Event_Iterator);
    procedure Initialize (I : out Device_Iterator);
    procedure Initialize (I : out Daemon_Iterator);
    procedure Initialize (I : out Mount_Iterator);
```





```
    procedure Next (I : in out Usage_Iterator);
    procedure Next (I : in out Outage_Iterator);
    procedure Next (I : in out Event_Iterator);
    procedure Next (I : in out Device_Iterator);
    procedure Next (I : in out Daemon_Iterator);
    procedure Next (I : in out Mount_Iterator);

    function Done (I : Usage_Iterator)  return Boolean;
    function Done (I : Outage_Iterator) return Boolean;
    function Done (I : Event_Iterator)  return Boolean;
    function Done (I : Device_Iterator) return Boolean;
    function Done (I : Daemon_Iterator) return Boolean;
    function Done (I : Mount_Iterator)  return Boolean;

    Job_Heap : constant System.Segment := Job_Segment.Get; -- cmg 03/06/92

    type Pstring is access String;  -- Strings are accessed by dereferencing
    --                                 pointers.

    pragma Segmented_Heap (Pstring); -- cmg 03/06/92


    -- Usage information is available for each half-hour during the
    -- report period.
    type Usage_Information is
        record
            Time         : Time_Utilities.Time;  -- Time of this sample
            Users        : Natural;              -- # users logged on
            Disk_Running : Boolean;      -- Disk Daemon running
            Outage       : Boolean;              -- System is down
        end record;


    -- Outage information is available for each system service outage.
    type Outage_Information is
        record
            Time        : Time_Utilities.Time;  -- time of outage
            Length      : Duration;             -- length of outage
            Cause       : Pstring;               -- Cause entered
            Explanation : Pstring;       -- Explanation entered
        end record;

    type Event_Class is (User_Operation, Exception_Cond,
                         System_Boot, Other_Event);

    -- Event information is available for each "interesting" event.
    -- The Event_Class gives some idea of the what the event is.
    -- The Info is the log entry for the event and has the standard
    -- format for a log entry.
```

```
type Event_Information is
    record
        Time       : Time_Utilities.Time;
        Info       : Pstring;
        Event_Kind : Event_Class;
    end record;

type Device_Class is (Disk, Tape, Ethernet, Memory, Other_Device);

-- Device information is available for each device error or other
-- event of interest.  Class indicates for which device it is, and
-- Info is the log entry for the event.
type Device_Information is
    record
        Time  : Time_Utilities.Time; -- Time of entry
        Info  : Pstring;             -- Log entry for device
        Class : Device_Class;        -- Class of device
    end record;

-- Daemon information is available for each run of a daemon.
-- The information is as listed below.
type Daemon_Information is
    record
        Time        : Time_Utilities.Time;          -- time of start
        Name        : Bounded_String.Variable_String (40);-- Daemon name
        Length      : Duration;                     -- length of run
        Pre_Size    : Natural;                    -- pages at start
        Post_Size   : Natural;                   -- pages of state at end
        Explanation : Pstring;                 -- Other info
    end record;

type Mount_Information is
    record
        Request_Time : Time_Utilities.Time;       -- time of request
        Volume       : Bounded_String.Variable_String (40);   -- Volume Name
        Mount_Time   : Time_Utilities.Time;         -- Tape on-line
        Unload_Time  : Time_Utilities.Time;        -- Tape unloaded
        Density      : Bounded_String.Variable_String (20);
    end record;

-- The value functions return the actual information for
-- each value of the iterator.
function Value (I : Usage_Iterator)  return Usage_Information;
function Value (I : Outage_Iterator) return Outage_Information;
function Value (I : Event_Iterator)  return Event_Information;
function Value (I : Device_Iterator) return Device_Information;
function Value (I : Daemon_Iterator) return Daemon_Information;
function Value (I : Mount_Iterator)  return Mount_Information;
```





```
private
    type Usage_Iterator  is new Integer;
    type Outage_Iterator is new Integer;
    type Event_Iterator  is new Integer;
    type Device_Iterator is new Integer;
    type Daemon_Iterator is new Integer;
    type Mount_Iterator  is new Integer;

end System_Information;
```

```
package System_Report is

-- Report generation from system availability information.
-- Provide a variety of reports.

    type Report_Class is (Availability, -- Uptime/downtime by classes
                          Usage,        -- Per half hour, # users, etc.
                          Disk,         -- Used disk space each day
                          Devices,      -- Disk, Mem, Tape, etc errors
                          Daemons,      -- Daemon state sizes and times
                          Outages,      -- System outages and reasons
                          Trouble,      -- Potential trouble areas
                          Advice,       -- Advice on cleaning things up
                          Everything,   -- All reports
                          Tape_Mounts); -- Tape processing for backups

    procedure Generate (Report_Type : Report_Class := System_Report.Everything;
                        Start_Time : String := "";
                        End_Time : String := "";
                        Log_Directory : String := "!Machine.Error_Logs");

    -- Run a report of the specified type.  Output goes to current
    -- output.  Start_Time null or illegal means "earliest time for
    -- which there is information".  End_Time null of illegal
    -- means "now".  Log_Directory is directory from which error
    -- log files will be read; this is changed mostly for testing.

    procedure Show_Bad_Blocks;
    procedure Show_Machine_Information;

    -- Produce a specific report about some specific subject.

end System_Report;
```





```
with Directory;
with Machine;
with Calendar;
with System;

package System_Utilities is

    subtype Job_Id     is Machine.Job_Id range 4 .. 255;
    subtype Session_Id is Machine.Session_Id;

    subtype Version is Directory.Version;
    subtype Object  is Directory.Object;

    subtype Port is Natural range 0 .. 4 * 16 * 16;
    subtype Tape is Natural range 0 .. 4;

    subtype Byte_String is System.Byte_String;

    -- Job (Process) characteristics
    function Get_Job return Job_Id;
    function Priority
                (For_Job : Job_Id := System_Utilities.Get_Job) return Natural;
    function Elapsed
                (For_Job : Job_Id := System_Utilities.Get_Job) return Duration;
    function Cpu (For_Job : Job_Id := System_Utilities.Get_Job) return Duration;
    function Job_Name
                (For_Job : Job_Id := System_Utilities.Get_Job) return String;


    -- Active Session characteristics
    function Get_Session                    return Session_Id;
    function Get_Session (For_Job : Job_Id) return Session_Id;

    function Session_Name
                (For_Session : Session_Id := System_Utilities.Get_Session)
                return String;
    function User_Name
                (For_Session : Session_Id := System_Utilities.Get_Session)
                return String;
    function Terminal (For_Session : Session_Id := System_Utilities.Get_Session)
                      return Port;
    function Terminal (For_Session : Session_Id := System_Utilities.Get_Session)
                      return Version;
    function Terminal (For_Session : Session_Id := System_Utilities.Get_Session)
                      return Object;
    function Session  (For_Session : Session_Id := System_Utilities.Get_Session)
                     return Version;
```

```
function Session  (For_Session : Session_Id := System_Utilities.Get_Session)
                return Object;
function User     (For_Session : Session_Id := System_Utilities.Get_Session)
             return Version;
function User     (For_Session : Session_Id := System_Utilities.Get_Session)
             return Object;

-- Inactive Session characteristics

function Home_Library (User : String := User_Name) return String;
function Last_Login (User : String; Session : String := "")
                return Calendar.Time;
function Last_Logout (User : String; Session : String := "")
                return Calendar.Time;
function Logged_In (User : String; Session : String := "") return Boolean;

-- Names for Text_IO/Simple_Text_IO standard file names

function Output_Name
            (For_Session : Session_Id := System_Utilities.Get_Session)
            return String;
function Input_Name
            (For_Session : Session_Id := System_Utilities.Get_Session)
            return String;
function Error_Name
            (For_Session : Session_Id := System_Utilities.Get_Session)
            return String;
function Tape_Name (Drive : Tape := 0) return String;

-- Terminal characteristics

subtype Stop_Bits_Range      is Integer range 1 .. 2;
subtype Character_Bits_Range is Integer range 5 .. 8;
type    Parity_Kind          is (None, Even, Odd);

function Terminal_Name
            (Line : Port := System_Utilities.Terminal) return String;
function Terminal_Type
            (Line : Port := System_Utilities.Terminal) return String;

function Input_Count
            (Line : Port := System_Utilities.Terminal) return Long_Integer;
function Output_Count
            (Line : Port := System_Utilities.Terminal) return Long_Integer;
-- The number of characters input/output from/to the specified terminal
-- since the machine was booted.  Input from the terminal that has not
-- been read by a session or user program will not be counted as input.
```





```
function Input_Rate
            (Line : Port := System_Utilities.Terminal) return String;
function Output_Rate
            (Line : Port := System_Utilities.Terminal) return String;

function Parity         (Line : Port := System_Utilities.Terminal)
                return Parity_Kind;
function Stop_Bits      (Line : Port := System_Utilities.Terminal)
                 return Stop_Bits_Range;
function Character_Size (Line : Port := System_Utilities.Terminal)
                    return Character_Bits_Range;

function Xon_Xoff_Characters
            (Line : Port := System_Utilities.Terminal) return String;
function Xon_Xoff_Bytes
            (Line : Port := System_Utilities.Terminal) return Byte_String;

function Receive_Xon_Xoff_Characters
            (Line : Port := System_Utilities.Terminal) return String;
function Receive_Xon_Xoff_Bytes
            (Line : Port := System_Utilities.Terminal) return Byte_String;
-- returns a 2-element string consisting of Xon followed by Xoff

function Flow_Control
            (Line : Port := System_Utilities.Terminal) return String;
function Receive_Flow_Control
            (Line : Port := System_Utilities.Terminal) return String;
-- return one of NONE, XON_XOFF, RTS, DTR

function Enabled (Line : Port := System_Utilities.Terminal) return Boolean;

function Disconnect_On_Disconnect
            (Line : Port := System_Utilities.Terminal) return Boolean;
function Logoff_On_Disconnect
            (Line : Port := System_Utilities.Terminal) return Boolean;
function Disconnect_On_Logoff
            (Line : Port := System_Utilities.Terminal) return Boolean;
function Disconnect_On_Failed_Login
            (Line : Port := System_Utilities.Terminal) return Boolean;
function Log_Failed_Logins
            (Line : Port := System_Utilities.Terminal) return Boolean;
function Login_Disabled
            (Line : Port := System_Utilities.Terminal) return Boolean;
function Detach_On_Disconnect
            (Line : Port := System_Utilities.Terminal) return Boolean;
```

```
-- Iterate over all active sessions

type Session_Iterator is private;
procedure Init  (Iter : out Session_Iterator);
function  Value (Iter : Session_Iterator) return Session_Id;
function  Done  (Iter : Session_Iterator) return Boolean;
procedure Next  (Iter : in out Session_Iterator);

-- Iterate over all jobs for a session

type Job_Iterator is private;
procedure Init  (Iter        : out Job_Iterator;
                 For_Session :     Session_Id := Get_Session);
function  Value (Iter : Job_Iterator) return Job_Id;
function  Done  (Iter : Job_Iterator) return Boolean;
procedure Next  (Iter : in out Job_Iterator);

type Terminal_Iterator is private;
procedure Init  (Iter : out Terminal_Iterator);
function  Value (Iter : Terminal_Iterator) return Natural;
function  Done  (Iter : Terminal_Iterator) return Boolean;
procedure Next  (Iter : in out Terminal_Iterator);

function System_Up_Time           return Calendar.Time;
function System_Boot_Configuration return String;

function Image (Version : Directory.Version) return String;

procedure Set_Page_Limit (Max_Pages : Natural;
                          For_Job   : Job_Id := System_Utilities.Get_Job);

-- Set the upper limit for pages created by the specified job.
-- Attempts to create additional pages result in Storage_Error.
-- Requires operator capability if For_Job specifies a job belonging
-- to a user different from the caller.  If For_Job parameter defaults,
-- then the limit applies to the calling job.  In the worst case,
-- the job can allocate twice Max_Pages pages before getting a storage
-- error.  Raises constraint_error if the job_id is illegal.

procedure Get_Page_Counts (Cache_Pages : out Natural;
                           Disk_Pages : out Natural;
                           Max_Pages : out Natural;
                           For_Job : Job_Id := System_Utilities.Get_Job);

-- Return the counts for the specified job.  Cache_Pages is the number of
-- pages presently in main memory; Disk_Pages is the number of pages that
-- have disk space allocated for them.  Max_Pages is the current page
-- limit.
```





```
-- Operations for reading machine information:

type Bad_Block_Kinds is new Long_Integer range 0 .. 7;
Manufacturers_Bad_Blocks : constant Bad_Block_Kinds := 1;
Retargeted_Blocks        : constant Bad_Block_Kinds := 2;
All_Bad_Blocks           : constant Bad_Block_Kinds := 3;

type Block_List is array (Natural range <>) of Integer;

function Bad_Block_List
            (For_Volume : Natural;
             Kind : Bad_Block_Kinds := Retargeted_Blocks) return Block_List;
-- Return the list of bad blocks of the specified kind on the specified
-- disk.  Return null array if Kind or volume are illegal.

function Get_Board_Info (Board : Natural) return String;
-- return information about the specified board in the machine.  The
-- string identifies the information.
--  Board specifies the particular board:
--        0 : IOA
--        1 : SYS/IOC
--        2 : SEQ
--        3 : VAL
--        4 : TYP
--        5 : FIU
--        100 : MEM0
--        101 : MEM1
--        102 : MEM2
--        103 : MEM3
--      etc.

function Terminal_Lines   (Line : Port := Terminal) return Natural;
function Terminal_Columns (Line : Port := Terminal) return Natural;

end System_Utilities;
```

```
with Io;

-- This package is used to produce neatly formatted tables with centered
-- headers and even amounts of white space between the columns.  The first
-- N calls should be to header, which defines a header and a type of
-- justification for the items that will go into each column.  Then the M*N
-- items of an M line table are sent into the package a row at a time.  An
-- item is defined by either a single call to Item, or a series of zero or
-- more calls to Subitem terminated by a call to Last_Subitem.  Multiple
-- parts of an item are separated by the subitem separator.  After all the
-- items have been defined, the table is output with a call to Display.

-- The package internally allocates enough memory to save a copy of the
-- entire table.   It is therefore a good idea to instantiate this
-- procedure in a local frame so that all the memory it allocates will go
-- away when the frame does.

-- An instantiation of this package will generate at most one table.
-- It is NOT possible to start over calling Header after Display.

generic
    Number_Of_Columns : Positive;
    Subitem_Separator : String := " ";

package Table_Formatter is

    type Adjust is (Left, Right, Centered);
    procedure Header  (S : String; Format : Adjust := Left);
    procedure Item    (S : String);
    procedure Subitem (S : String);
    procedure Last_Subitem;
    procedure Display (On_File : Io.File_Type);

    type Field_List is array (Integer range <>) of Integer;
    -- For N > 0 sort by field N in increasing order.
    -- For N < 0 sort by field abs (N) in decreasing order.
    -- Sorting is done on input value before right adjustment or centering.

    procedure Sort (On_Field : Integer := 1);
    procedure Sort (On_Fields : Field_List);

end Table_Formatter;
```





```
generic
    type Element is private;
    pragma Must_Be_Constrained (Yes => Element);

    type Index         is (<>);
    type Element_Array is array (Index range <>) of Element;
    with function "<" (Left, Right : Element) return Boolean is <>;

procedure Table_Sort_Generic (Table : in out Element_Array);
```

```
with Device_Independent_Io;

package Tape_Tools is

    type Logical_Device is private;

    subtype Vol_Id_String    is String;  -- (1 .. 6);
    subtype Vol_Name_String  is String;  -- (1 .. 76);
    subtype Owner_String     is String;  -- (1 .. 13);
    subtype File_Id_String   is String;  --  (1 .. 17);
    subtype File_Name_String is String;  -- (1 .. 532);

    subtype User_Label_Number is Natural range 0 .. 255;

    subtype Byte  is Device_Independent_Io.Byte;
    subtype Bytes is Device_Independent_Io.Byte_String;

    procedure Initialize (Tape   : out Logical_Device;
                          Format :     String := "R1000");

    function Initialize (Format : String := "R1000") return Logical_Device;

    procedure Connect_For_Input (Tape        : in out Logical_Device;
                                 Vol_Id      :        Vol_Id_String;
                                 Vol_Name    :        Vol_Name_String := "";
                                 To_Operator :        String := "Thank You");

    -- Request tape with given Vol_Id/Vol_Name be mounted on a drive for
    -- the purpose of reading.

    procedure Connect_For_Output (Tape        : in out Logical_Device;
                                  Vol_Id      :        Vol_Id_String;
                                  Vol_Name    :        Vol_Name_String := "";
                                  Owner       :        String := "";
                                  To_Operator :        String := "Thank You");

    -- request a tape be mounted on a drive for writing; Assign the
    -- Vol_Id/Vol_Name/Owner per given parameters;

    function Vol_Id (Tape : Logical_Device) return String;

    -- Volume Id of mounted tape.

    function Vol_Name (Tape : Logical_Device) return String;

    -- Volume name of mounted tape (format dependent)
```





```
    function Owner (Tape : Logical_Device) return String;

    -- Owner of mounted tape (format dependent)

    procedure Label (Tape   : Logical_Device;
                     Number : User_Label_Number;
                     Label  : String);

    -- Define a User Label for the next Create.
    -- Up to 256 labels may be defined for one file

    type Record_Format is (Fixed_Length, Variable_Length, Spanned, Undefined);

    Default_Record_Format : constant Record_Format := Variable_Length;
    Default_Record_Length : constant Natural       := 512;
    Default_Block_Length  : constant Natural       := 2048;

    procedure Format (Tape          : Logical_Device;
                      Kind          : Record_Format := Default_Record_Format;
                      Record_Length : Natural       := Default_Record_Length;
                      Block_Length  : Natural       := Default_Block_Length);

    -- Define the record format for the next Create.  When the
    -- Logical_Device is initialized the record format is set to the above
    -- defaults.  An changes made by this procedure remain in effect until
    -- the next call of this procedure or the tape is disconnected.

    procedure Create (Tape : in out Logical_Device;
                      Id   :        File_Id_String;
                      Name :        File_Name_String := "");

    -- Start a new output file with the given Id (and Name) and any user
    -- labels defined before this call.

    procedure Open (Tape : in out Logical_Device);

    -- Open the next file on the tape;

    function File_Id (Tape : Logical_Device) return String;

    -- File Id of currently open tape file.

    function File_Name (Tape : Logical_Device) return String;

    -- File name of file opened/created on tape; (Format dependent);

    function Labels (Tape : Logical_Device) return Natural;

    -- number of user labels associated with the currently open file
```

```
function Label (Tape : Logical_Device; Index : Natural) return String;
function Label (Tape : Logical_Device; Index : Natural) return Natural;

-- Index-th user label text or number associated with currently open file.
-- Index must be less than the value returned by Labels

function Format (Tape : Logical_Device) return Record_Format;

-- The record format of the currently open file

function Block_Length (Tape : Logical_Device) return Natural;

-- The block length of the currently open file

function Record_Length (Tape : Logical_Device) return Natural;

-- The Record Length of the currently open file

procedure Skip (Tape : Logical_Device; Number : Integer := 1);

procedure Put_Line (Tape : Logical_Device; Line : String);
function  Get_Line (Tape : Logical_Device) return String;
procedure Get_Line (Tape   :     Logical_Device;
                    Line   : out String;
                    Length : out Natural);
procedure Get_Line (Tape   :     Logical_Device;
                    Line   : out String;
                    Length : out Natural;
                    Eof    : out Boolean);

procedure Put (Tape : Logical_Device; Data : Bytes);
function  Get (Tape : Logical_Device) return Bytes;
procedure Get (Tape   :     Logical_Device;
               Data   : out Bytes;
               Length : out Natural);
procedure Get (Tape   :     Logical_Device;
               Data   : out Bytes;
               Length : out Natural;
               Eof    : out Boolean);

function End_Of_File (Tape : Logical_Device) return Boolean;
function End_Of_Tape (Tape : Logical_Device) return Boolean;

procedure Close      (Tape : Logical_Device);
procedure Disconnect (Tape : Logical_Device);
procedure Abandon    (Tape : Logical_Device);
```





```
    type Condition is (No_Errors, Vol_Already_Open_Or_Created,
                       Vol_Not_Open_Or_Created, Not_Initialized,
                       Not_Original_Client, Read_While_Writing,
                       Write_While_Reading, Position_While_Writing,
                       Previous_Fatal_Error, File_Still_Being_Written,
                       File_Still_Being_Read, File_Not_Created,
                       File_Not_Open, Retry_Count_Exhausted, Vol_Set_Error,
                       Unexpected_Tape_Error, No_Drive_Available,
                       Desired_Drive_Unavailable, Desired_Volume_Not_Found,
                       Vol_Access_Denied, File_Access_Denied,
                       File_Expired, End_Of_Vol_Set_Encountered,
                       Incorrect_File_Seq_No, Incorrect_File_Sect_No,
                       Need_Vol_Completion, Not_At_Eof, Not_At_Eov, Not_At_Eovs,
                       Incorrect_Buffer_Size, Block_Length_Too_Short,
                       Block_Length_Too_Long, File_Not_In_Vol_Set,
                       Record_Not_In_File, Format_Violation,
                       Unimplemented_Format, Attempt_To_Read_While_Writing,
                       Attempt_To_Write_While_Reading, Other_Error);

    function Status   (Tape : Logical_Device) return Condition;
    function Status   (Tape : Logical_Device) return String;
    function Is_Fatal (Error : Condition)     return Boolean;
    function Is_Fatal (Tape : Logical_Device) return Boolean;

private
    type Logical_Device_Record;
    type Logical_Device is access Logical_Device_Record;
    pragma Segmented_Heap (Logical_Device);

end Tape_Tools;
```

```
with Calendar;

package Time_Utilities is

    Minute : constant Duration := 60.0;
    Hour   : constant Duration := 3600.0;
    Day    : constant Duration := 86_400.0;

    ------------------------------------------------------------------
    -- Time_Utilities.Time is a segmented version of Calendar.Time
    --        with image and value functions
    ------------------------------------------------------------------

    type Years  is new Calendar.Year_Number;
    type Months is (January, February, March, April, May, June, July,
                    August, September, October, November, December);
    type Days   is new Calendar.Day_Number;

    type Hours   is new Integer range 1 .. 12;
    type Minutes is new Integer range 0 .. 59;
    type Seconds is new Integer range 0 .. 59;

    type Sun_Positions is (Am, Pm);

    type Time is
        record
            Year         : Years;
            Month        : Months;
            Day          : Days;
            Hour         : Hours;
            Minute       : Minutes;
            Second       : Seconds;
            Sun_Position : Sun_Positions;
        end record;

    function Get_Time return Time;

    function Convert_Time (Date : Calendar.Time) return Time;
    function Convert_Time (Date : Time)          return Calendar.Time;

    function Nil                  return Time;
    function Is_Nil (Date : Time) return Boolean;

    function Nil                           return Calendar.Time;
    function Is_Nil (Date : Calendar.Time) return Boolean;
```





```
    type Time_Format is (Expanded,  -- 11:00:00 PM
                         Military,  -- 23:00:00
                         Short,     -- 23:00
                         Ada        -- 23_00_00
                         );

    type Date_Format is (Expanded,        -- September 29, 1983
                         Month_Day_Year,  -- 09/29/83
                         Day_Month_Year,  -- 29-SEP-83
                         Year_Month_Day,  -- 83/09/29
                         Ada              -- 83_09_29
                         );

    type Image_Contents is (Both, Time_Only, Date_Only);

    function Image (Date       : Time;
                    Date_Style : Date_Format    := Time_Utilities.Expanded;
                    Time_Style : Time_Format    := Time_Utilities.Expanded;
                    Contents   : Image_Contents := Time_Utilities.Both)
                   return String;

    function Value (S : String) return Time;

    ------------------------------------------------------------------
    -- Time_Utilities.Interval is a segmented version of Duration
    --        with image and value functions
    ------------------------------------------------------------------

    type Day_Count      is new Integer range 0 .. Integer'Last;
    type Military_Hours is new Integer range 0 .. 23;
    type Milliseconds   is new Integer range 0 .. 999;

    type Interval is
        record
            Elapsed_Days         : Day_Count;
            Elapsed_Hours        : Military_Hours;
            Elapsed_Minutes      : Minutes;
            Elapsed_Seconds      : Seconds;
            Elapsed_Milliseconds : Milliseconds;
        end record;

    function Convert (I : Interval) return Duration;
    function Convert (D : Duration) return Interval;

    function Image (I : Interval) return String;
    function Value (S : String)   return Interval;

    function Image (D : Duration) return String;
```

```
    function Duration_Until (T : Time)          return Duration;
    function Duration_Until (T : Calendar.Time) return Duration;
    function Duration_Until_Next
                (H : Military_Hours; M : Minutes := 0; S : Seconds := 0)
                return Duration;

    -- Day of week support; Monday is 1.
    type Weekday is new Positive range 1 .. 7;

    function Day_Of_Week (T : Calendar.Time)                     return Weekday;
    function Day_Of_Week (T : Time := Time_Utilities.Get_Time)  return Weekday;
    function Image       (D : Weekday)                          return String;

    function "+" (D : Weekday; I : Integer) return Weekday;
    function "-" (D : Weekday; I : Integer) return Weekday;

end Time_Utilities;
```





```
generic
    Default_Maximum_Length : Natural := 20;

package Unbounded_String is

-------------------------------------------------------------------------------
    --  Managed Pointer Sequential Unbounded Strings:
    --  Restrictions and assumptions
    -- 1.   Storage management is performed
    -- 2.   Extending a string in a way that requires a new allocation allows
    --      space for expansion.
    -- 3.   CANNOT be used by multiple tasks unless user provides serialization
    -- 4.   := is reference copy, use copy to assign contents
    -- 5.   Uninitialized or Freed objects are true null's and changes to one
    --      of the referents will not be reflected in the other;
    -- 6.   Use Free prior to assignment to prevent garbage
    -- 7.   = is object identity, compare Images for value equality

-------------------------------------------------------------------------------

    subtype String_Length   is Natural;
    type    Variable_String is private;

    -- release storage associated with a string
    procedure Free (V : in out Variable_String);

    -- Get information about current length or contents of a string
    function Length  (Source : Variable_String) return String_Length;
    function Char_At (Source : Variable_String; At_Pos : Positive)
                 return Character;

    function Extract (Source    : Variable_String;
                      Start_Pos : Positive;
                      End_Pos   : Natural) return String;

    function Image (V : Variable_String) return String;
    function Value (S : String)          return Variable_String;

    -- Image (Target) := Image (Source);
    procedure Copy (Target : in out Variable_String; Source : Variable_String);
    procedure Copy (Target : in out Variable_String; Source : String);
    procedure Copy (Target : in out Variable_String; Source : Character);

    -- Target := Source; Source := ""; with appropriate storage management
    procedure Move (Target : in out Variable_String;
                    Source : in out Variable_String);
```

```
    -- Target := Target & Source;
    procedure Append (Target : in out Variable_String;
                      Source :        Variable_String);

    procedure Append (Target : in out Variable_String; Source : String);

    procedure Append (Target : in out Variable_String; Source : Character);

    procedure Append (Target : in out Variable_String;
                      Source :        Character;
                      Count  :        String_Length);

    -- Target := Target (1..At_Pos-1) & Source & Target (At_Pos..Target'Length)
    procedure Insert (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Source :        Variable_String);

    procedure Insert (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Source :        String);

    procedure Insert (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Source :        Character);

    procedure Insert (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Source :        Character;
                      Count  :        String_Length);

    -- Target (At_Pos .. At_Pos + Count -1) := "";
    procedure Delete (Target : in out Variable_String;
                      At_Pos :        Positive;
                      Count  :        String_Length := 1);

    -- Target (At_Pos .. At_Pos + Source'Length - 1) := Source;
    procedure Replace (Target : in out Variable_String;
                       At_Pos :        Positive;
                       Source :        Character);

    procedure Replace (Target : in out Variable_String;
                       At_Pos :        Positive;
                       Source :        Character;
                       Count  :        String_Length);

    procedure Replace (Target : in out Variable_String;
                       At_Pos :        Positive;
                       Source :        String);
```





```
    procedure Replace (Target : in out Variable_String;
                       At_Pos :        Positive;
                       Source :        Variable_String);

    -- Target'Length := New_Length;
    -- Target (Target'Length .. New_Length) := Fill_With;
    procedure Set_Length (Target     : in out Variable_String;
                          New_Length :        String_Length;
                          Fill_With  :        Character := ' ');

    -- Determine if a Variable_String is null; different from = ""
    function Is_Nil (V : Variable_String) return Boolean;

    -- Return a null Variable_String.  Note that assignment of Nil may
    -- create garbage; see procedure Free above.
    function Nil return Variable_String;

private
    type Pointer is access String;

    type    Real_String;
    type    Variable_String is access Real_String;
    subtype String_Bound    is Integer range -1 .. Integer'Last;

    type Real_String is
        record
            Length    : String_Bound;
            Contents  : Pointer;
            Next_Free : Variable_String;
        end record;

    Null_String : Pointer := new String (1 .. 0);

    Free_List_Item : constant String_Bound := -1;

    Free_List : Real_String :=
       Real_String'(Free_List_Item, Null_String,
                    new Real_String'(Free_List_Item, Null_String, null));

end Unbounded_String;
```

```
with System;

package Unchecked_Conversions is

    generic
        type Source is limited private;
        type Target is limited private;
    package Unchecked_Conversion_Package is
        function Convert (S : Source) return Target;
        -- Package form of LRM Unchecked_Conversion.
        -- Type-specific calculations are made during package elaboration, making
        -- calls to this convert faster than to an equivalent Unchecked-
        -- _Conversion instantiation.  Speed improvement depends on the
        -- type involved.
    end Unchecked_Conversion_Package;


    generic
        type Source is limited private;
    function Convert_To_Byte_String (S : Source) return System.Byte_String;
    -- Convert from Source to a byte string.  The byte string may contain
    -- more bits than the object.

    generic
        type Target is limited private;
    function Convert_From_Byte_String (S : System.Byte_String) return Target;
    -- Convert from a byte string to a Target type.  The string should
    -- have been produced by an instantiation of Convert_To_Byte_String
    -- with the same type.  A constrained object is always returned.

end Unchecked_Conversions;
```





```
package Xref is

    procedure Used_By (List_Of_Names             : String  := "<IMAGE>";
                       Do_Functions              : Boolean := True;
                       Do_Generics               : Boolean := True;
                       Do_Procedures             : Boolean := True;
                       Do_Attributes             : Boolean := False;
                       Do_Record_Components      : Boolean := False;
                       Do_Constants              : Boolean := False;
                       Do_Entries                : Boolean := False;
                       Do_Exceptions             : Boolean := False;
                       Do_Labels                 : Boolean := False;
                       Do_Packages               : Boolean := False;
                       Do_Parameters             : Boolean := False;
                       Do_Pragmas                : Boolean := False;
                       Do_Task_Bodies            : Boolean := True;
                       Do_Types                  : Boolean := False;
                       Do_Variables              : Boolean := False;
                       Exclude_References_From   : String  := "";
                       List_File_Name            : String  := "");

    -- Produce a report showing all of the units that reference (use) something
    -- defined in the units specified in "List_Of_Names".  Only the IDs defined
    -- in units specified in "List_Of_Names" will be included in the report.
    -- "Using" units can be excluded by listing their names in
    -- "Exclude_References_From".
```

```
    procedure Uses (List_Of_Names              : String  := "<IMAGE>";
                    Visible_Declarations_Only  : Boolean := True;
                    Do_Functions               : Boolean := True;
                    Do_Generics                : Boolean := True;
                    Do_Procedures              : Boolean := True;
                    Do_Attributes              : Boolean := False;
                    Do_Record_Components       : Boolean := False;
                    Do_Constants               : Boolean := False;
                    Do_Entries                 : Boolean := False;
                    Do_Exceptions              : Boolean := False;
                    Do_Labels                  : Boolean := False;
                    Do_Packages                : Boolean := False;
                    Do_Parameters              : Boolean := False;
                    Do_Pragmas                 : Boolean := False;
                    Do_Task_Bodies             : Boolean := True;
                    Do_Types                   : Boolean := False;
                    Do_Variables               : Boolean := False;
                    Exclude_References_To      : String  := "";
                    Only_References_To         : String  := "";
                    List_File_Name             : String  := "");

    -- produce a report of the items referenced by the units specified
    -- in "list_of_names".  Units mentioned in "Exclude_references_To"
    -- are not included in the report.  If units are mentioned in
    -- Only_references_to then these are the only units included in the
    -- report.  It is an error to specify both Exclude... and Only...

end Xref;
```

## STANDARD ABBREVIATIONS

The following abbreviations are provided by the Rational Environment to reduce the typing required for executing frequently used commands and referring to commonly used units. These abbreviations are defined in world !Commands.Abbreviations and are visible through an entry on your searchlist.

The unit-name abbreviations defined by links are:

```
LINK NAME      => SOURCE

ACL            => !COMMANDS.ACCESS_LIST
CMVC_ACL       => !COMMANDS.CMVC_ACCESS_CONTROL
COMP           => !COMMANDS.COMPILATION
DEMON          => !COMMANDS.DAEMON
DIR            => !COMMANDS.LIBRARY
FILE           => !COMMANDS.FILE_UTILITIES
LIB            => !COMMANDS.LIBRARY
LINK           => !COMMANDS.LINKS
MSG            => !COMMANDS.MESSAGE
OP             => !COMMANDS.OPERATOR
Q              => !COMMANDS.QUEUE
SA             => !COMMANDS.ARCHIVE
SL             => !COMMANDS.SEARCH_LIST
SOURCE_ARCHIVE => !COMMANDS.ARCHIVE
SWITCH         => !COMMANDS.SWITCHES
VIEW           => !COMMANDS.CMVC
WO             => !COMMANDS.WORK_ORDER
```

The following procedures provide abbreviations of commands. Note that parameters not included from the underlying command default to the default values defined for the underlying command. For more information on these abbreviations, examine their bodies in world !Commands.Abbreviations.

```
procedure Aedit (The_Activity : String := "<ACTIVITY>");
-- Activity.Edit


procedure Alist (Pattern    : String  := "@'C(ADA)";
                 Descending : Boolean := False;
                 Response   : String  := "<PROFILE>";
                 Options    : String  := "");
-- Library.Ada_List
```





```
procedure Cancel_Print_Request
          (Printer : String := "<Default>"; Request_Id : Positive);
-- Queue.Cancel
-- Removes the specified print request from the queue of the
-- specified printer.
--
-- The Printer parameter accepts any printer name that has been
-- defined in the printer-configuration files.  By default, the
-- parameter specifies the printer that is associated with the
-- user who enters the command.
--
-- Print request numbers, can be obtained from the display
-- generated by the Display_Queue procedure in this directory.


procedure Code (Unit        : String  := "<IMAGE>";
                Limit       : String  := "<WORLDS>";
                Effort_Only : Boolean := False;
                Response    : String  := "<PROFILE>");
-- Compilation.Make. Redirects logging output to a file.
-- Checks for errors.


procedure Compare (File_1      : String  := "<REGION>";
                   File_2      : String  := "<IMAGE>";
                   Subobjects  : Boolean := False;
                   Ignore_Case : Boolean := False;
                   Options     : String  := "");
-- File_Utilities.Compare


procedure Ddef (Location    : String  := "<SELECTION>";
                Stack_Frame : Integer := 0);
-- Debug.Source


procedure Def (Name     : String  := "<CURSOR>";
               In_Place : Boolean := False;
               Visible  : Boolean := True);
-- Common.Definition


procedure Diff (File_1            : String  := "<REGION>";
                File_2            : String  := "<IMAGE>";
                Result            : String  := "";
                Compressed_Output : Boolean := False;
                Subobjects        : Boolean := False);
-- File_Utilities.Difference
```

```
with Operator;
procedure Disk_Space;
-- Operator.Disk_Space

procedure Display_Queue (Printer : String := "<Default>");
-- Queue.Display
-- Displays the print requests currently queued on the specified
-- printer.
--
-- The display shows the identification number for each request.
--
-- The Printer parameter accepts any print name that has been
-- defined in the printer_configuration files.  By default,
-- the parameter specifies the printer that is associated with
-- the user who enters the command.

with System_Backup;
procedure Do_Backup (Variety     : System_Backup.Kind := System_Backup.Full;
                     Starting_At : String             := "");
-- System_Backup.Backup. Performs various other housekeeping functions.

with Library;
procedure Exp (Existing : String := "<CURSOR>");
-- Library.Expunge

procedure Find (Pattern     : String  := "";
                File        : String  := "<IMAGE>";
                Wildcards   : Boolean := False;
                Ignore_Case : Boolean := True;
                Result      : String  := "");
-- File_Utilities.Find

procedure Full_Backup (Starting_At : String := "");
-- Do_Backup (see above)

procedure Help (Name : String := "Help_On_Help");
-- What.Does

procedure Input (Name : String := "<CURSOR>");
-- Io.Set_Input
```





```
procedure Install (Unit        : String  := "<IMAGE>";
                   Limit       : String  := "<WORLDS>";
                   Effort_Only : Boolean := False;
                   Response    : String  := "<PROFILE>");
-- Compilation.Promote. Redirects logging output to a file.
-- Checks for errors.

procedure Ledit (World : String := "<IMAGE>");
-- Links.Edit

procedure List (Pattern    : String  := "@";
                Descending : Boolean := False;
                Response   : String  := "<PROFILE>";
                Options    : String  := "");
-- Library.List

procedure Need (Unit       : String  := "<IMAGE>";
                Transitive : Boolean := False;
                Response   : String  := "<PROFILE>");
-- Compilation.Dependents

procedure Output (Name : String := ">>FILE NAME<<");
-- Log.Set_Output; Log.Set_Error

procedure Primary_Backup (Starting_At : String := "");
-- Do_Backup (see above)

procedure Print
   (Object_Or_Image                         : String   := "<CURSOR>";
    From_First_Page                         : Positive := 1;
    To_Last_Page                            : Positive := 3000;
    Display_As_Twoup                        : Boolean  := True;
    Display_Border                          : Boolean  := True;
    Display_Filename                        : Boolean  := True;
    Display_Date                            : Boolean  := True;
    Ignore_Display_Parameters_For_Postscript : Boolean  := True;
    Highlight_Reserved_Words_For_Ada        : Boolean  := True;
    Other_Options                           : String   := "";
    Number_Of_Copies                        : Positive := 1;
    Printer                                 : String   := "<Default>";
    Effort_Only                             : Boolean  := False);
-- Queue.Print
-- Prints one or more objects.
```

```
-- Additional parameters specify how the object(s) should be printed.
-- The printer used is based on the user via a map set up by the system
-- manager. For more information, see the online specification.

procedure Print_Window (Options : String := "<DEFAULT>";
                        Banner  : String := "<DEFAULT>";
                        Header  : String := "<DEFAULT>";
                        Footer  : String := "<DEFAULT>");

-- Copies the content of the current window (its image) into a temporary
-- file, prints that file using queue.print, then destroys that file.
-- The file is named
-- !machine.temporary.<User_Name>_<Job_Number>_<Window_Type>

procedure Run_Job (S        : String   := "<SELECTION>";
                   Debug    : Boolean  := False;
                   Context  : String   := "$";
                   After    : Duration := 0.0;
                   Options  : String   := "";
                   Response : String   := "<PROFILE>");
-- Program.Run_Job

procedure Schedule_Shutdown (At_Time     : String := "23:59";
                             Reason      : String := "COPS";
                             Explanation : String := "Cause not entered");
-- Operator.Shutdown. Also sends warning message.

procedure Secondary_Backup (Starting_At : String := "");
-- Do_Backup (see above)

procedure Sedit (Switch_File : String := "<SWITCH>");
-- Switches.Edit

procedure Sledit (Session : String := "";
                  User    : String := "");
-- Search_List.Show_List. Allows editing.

procedure Ssedit (Switch_File : String := "<SESSION>");
-- Switches.Edit. Edits session switches by default.
```





```
procedure Users (Jobs_Too : Boolean := False;
                 Verbose  : Boolean := False);
-- Similar to What.Users but displays information only for
-- the current user and, optionally, the user's jobs.

procedure Vlist (Pattern    : String  := "@";
                 Descending : Boolean := False;
                 Response   : String  := "<PROFILE>";
                 Options    : String  := "");
-- Library.Verbose_List
```

# Terminal Types and Keymaps

A *terminal type* identifies a set of input and output characteristics associated with a given terminal, terminal emulator, or workstation. Terminal types usually are associated with an autorecognition sequence that the R1000 uses to automatically identify specific terminal types during login. Terminal types also can be specified at the `Enter Terminal Type` prompt during login.

This section provides:

- A list of the terminal types defined by Rational for use of the Environment directly and through the Rational Windows Interface (RWI) and Rational X Interface (RXI) terminal emulators
- A brief overview of the objects that define terminal types, describing what they do and identifying relationships between them

When a reference to *Terminal_Type* appears in this section, you can substitute any valid terminal type. For example, if you are using the Pc101 terminal type, you should substitute Pc101_Commands wherever *Terminal_Type*_Commands is specified.

For information about creating and using terminal types, refer to the *System Manager's Guide*. For information about creating user-specific key bindings for an existing terminal type, see the Key Concepts tabbed section of the Session and Job Management (SJM) book.

## PREDEFINED TERMINAL TYPES

Rational provides terminal types for direct use with the standard Environment and for use of the Environment through the Rational Windows Interface (RWI) and the Rational X Interface (RXI) terminal emulators. The terminal types supplied by Rational for each of these interfaces are described below.

### Standard Terminal Types

The standard Environment includes defining objects for three terminal types. These terminal types and the kinds of terminals they support are listed in Table 1.

***Table 1 Standard Environment Terminal Types***

| Terminal Type | Supports |
|---|---|
| Facit | Facit terminal |
| Rational | Rational terminal |
| VT100™ | VT100 terminal |

## Rational Windows Interface Terminal Types

The Rational Windows Interface (RWI) terminal emulator makes it possible to access the Environment from IBM personal computers (PCs). RWI includes terminal types to support the configurations listed in Table 2.

***Table 2 Rational Windows Interface Terminal Types***

| Terminal Type | Supports |
|---|---|
| Pc86 | Personal computer with Pc86 keyboard |
| Pc91 | Personal computer with Pc91 keyboard |
| Pc101 | Personal computer with Pc101 keyboard |

## Rational X Interface Terminal Types

The Rational X Interface (RXI) terminal emulator makes it possible to access the Environment from many different types of X terminals and workstations. RXI includes terminal types to support the configurations listed in Table 3.

***Table 3 Rational X Interface Terminal Types***

| Terminal Type | Workstation | Operating System | Window System | Keyboard |
|---|---|---|---|---|
| Xdecus | DEC VAXstation™ and DECstation™ | VMS™ | DECwindows™ | LK201 |
| Xhp46021a | HP 9300 | HP-UX | X Window System™ | 46021a |
| Xncd | n/a* | n/a* | n/a* | NCD 101 |
| Xr6us | IBM RISC System/6000™ | AIX™ | AIX/R2 X Windows | IBM U.S. |
| Xnews4 | Sun Workstation® | SunOS™ | Sun X11/NeWS™ (OpenWindows™) | Type 4 |
| Xnews101 | Sun Workstation | SunOS | Sun X11/NeWS (OpenWindows) | Type 101A |
| Xsun4 | Sun Workstation | SunOS | MIT X11 | Type 4 |
| Xsun101 | Sun Workstation | SunOS | MIT X11 | Type 101A |
| Xultus | DEC DECstation | ULTRIX™ | DECwindows | LK201 |

* NCD terminals can be used with any of the workstations, operating systems, and window managers listed in this table.

# OBJECTS THAT DEFINE TERMINAL TYPES

The objects that define terminal types are located in the !Machine.Editor_Data world. All terminal types are made up of at least the following three objects (where *Terminal_Type* stands for the name of any recognized terminal type, such as those listed in the previous section):

- A *Terminal_Type*_Keys file
- A *Terminal_Type*_Key_Names package
- A *Terminal_Type*_Commands procedure

A terminal type may also include:

- A *Terminal_Type*_User_Commands file
- An entry in the Terminal_Types file
- An entry in the Terminal_Recognition file

These objects are described in the following sections, according to the kind of operation they perform:

- The *Terminal_Type*_Keys file and *Terminal_Type*_Key_Names package are used for key naming and character recognition.
- The *Terminal_Type*_Commands procedure binds Environment commands to particular keys or key combinations. A template for this procedure exists in the *Terminal_Type*_User_Commands file.
- The Terminal_Types and Terminal_Recognition files define the terminal types that can be recognized by the Environment.

## Key Naming and Character Recognition

***Caution:** You should not make changes to the key-naming and character-recognition objects in !Machine.Editor_Data, because this changes the systemwide key bindings, possibly preventing other users from logging in.*

### The *Terminal_Type*_Keys File

For the Environment to recognize which keys are pressed on a given terminal's keyboard, each terminal type includes a list of character definitions called the *Terminal_Type*_Keys file. The *Terminal_Type*_Keys text file specifies to the Environment what characters to receive and what keystrokes to infer based on those received characters. Two maps are created from this file:

- A two-way map between key names and the Environment's internal key-code values
- A one-way map from ASCII character sequences to internal key codes

Part of the Xr6us_Keys file (for use with RXI on an IBM RS/6000) is shown here:

```
52
1 C_G
1 C_L
2 @ ' '
2 A '!'
2 B '"'
2 C '#'
2 D '$'
...
3(!  '@'
3(!! 'A'
3(!" 'B'
3(!# 'C'
3(!$ 'D'
...
3(-! CMS_A
```

```
3(-" CMS_B
3(-# CMS_C
3(-$ CMS_D
...
41_@  MENU_JOB_KILL
41_@! MENU_JOB_DISABLE
41_@" MENU_DEBUG_STOP
41_@# MENU_RETURN
...
2z! MOUSE_0
2z" MOUSE_1
2z# MOUSE_2
2z$ MOUSE_3
...
```

The format of the *Terminal_Type*_keys file differs slightly for standard Environment and RWI terminal types.

## The *Terminal_Type*_Key_Names Package

The *Terminal_Type*_Key_Names package is an end-user convenience, providing easily readable key names in place of the symbols found in the *Terminal_Type*_Keys file.

The *Terminal_Type*_Key_Names package contains an enumeration type whose literals form the names of all R1000-supported keys for the terminal device's keyboard. (Note that some keys may not transmit and others may perform functions that are not supported by the R1000; consequently, it is not always true that every physical key on a keyboard will have a name used in the Environment.)

Part of the Xr6us_Key_Names package (for use with RXI on an IBM RS/6000) is shown here:

```
package Xr6us_Key_Names is

    type Key_Names is
       (
        Menu_Job_Kill, Menu_Job_Disable, Menu_Debug_Stop,
        Menu_Return,
        ...
        Backspace,       -- back space, back char
        S_Backspace, C_Backspace, M_Backspace, Cs_Backspace,
        Cm_Backspace, Ms_Backspace, Cms_Backspace,
        ...
        F1, C_F1, M_F1, Cm_F1, S_F1, Cs_F1, Ms_F1, Cms_F1,
        F2, C_F2, M_F2, Cm_F2, S_F2, Cs_F2, Ms_F2, Cms_F2,
        F3, C_F3, M_F3, Cm_F3, S_F3, Cs_F3, Ms_F3, Cms_F3,
        ...
        ' ', C_Space, M_Space, Cm_Space, S_Space, Cs_Space,
        '!', C_Exclam, M_Exclam, Cm_Exclam,
        '"', C_Quotation, M_Quotation, Cm_Quotation,
        '#', C_Sharp, M_Sharp, Cm_Sharp,
        '$', C_Dollar, M_Dollar, Cm_Dollar,
        ...
        );

end Xr6us_Key_Names;
```

The format of *Terminal_Type*_Key_Names package differs slightly for standard Environment and RWI terminal types.

## Key Binding

***Caution:*** *You should not make changes to the key-binding objects in !Machine-.Editor_Data, because this changes the systemwide key bindings, possibly preventing other users from logging in.*

### Procedure *Terminal_Type*_Commands

Each terminal type is associated with a *Terminal_Type*_Commands key-binding procedure. The body of *Terminal_Type*_Commands is a long case statement divided into the three sections: Interrupt, Prompt, and Execute. Nested case statements within each section bind keystroke sequences to R1000 system commands. This procedure is commonly called the *keymap* or the *key bindings* for a terminal type.

The format of the *Terminal_Type*_Commands procedure is the same for all terminal types. Part of the body of the Xr6us_Commands procedure (for use with RXI on an IBM RS/6000) is shown here:

```
with Access_List;
with Ada;
with Cmvc;
with Command;
with Common;
with Compilation;
with Debug;
with Editor;
with Io;
with Job;
with Library;
with Operator;
with Queue;
with Script;
with System_Utilities;
with Text;
with What;
with Xr6us_Key_Names;
procedure Xr6us_Commands is
    use Xr6us_Key_Names;
    type Intent is (Interrupt, Prompt, Execute);
    Action : Intent;
    Key1, Key2, Key3, Key4, Key5, Key6 : Key_Names;
begin
    case Action is
        when Interrupt =>
            case Key1 is
                when Menu_Job_Kill =>
                    Job.Kill (0);
                when Menu_Job_Disable =>
                    Job.Disable (0);
                when Menu_Debug_Stop =>
                    Debug.Stop (Name => "");
                when Scroll_Lock =>
                    Job.Disable (0);
                when C_Scroll_Lock =>
                    Job.Kill (0);
                when Cm_F1 =>
                    Debug.Stop (Name => "");
```

```
                when C_G | Cs_G =>
                    Job.Interrupt;
                when M_G | Cm_G | Ms_G | Cms_G =>
                    Job.Kill (0);
                when others =>
                    null;
            end case;
        when Prompt =>
            case Key1 is
                when Menu_Pick =>
                    case Key2 is
                        when 'G' =>
                            case Key3 is
                                when 'D' =>
                                    Text.Create (Image_Name => ">>IMAGE_NAME<<",
                                                 Kind => Text.File);
                                when 'E' =>
                                    Library.Create_Directory
                                        (Name => ">>DIRECTORY NAME<<",
                                         Kind => Library.Directory,
                                         Vol => Library.Nil,
                                         Model => "",
                                         Response => "<PROFILE>");
                                when 'F' =>
                                    Library.Create_World
                                        (Name => ">>WORLD NAME<<",
                                         Kind => Library.World,
                                         Vol => Library.Nil,
                                         Model => "!Model.R1000",
                                         Response => "<PROFILE>");
                                when others =>
                                    null;
                            end case;

                        when 'V' =>
                            Editor.Quit (Ignore_Changes => False);
                        when others =>
                            null;
                    end case;
                when Object =>
                    case Key2 is
                        when 'l' | C_L | 'L' | Cs_L =>
                            Common.Revert;
                        when others =>
                            null;
                    end case;
                when Image =>
                    case Key2 is
                        when '/' | '?' =>
                            Editor.Image.Find (Name => "");
                        when others =>
                            null;
                    end case;
                when Cm_F4 =>
                    Debug.Modify (New_Value => "",
                                  Variable => "<SELECTION>",
                                  Stack_Frame => 0);
. . .
end Xr6us_Commands;
```

## The *Terminal_Type*_User_Commands File

The *Terminal_Type*_User_Commands file is a skeleton of the *Terminal_Type*_Commands procedure and can be copied by users as the basis for custom key bindings. This file has the same overall structure as the *Terminal_Type*_Commands procedure described in the previous section. The Xr6us_User_Commands file (for use with RXI on an IBM RS/6000) is shown here:

```
with Xr6us_Key_Names;
procedure Xr6us_Commands is
    use Xr6us_Key_Names;
    type Intent is (Prompt, Execute, Interrupt);
    Action : Intent;
    Key_1  : Key_Names;
    Key_2  : Key_Names;
begin
    case Action is
        when Prompt =>
            case Key_1 is
                when others => null;
            end case;
        when Execute =>
            case Key_1 is
                when others => null;
            end case;
        when Interrupt =>
            case Key_1 is
                when others => null;
            end case;
    end case;
end Xr6us_Commands;
```

Note that the *Terminal_Type*_User_Commands file is a text file, not an Ada unit. To create your own key bindings from this template, you must parse the template into a custom *Terminal_Type*_Commands Ada unit in your home library. Refer to the Key Concepts tabbed section of the Session and Job Management (SJM) book for more information.

# Terminal-Type Definition and Recognition

## The Terminal_Types File

The Terminal_Types file contains definitions of terminal types used with RXI and RWI, and possibly other terminal-type implementations. (The Terminal_Types file does not define the Rational, Facit, or Vt100 terminal types; these types are defined internally in the Environment editor.) Definitions in the Terminal_Types file are used both during system elaboration and during user login.

The Terminal_Types file typically is created when RXI or RWI is first installed. During subsequent installations of RXI, RWI, or other terminal emulators, new terminal-type definitions are added to the Terminal_Types file.

***Note:** If RWI, RXI, or a similar terminal emulator is not installed on an R1000, the !Machine.Editor_Data world probably will not contain a Terminal_Types file.*

A typical Terminal_Types file is shown here:

```
Pc91                        Rational 66 80
Pc101                       Rational 66 80
Xhp46021a                   XRTERM 60 80
Xhp         : xhp46021a     XRTERM 60 80
XNCD                        XRTERM 70 80
XNews4                      XRTERM 77 80
XR6US                       XRTERM 64 80
XR6         : XR6us         XRTERM 64 80
XSun4                       XRTERM 79 80
```

Consider the last entry. The element `XSun4` is a terminal-type name associated with RXI. You can enter this terminal-type name in response to the `Enter Terminal Type` prompt. The terminal-type name is also associated with a set of key-binding objects located in the !Machine.Editor_Data world, and it is required when making custom modifications.

Terminal-type definitions have the following syntax:

*`Terminal_Type [:Aliased Type] [.Custom Keymap] [Output] [Lines [Cols]]`*

where:

- *:Aliased Type* is an optional element that names another terminal type defined earlier in the file. If this element is included, then the new terminal type is an alias for the key-binding portion of the previously defined terminal type and is associated with the same set of key-binding objects.
- *.Custom Keymap* is an optional element that names the prefix for a *Terminal_Type*_Commands custom key-binding procedure. Note that this element is needed only if the prefix for the custom *Terminal_Type*_Commands procedure differs from the prefix for the systemwide *Terminal_Type*_Commands procedure; by default, the Environment assumes that any relevant custom key bindings will have the same name as the systemwide key bindings.
- *Output* specifies which *output driver* manages data sent from the R1000 to the terminal device. The Environment supports the XRTERM (for use with RXI), Rational, Facit, and Vt100 output drivers.
- *Lines* specifies the number of character lines supported by the terminal. If this component is omitted, the default value is 24 lines.
- *Cols* specifies the number of character columns supported by the terminal. If this component is omitted, the default value is 80 columns. Note that if you specify the number of columns, you must also specify the number of lines.

## The Terminal_Recognition File

The installation of RXI or RWI creates a Terminal_Recognition file if this file does not exist. An entry for the new workstation or PC terminal type is added automatically. This entry corresponds to the terminal type defined in the Terminal_Types file.

The Rational, Facit, and Vt100 terminal types are not defined in the Terminal_Recognition file. The standard Environment is coded internally to recognize these three terminals. However, you can enter new entries based on these terminal types to provide an aliased type or custom keymap.

A typical Terminal_Recognition file is shown here:

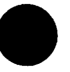
```
Pc86 [pc86c
Pc91 [pc91c
Pc101 [pc101c
Xhp46021a [[?1;66c
XNCD [[?1;42c
XNews4 [[?1;58c
XSun4 [[?1;18c
XR6US [[?1;74c
```

Each entry has the following syntax, in which the recognition sequence is limited to 32 characters:

*Terminal_Type Recognition sequencec*

Entries in the Terminal_Recognition file are read sequentially. If an entry later in the file specifies a recognition sequence identical to an earlier entry, then the entry appearing later in the file will override the earlier one. Consequently, when a terminal corresponding to the first entry attempts to log in, the R1000 will recognize it as a subsequent terminal type, which is incorrect.

If the terminal-recognition entry is missing for a particular terminal type, the Environment will query for a terminal type by displaying the `Enter Terminal Type` prompt.



# Glossary:
# Rational and DOD-STD-2167/2167A Terms

*Italicized* words are defined in this glossary. For definitions of terms not specific to the Rational Environment, refer to the glossaries in the *Reference Manual for the Ada Programming Language* and *Software Engineering with Ada* by Grady Booch.

DOD-STD-2167/2167A terms appear in a separate list after the Rational terms.

## RATIONAL TERMS

**!:** The symbol (pronounced "bang") used to refer to the *root world*, the topmost node of the Environment's *library* hierarchy. This symbol can also appear in a *window banner* to indicate that a *job* currently has access to the *object* displayed in the window.

**abandon:** To *release* an *object* and destroy its *image* without saving any changes.

**accept changes:** A *CMVC* term that refers to a means of automatically propagating changes made in one development path to another path.

**access control:** A set of mechanisms that restrict which Environment *objects* can be viewed and/or modified by users. See *access-control list*.

**access-control list (ACL):** The mechanism that specifies user access to specific objects in the Environment. Access lists specify *groups* and the access (Read, Write, Owner, Create, or Delete) that is granted to members of each group.

**account:** Defines a user to the Environment. An account provides the *system* with a record of essential information about a user and reserves a workspace for the user in the Environment *library* hierarchy.

**activity:** Defines mappings between *subsystems* and specific *releases* of that subsystem. Each subsystem entry can contain a reference to a *spec view*, a *load view*, or both. The referenced spec views specify a consistent semantic network of subsystem interfaces. The referenced load views specify which combination of implementation releases to collect into an executable system for testing.

**annotation:** A structured comment added to Ada *units* to form a *PDL*. Annotations capture additional information (often relating to design) that cannot be specified with Ada constructs alone.

**archive:** An Environment mechanism that permits *objects* to be saved on tape, or elsewhere on disk, for later retrieval if necessary.

**archived:** One of the four possible *states* of an Ada *unit*, in which the unit is stored with an *image* only. The archived state is available for the compact storage of units that are not of current interest. See also *coded*, *installed*, and *source*.

**argument:** A value supplied as a parameter to a subprogram or a value appearing after a *keyword* in a *PDL annotation.*

**assembler:** A *cross-development* facility tool that translates assembly-language source files into *object modules.* Assembly language is generated during the cross-compilation process and also can be written directly by users.

**attach:** See *connect.*

**attribute:** A characteristic of an *object.* For example, the attribute 'Body in the name Factorial'Body denotes the body of the object Factorial; Documentation'V(5) denotes *version* 5 of the object Documentation.

**background:** Describes the execution of a *job* that proceeds independently, permitting the user's *session* to initiate other jobs. When a user *disconnects* from a job, that job will continue execution in the background. See also *disconnect* and *foreground.*

**backup:** The result of the System_Backup.Backup operation that saves the *state* of the Rational Environment on magnetic tape. A full backup captures the entire system state. Primary backups preserve any changes since the last full backup, and secondary backups preserve changes made since the last primary backup. In emergencies, system state can be recovered by restoring a backup.

**banner:** A line of text displayed in reverse video that forms the bottom border of a *window.* The banner provides information about the *object* displayed in the window, such as its *name* and *class.* The definition of various symbols used in the banner can be found in Chapter 4, "Managing Windows," of the *Rational Environment User's Guide.*

**bind:** To establish a relationship between a key on the terminal keyboard and the execution of an operation or *command.* Operations can include *editor* keyboard *macros.*

**boot:** To initialize the machine so that users can *log in.*

**change analysis:** To calculate the effect of modifying an Ada *unit* or a component of an Ada unit. Specifically, the Environment computes the set of Ada units that must be recompiled if a particular change is made.

**check in:** To relinquish the right to modify an *object* under *CMVC* control. The *CMVC database* records differences between the object that was checked out and the object checked in.

**check out:** To obtain the right to modify an *object* under *CMVC* control.

**class:** A particular kind of *object attribute.* Every object in the Environment has a class and optionally may have a *subclass.* The three most common object classes are Ada, file, and *library* objects.

**client:** An entity that uses an interface or service. An Ada unit that "withs" a package specification might be called that package's client. Environment services can be used interactively by the user or indirectly by a user program; both are considered clients of the service. Individual *jobs* (such as *disk collection* and *snapshots*) that are run by the Environment *daemon* are also referred to as clients of the daemon. Finally, a *view* that *imports* a *spec view* from another subsystem is called a client of the imported spec view.

**CMVC:** Abbreviation for "configuration management and version control," the set of Environment tools that records modifications to *objects* under its control and automates the building of *system releases* from consistent sets of objects.

**CMVC database:** *Subsystem*-level *object* that collects information about modifications to *controlled objects* within the subsystem. Modifications to controlled objects result in new *generations* and historical records in the CMVC database.

**coded:** One of the four possible *states* of an Ada *unit*, in which the unit has executable code generated for it. See also *archived*, *installed*, and *source*.

**coded main program:** A subprogram that contains pragma Main and has been *promoted* to the *coded state*. Coded main programs are prelinked when they are promoted to the coded state. Thus, they retain completeness and elaboration information for use each time the program is executed.

**code-only:** Term applied to Ada *units* and *views* meaning that only executable code is stored for that *object*. Code-only objects require the minimum amount of storage necessary to permit execution of the Ada unit or view, but they do not allow modification, browsing of the implementation code, or full debugging. See also *loaded main program*.

**code section:** A specific kind of *control section* containing executable instructions. See also *control section*, *data section*, and *literal section*.

**code view:** Another term for a *code-only* view.

**combined view:** A single *view* that contains both the exported interfaces and the implementation of a *subsystem*. Combined views can be used in place of a *spec/load-view* pair, allowing nonhierarchical subsystem decomposition.

**command:** An Environment procedure that performs some operation or displays some information. A command can be invoked by way of a *command window*, by pressing a key, by pressing a mouse button, or by choosing a menu entry.

**command window:** A *window* attached to the bottom of an Environment window that is created by the user for the construction and execution of *commands*.

**commit:** The process of initiating some action. Committing an *object* saves its current *state*, for example. The phrase "committing a command window" occasionally is used to refer to the process of initiating execution of a *command* or a sequence of commands in a command window with the [Promote] key.

**compatibility:** Refers to whether a particular *spec/load view* pair in a *subsystem* can be executed together legally. Specifically, compatibility is the reconciliation of the Ada specifications in the spec view with the corresponding (and possibly different) specifications in the load view. Compatibility, for example, allows the private parts of *units* in the spec view to differ from those in corresponding load views.

**compilation unit:** An Ada term that refers to an independently compilable Ada construct. A compilation unit can be a subprogram declaration or body, package declaration or body, generic declaration, generic instantiation, *subunit*, or task body.

**completion:** The process whereby the Environment generates a syntactically or semantically correct *image* from a fragment of an Ada declaration or statement. Syntactic completion finishes many incomplete structures, *prompting* the user for additional information as required by syntactic rules of Ada. Semantic completion can be used to complete *names* and to prompt for parameters. Semantic completion uses the Ada name-resolution rules of the current *context* to *resolve* the name or name fragment and expands its definition if there is a unique match. Completing the name of a procedure, for example, expands the call to include prompts for all parameters.

**configuration:** A list of the specific *generations* of the *controlled objects* in a *subsystem* that compose a particular *view*. Together with the *CMVC database*, a configuration includes enough information to allow the reconstruction of any *released view*. Configurations are recorded in configuration objects.

**connect:** The process of bringing a *job* executing in the *background* to execution in the *foreground*. The job will then be the current job of a user's *session*.

**context:** The set of visible names within which the definition of a *name* is determined. For a name of a *command*, the context is provided by a user's current *searchlist*. For a *pathname* specified as a parameter to a *command*, the pathname is *resolved* (determined) relative to the pathname of the *current library*, which is determined by the *current context*. For Ada *units* in a given *library*, the context is determined by the set of Ada units in the library plus the library's *links* to Ada units in other libraries.

**controlled object:** An *object*, such as an Ada *unit*, that has been placed under the control of the *CMVC system* and requires *checkout* and *checkin*.

**control section:** Code generated for *targets* other than the R1000 is divided into one of three control sections: (1) *code sections* containing executable instructions; (2) *read-only literal sections*; and (3) read/write *data sections*. One or more control sections constitute an *object module*.

**crash:** The failure of the machine to continue normal execution. A failed machine must be re*booted*.

**cross-development:** The process of developing software on a computer other than the one intended for final execution of the application. Cross-development includes the design, coding, and initial testing of software in the Rational Environment, followed by the generation of executable code for final testing on the *target* computer.

**current:** Applies to entities such as *context*, *images*, *objects*, *selections*, *activities*, *sessions*, and *switches*; denotes that the corresponding entity is used as the *default* for the *command* being executed. Often, the current entity is determined by the position of the *cursor*.

**current context:** "Where you are" in the Environment. If the *current window* contains an Ada *unit*, that Ada unit is the current context. Otherwise, the current context is the nearest *library* that is or encloses the object in the current window.

**Current_Error:** A redirectable location to which *jobs* can write error messages or other *output*. Users can specify that Current_Error be a text file, the *message window*, or an *I/O window*. By default, Current_Error is the same as *Standard_Error*, which is the message window. See also *Standard_Error*.

**Current_Input:** A redirectable location from which *jobs* can read input. Users can specify that Current_Input be a text file, the *message window*, or an Environment *I/O window*. By default, Current_Input is the same as *Standard_Input*, which is an I/O window. See also *Standard_Input*.

**current library:** The nearest *library* that either is or encloses the *current context*. (Whereas the current context is either an Ada *unit* or a library, the current library is always a library.) *Pathnames* entered as parameter values are *resolved* in the current library.

**Current_Output:** A redirectable location to which *jobs* can write output. Users can specify that Current_Output be a text file, the *message window*, or an *I/O window*. By default, Current_Output is the same as *Standard_Output*, which is an I/O window. See also *Standard_Output*.

**current window:** The *window* containing the *cursor*; if the cursor is in a *command window*, the current window is the window to which the command window is attached.

**cursor:** A visible symbol that marks a specific point on the *screen*. The placement of the cursor often identifies the place or *object* on which some action will be performed.

**daemon:** A *background job* that acts as a custodian for the Environment's data structures, such as *disk collection*. The periodic execution of the individual *clients* of the Environment daemon is scheduled by the system operator.

**data section:** A specific kind of *control section* containing read/write data. See also *code section*, *control section*, and *literal section*.

**debugger:** A Rational Environment facility that permits a user to closely control and query the *state* of a running program as an aid in testing.

**default:** A predefined value that is used in a *command* or operation if a user does not explicitly provide another value.

**Definition command:** A fundamental operation for *traversing* the Environment. Given a reference to a *name*, this *command* displays a *window* containing its defining occurrence or "declaration." For example, the definition of a file listed in a *library* brings up a window containing an *image* of that file; the definition of the identifier Text_Io.File_Type in an Ada unit brings up a window highlighting the corresponding declaration of the limited private *type* File_Type in the package Text_Io. Traversal between Design Facility documents and *PDL* also can be accomplished with the Definition command.

**delete:** To remove an *object*'s *default version* from the *library* system. If the *retention count* for the enclosing library is not zero, deleted versions are retained and can be undeleted with the Library.Undelete *command*. See also *destroy* and *expunge*.

**demote:** To bring an Ada *unit* or set of units to their next lower *state*. For example, demoting a *coded* unit discards its generated code and brings the unit to the *installed* state. Demoting an installed unit brings it to the *source* state. Demotion of a unit from the installed state can proceed only if *change analysis* determines that no other units would be *obsolesced* by this change.

**design rule:** An addition to the compilation system that further constrains the definition of a legal Ada program. Design rules can either require the existence of certain *annotations* or prohibit the use of certain Ada constructs.

**destination name:** The *name* of an *object* that results from a move or copy operation. The destination object is a copy of a *source* object; furthermore, move operations delete the source object after the copy is made.

**destroy:** To *delete* and *expunge* all *versions* of an *object*. A destroyed object has no entry in the *library* system and cannot be recovered. See also *delete* and *expunge*.

**detach:** See *disconnect*.

**device:** A *class* of Environment *objects* denoting tapes, terminals, printers, and other such I/O hardware.

**DIANA:** Acronym for "descriptive intermediate attributed notation for Ada"; a tree-based representation for Ada programs that is used internally by the Environment.

**directory:** A *subclass* of *libraries*. Directories can contain any *object*, including other libraries. Directories, unlike *worlds* (another library subclass), do not have

*links*, *access lists*, or *target keys* associated with them; they inherit these characteristics from the enclosing world.

**disconnect:** The process of placing a *foreground job* in the *background*. A disconnected job proceeds independently, allowing the user to initiate new jobs.

**disk collection:** Some Environment operations allocate space to store temporary information that is not immediately recovered after the operation terminates. Disk collection is the systematic process of identifying all allocated space that no longer has any references and marking it for reuse.

**DOD-STD-2167/2167A:** See the list of terms at the end of this glossary.

**downloader:** A *cross-development* facility tool used to transfer an *executable module* linked on the Rational Environment to the actual *target* hardware—for example, a *MIL-STD-1750A* or 68020 processor. The downloader typically is customized for each particular target.

**edit:** To modify an *object*; the Environment will not allow editing if that object is *read-only*, if the *access control* associated with that object does not specify write access, if the object is controlled and not checked out, or if the change would *obsolesce* other objects, requiring a consistent set of changes across several objects.

**editor:** The facility through which the user interacts with the Rational Environment. All Environment operations are performed through interaction with the Rational editor. Both simple text editing and *object*-specific editing operations are supported. There are many kinds of objects in the Environment, and many have a well-defined internal structure. The editor assists the user in constructing legal objects through its knowledge of an object's structure and the values it can have.

**elaboration:** An Ada term defining the process by which a declaration is brought into existence and is assigned initial value, if any. Elaboration normally takes place during program execution.

**enclosing object:** The *object* that immediately surrounds a particular object. For example, an Ada *unit* is enclosed by a *library* (which may be further enclosed by another library, and so on). The enclosing object of a *subunit* is the Ada unit (often a package body) in which the subunit's *stub* is declared.

**Environment:** Short name for the Rational Environment. The Environment is the composite of the mechanisms, tools, and facilities that allow a user or a team of users to develop Ada software.

**executable module:** A binary file produced by the *linker*. Several *object modules* are linked together to form an executable module that is ready for *target* execution.

**exports:** The Ada *units* that are made visible outside the *subsystem* to other *client* subsystems. A *spec view* or *combined view* contains a subsystem's exports.

**expunge:** To remove obsolete *objects*, such as nondefault *versions* of text files and Ada *units*, mail messages that have been marked for deletion, and *links* that reference deleted objects. Expunged objects cannot be referenced or undeleted. See also *delete* and *destroy*.

**foreground:** Denotes that the execution of a *job* proceeds synchronously with the user *session* that initiated it, preventing that session from initiating other jobs before the current job completes execution. See also *background* and *connect*.

**Format command:** A *command* that results in syntactic checking and possible *completion* of the *current* Ada *unit.*

**frame:** The *screen* space occupied by a *window.* The available screen is divided into several frames, within which windows can be placed.

**freeze:** To preserve the *state* of an *object* so that no further changes can be made. If necessary, frozen objects can be unfrozen subsequently, allowing changes to be made.

**FTP:** Abbreviation for "file transfer protocol"; FTP is a standard file transfer protocol established by the U.S. Department of Defense.

**fully qualified name:** An unambiguous name of an *object,* consisting of its simple name prefixed by the fully qualified name of the object that contains its definition. Fully qualified *pathnames* begin at the *root world* (*!*), appending *subdirectory* names separated with the period (.) character, as in !Users.Operator.Login.

**garbage collection:** See *disk collection.*

**generation:** Each time an *object* is checked out, a new generation of that object is created in the CMVC database. Generations are distinct from *versions,* which are created when *units* are opened for editing. See also *check in* and *check out.*

**group:** A list of users. Specific access to *objects* within the Environment is granted to groups placed on the *access list* of an object.

**history:** The recorded *state* of an *object,* maintained in the order in which each new state was saved. The *CMVC* system records history for *controlled objects,* including who modified the unit, when it was modified, what modifications were made, and why the unit was modified.

**home library:** Also called *home world.* The library associated with a user's *account.* A user's *sessions* are declared in the user's home library.

**host/target development:** A development methodology in which software is created and maintained on the host (an R1000) and executed and debugged on another *target* machine (not necessarily an R1000).

**image:** The user-readable textual representation of an *object.* A *window* displays a portion of the entire image of an object at any one time.

**imports:** A list of *spec views* or *combined views* referenced by a given *subsystem.*

**I/O window:** Also called *input/output window* or *output window.* An Environment *window* reserved for displaying *job output* and for *prompting* for job input. The *image* in an output window is not associated with an *object* and therefore cannot be *edited,* with the exception of providing input specifically requested by a job.

**incremental compilation:** The checking of an Ada *unit* for conformance with the rules of the *Reference Manual for the Ada Programming Language,* at the granularity of a declaration or statement. Incremental compilation allows additions, deletions, and modifications to be made with the minimum amount of compilation necessary to perform that change legally.

**indirect file:** A text file that contains one or more *pathnames.* When the *name* of the indirect file is preceded by an underscore and given as a parameter value, the *Environment* converts the file's contents into *set notation.* Indirect files are useful for referencing the same set of *objects* in multiple *commands* or in commands entered multiple times.

**insertion point:** The place in an *installed* or *coded* Ada *unit* at which a user wants to insert a new declaration, statement, or comment. When an insertion point is created, a *prompt* designates the place at which the new declaration or statement will reside. Insertion points also can be created in *libraries* when new Ada *objects* are created.

**install:** The process of *promoting* an Ada *unit* from *source* to *installed state.*

**installed:** One of the four possible *states* of an Ada *unit.* An installed unit is semantically correct and is available for use by other Ada units (for example, other units can "with" an installed unit and be checked for semantic correctness). See also *archived, coded,* and *source.*

**instruction-level simulator:** A *cross-development* facility tool hosted on the R1000 that simulates the execution of a program on a *target* machine. An instruction-level simulator executes the user's program by mimicking the behavior of each target-machine instruction.

**item:** An entity that can be modified through an editing operation. Words, lines, *regions,* entire *images,* and *windows* are examples of such items.

**job:** A numbered process that acts as the thread of control for an Environment operation. A job consists of one or more *commands* that are executed together. A job is initiated each time you *edit* an *object* or execute a command, using a *command window,* a key combination, a mouse, or a menu. Commands in a job can *spawn* additional jobs.

**job identification number:** Also called "job id" or "job number." A number in the range 0..255 that uniquely identifies a job. A job's identification number is assigned by the Environment when the job begins.

**job identity:** The name of the user and *session* under which a *job* was started and with which the job is associated. A job's identity is used to determine the *objects* to which the job is permitted access and the session from which the job is to obtain its initial *response characteristics.*

**job output:** Anything written by a *job* to the screen or to a file. Job output is divided into two kinds: *job reports* and *logs.*

**job report:** A kind of *job output.* A job report typically displays information specifically requested by the user. It also indicates errors encountered by interactive *commands,* such as those in packages Ada, Common, and Editor.

**job response profile:** The *response profile* associated with a job. The job response profile for a job defines the default *response characteristics* to be used by all commands executed as part of that job. The job response profile is indicated by the special value <PROFILE>. See also *response profile, session response profile,* and *system default profile.*

**join:** Allows synchronized changes to multiple copies of the same *object* located in several development *paths.* When objects are joined, they share a single *reservation token* to ensure that only one copy of an object is modified at a time.

**key binding:** The mapping of an operation to a key sequence. See also *bind.*

**keyword:** Relating to Ada, this term means an Ada-reserved word such as "begin," "if," or "package." As an Environment term, it specifies the part of an *annotation* that indicates what kind of annotation it is. A keyword might also be thought of as the name of the annotation, such as "@PURPOSE" or "@SUMMARY."

**kill:** To terminate the execution of a *job* before its normal completion.

**library:** A *class* of Environment *objects* used to partition objects into a hierarchical structure analogous to directory systems on other computer systems. Libraries can contain any kind of object, including Ada *units*, files, and other libraries. Libraries can be *worlds*, *directories*, *subsystems*, or *views*, each a *subclass* of the library class. (Library objects have the same general semantics as required by the *Reference Manual for the Ada Programming Language.*)

**linker:** A *cross-development* facility tool that binds a program's *object modules* into one *executable module*, making it ready for *target* execution. The linker can be used to organize software for optimal execution on the target hardware, specifying, for example, exact locations for code and data in memory.

**links:** Links govern the visibility of Ada *units* within a *world*. External links are the mechanisms by which Ada units external to a world are imported or made visible inside that world. Internal links provide internal visibility to *objects* within a world. Links are implemented as a mapping of a simple Ada *name* to a *fully qualified* pathname of an Ada unit.

**literal section:** A specific kind of *control section* containing *read-only* literal and constant data. See also *code section*, *control section*, and *data section*.

**loaded main program:** A special kind of Ada *unit* containing prelinked code ready for execution. Similar to program files on other systems, loaded main programs are referred to as *code-only* units. Loaded main programs are of *subclass* Load_Proc or Load_Func.

**Load_Proc:** See *loaded main program*.

**load view:** An implementation of a *subsystem*. A subsystem can have several load views, each representing a different, perhaps evolutionary, implementation of the subsystem. Frozen load views are referred to as *releases*.

**lock:** A mutually exclusive hold on an *object* that restricts how *clients* can manipulate that object. Clients can acquire two forms of locks, read and write. While a client (for example, a user) has a read lock on an object, other clients can read that object, but no clients can write into it. While a client has a write lock on an object, no other client can read it or write into it.

**log:** A kind of *job output*, logs are a sequence of progress messages generated by many Environment *commands*. A command's *response characteristics* govern what information is included in a log. Logs can be temporary (that is, they exist only for the duration of a session), or they can be redirected, appended, copied, or filtered into permanent files.

**log in:** The process of validating a username and a password to permit further interaction with the Environment. A user's *home library* may contain a Login procedure that is invoked when the user is granted entry to the Environment. A Login procedure overrides the default Environment login.

**log out:** The process of leaving a *session* and suspending interaction with the Environment.

**macro:** An ordered sequence of keystrokes that can be executed as a whole. A macro can be bound to a key. See also *bind*.

**mark:** A record of a position (line and column) within an *image* of an *object*. The user can set a mark and return to it at some later time.

**menu:** A kind of Environment *window* that lists a series of *pathnames* of Environment *objects*. The Show_Usage *command*, for example, may list a menu of objects

that use a particular Ada object or declaration. Several operations, including the *Definition command,* can be performed on each pathname listed in a menu.

**merge:** A *CMVC* term defining the process of consolidating multiple changes to *objects* in different development *paths.*

**message window:** A predefined *window* at the top of the *screen* that displays *system* and error information from the Environment.

**MIL-STD-1750A:** A DOD standard defining a computer architecture (instruction set) for use in real-time applications.

**model:** An Environment *library* that is used during the creation of *worlds* and new *subsystem views.* A model defines the *links, switches, target key,* and other predefined *attributes* of the new view.

**name:** An identifier that either defines a new entity or is a reference to the definition of some entity; a label by which an entity can be referenced.

**nickname:** A *name* that may be associated with an *overloaded* Ada declaration. Nicknames support the construction of unique *pathnames* that differentiate one declaration from other declarations with the same overloaded name. Nicknames are specified by the Nickname pragma and referenced with the nickname *attribute* ('N(Nickname)).

**numeric keypad:** The set of keys to the right of the main portion of the keyboard that is used for specifying the number of times an operation should be performed.

**object:** An entity in the Environment that has *state.* An Environment object may be an Ada *unit, library* (*world* or *directory*), file (text file, *switch,* or *activity*), *session,* user, tape, terminal, or *pipe.*

**object editor:** A term that is sometimes used to refer to the mechanism that allows structured editing of *objects.* See also *edit* and *editor.*

**object module:** A binary file produced by an *assembler* that contains both code and data. The *linker* binds several object modules together into a single *executable module.*

**obsolesce:** The removal of the semantic validity of an Ada *unit* or set of Ada units. Certain changes to Ada units will make obsolete other dependent Ada units in the Environment, requiring the dependent units to be *demoted* from a semantically valid *state* (*installed* or *coded*) to the *source* state.

**operator capability:** Special access that allows users to perform system-management functions such as creating and deleting user accounts. This access initially is granted to the user Operator, but it also can be granted to other users.

**output window:** See *I/O window.*

**overloading:** An Ada term that denotes the ability to form declarations with the same name but with different specifications.

**path:** A logically connected series of *views* in a *subsystem.* A subsystem can contain multiple paths—for example, an application that is intended for multiple *targets* can be structured with one path per target. Corresponding *units* in different paths can (but need not) be *joined.*

**pathname:** A *name* reference to an *object* in the Environment. *Fully qualified* pathnames begin at the *root world* (*!*), appending *subdirectory* names separated with the period (.) character, as in !Users.Operator.Login. Pathnames also can be specified relative to the current *context.*

**PDL:** Abbreviation for "program design language," which is a combination of Ada, *annotations*, and *design rules* that formally define the design of an Ada program.

**pipe:** An *object* supporting asynchronous reading and writing of ordered (first in, first out) message streams. Pipes are especially useful for inter*job* communication.

**placeholder:** A string enclosed in ">> <<" characters that appears as a *default* value for many *command arguments*. A placeholder indicates the kind of information required from the user for that argument.

**predefined library package:** Any of several packages required by the *Reference Manual for the Ada Programming Language*, such as Text_Io, System, and Calendar.

**pretty-printer:** The Environment mechanism that takes the representation of an *object* and produces a user-readable *image* for display in a *window*. The image of an Ada *unit* is pretty-printed from its *DIANA* representation.

**preview document:** A document containing consolidated design information extracted from the *PDL* for an Ada application. Preview documents list design information in paragraph form and allow traversal to and from the Ada *units* containing the *PDL* and other documents.

**primary subsystem:** A modifiable *subsystem* in which ongoing development can proceed. In a multi-R1000 application, one R1000 contains the primary subsystem and copies of it (called *secondary subsystems*) are distributed to the other R1000s.

**privileged:** A predefined *group* whose members can bypass *access control* with the Operator.Enable_Privileges *command*.

**profile:** See *response profile*.

**promote:** To bring an Ada *unit* to its next higher *state*. For example, promoting a *source* unit brings it to the *installed* state (assuming there are no syntactic or semantic errors). Promoting an installed unit generates code for that unit, bringing it to the *coded* state. Promoting a coded procedure executes that unit. Promoting a *command window* executes that command window.

**prompt:** A reverse video region of text, provided by the *editor*, that denotes an incomplete program or command fragment awaiting user action. A prompt is used as a placeholder for text, and it disappears when text is entered on it.

**quarter-plane:** Describes how *windows* display *images*. The origin of an image is the first column of the first row, and it can extend an arbitrary distance to the right and down. A window displays a contiguous portion of an image.

**queue:** An Environment-maintained structure that maintains an ordered set of *objects* designated for transmission to a *device* such as a printer.

**quit:** The *command* used to terminate interaction with the Environment. This term is synonymous with the term *log out*, often used by other computer systems.

**read-only:** Describes an *object* that can be *viewed* and *traversed* but not modified. A read-only object is indicated by an equals (=) sign in a *window banner*. A user must gain write access to an object to modify it.

**redo:** The process of stepping forward through the sequence of *states* saved by the *editor* during the evolution of an *object*.

**region:** A contiguous area of text in an *image*.

**release:** This term has two different meanings. In reference to editing *objects*, it means to remove a *lock* (read or write access) on an object. In reference to proj-

ect development using Rational *Subsystems*, the term refers to a specific *view* of a subsystem or a particular implementation that is ready for integration and testing.

**released view:** A frozen *view* created from a *working view*. Combinations of released views from all *subsystems* within an application can be tested together to determine a project-level release.

**remote-passwords file:** A text file containing the names of remote machines and the usernames and passwords to be used for accessing those machines from the current machine. Remote-passwords files are used by *commands* that perform operations across a network, such as those in package !Commands.Ftp.

**remote-sessions file:** A text file containing the names of remote machines and session (or account) to be used for accessing those machines from the current machine. Remote-sessions files are used by *commands* that perform operations across a network, such as those in package !Commands.Ftp.

**reservation token:** Associated with each *controlled object* or set of *joined* objects. The reservation token is obtained when an object is checked out and returned when the object is checked in. See also *check in* and *check out*.

**resolve:** To determine the meaning of a *name* within the current *context* by looking up the name in the list of currently visible names.

**response characteristics:** A set of values that control the execution of a *command*. A command's response characteristics tell the command how to respond to errors, where to direct the *logs* that summarize command behavior, how to format and filter those logs, and where to find the correct *activity*, *remote-passwords file*, and *remote-sessions file* to use. Most Environment commands obtain their response characteristics from a Response parameter. See also *response profile*.

**response profile:** A prepackaged set of *response characteristics* that are stored by the Environment as an aggregate of component values. These component values are set initially by the Environment and can be changed by users. See also *job response profile*, *session response profile*, and *system default profile*.

**restore:** The act of replacing an *object* or set of objects from a *backup* or *archive* of those objects.

**retention count:** The number of older (*deleted*) *versions* of an *object* that are kept by the directory system. These deleted versions can be recovered if necessary. When a new object is created, versions older than the retention count are *destroyed* and cannot be recovered.

**revert:** To return an *object's state* to an earlier *version*.

**root world:** The topmost node in the Environment *library* hierarchy, denoted by the symbol *!* (pronounced "bang").

**RPC:** Abbreviation for "remote procedure call," an Environment mechanism that supports the initiation of operations that are to be accomplished on some other remote machine. Results from the remote operation's execution are returned to the initiating machine.

**screen:** The visible display area on a terminal. The screen is divided into some number of *frames* (the *default* is three), in which *windows* can be placed.

**scroll:** To change the portion of the *image* that is currently visible in a *window*; both horizontal and vertical scrolling are possible.

**searchlist:** A list of *libraries* that are searched during the resolution of *command* names. A searchlist defines an ordered list of places to look for a command's defi-

nition and thus determines what *commands* are visible and therefore executable from a command window. One searchlist is associated with each *session.*

**secondary subsystem:** A copy of a *primary subsystem* in a multi-R1000 application. Secondary subsystems are distributed from the R1000 containing the primary subsystem to all other R1000s in the project. Changes should not be made to a secondary subsystem, except to make it consistent with its primary.

**selection:** A highlighted *image* of an *object* or part of an object. Many *commands* recognize a selection only when the *cursor* is inside the selected text.

**session:** The *state* of a user's interaction with the Environment from *login* to *logout.* Some state (session *switches*) remains between logins. That state is associated with a session *object* in the user's home *library.* Note that a given user can have multiple sessions, one for each kind of work being done.

**session response profile:** The *response profile* associated with a *session.* The session response profile defines the initial *job response profile* for all *jobs* started under that session, and therefore defines the *default response characteristics* to be used by all *commands* executed under that session. The session response profile is indicated by the special value <SESSION>. See also *response profile, job response profile,* and *system default profile.*

**set notation:** A means of supplying a list of *pathnames,* enclosed in brackets, as a single parameter value. See also *indirect file.*

**sever:** The opposite of *join.* Severing joined *objects* allows them to be modified independently.

**simple name:** An unqualified, unattributed identifier associated with the declaration of an entity. It is the rightmost component of a *pathname* after any *attributes* (for example, 'Body or 'V(*N*)) are removed.

**snapshot:** An Environment *job* that periodically captures the *state* of the Environment. If the machine fails (*crashes*), all changes saved (*committed*) up to the last snapshot will be restored when the machine is re*booted.*

**source:** One of the four possible *states* of an Ada *unit,* in which neither syntactic nor semantic correctness is assured, and in which arbitrary textual modifications can be made. See also *archived, coded,* and *installed.* In discussions of move and copy operations, source refers to the original *object* from which the copy, or *destination* object, is made.

**spawn:** To create a new instance of an entity. This term is used in two specific ways in the Rational Environment. The first refers to the initiation of a separately executing *job* by another currently executing job. The second refers to the creation of new *views* of a *subsystem.*

**special name:** A name enclosed in "< >" characters that appears as a *default* value for many *command arguments.* It indicates an action that is predefined in the Rational Environment. For example, the special name <CURSOR> represents the object located under the *cursor* position in the *current context.*

**spec view:** The exported interfaces that a *subsystem* provides for import and use by other subsystems. A subsystem can have multiple spec views corresponding to different *releases* of that subsystem's interfaces.

**Standard_Error:** A location to which *jobs* can write error messages and other *output.* In most cases, Standard_Output is the *message window* and cannot be changed. See also *Current_Error.*

**Standard_Input:** A location from which *jobs* can read input. In most cases, Standard_Input is the *I/O window* and cannot be changed. See also *Current_Input.*

**Standard_Output:** A location to which *jobs* can write *output.* In most cases, Standard_Output is an *I/O window* and cannot be chagned. See also *Current_Output.*

**state:** This term has two meanings depending on the *class* of *object* to which it is applied. For Ada *units,* it refers to the amount of compilation that has been applied to a unit (that is, *source, installed,* and *coded* states). For other objects, it refers to the current value of the object. An object being edited will have a constantly changing state. State can be permanently saved by *committing* that object.

**stub:** In the Ada sense, the declaration of a separate *unit* (*subunit*). The *image* of an *object's* declaration in a *library* can also be referred to as a stub.

**subclass:** A finer division of an *object's class.* For example, within class Library are subclasses such as World, Directory, Subsystem, and so on.

**subpath:** A set of *views* that allow multiple developers to work in parallel within a single *subsystem.*

**subsystem:** A logical collection of *objects* that defines part of a system. Subsystems can be used to decompose large software systems. Analogous to Ada packages, subsystems have both specifications (called *spec views*) and implementations (called *load views*).

**subunit:** An Ada term; the actual *object* or Ada *unit* that is denoted by a *stub* in a parent unit's body.

**switch:** A system parameter that affects the behavior of Environment operations. The values of many switches are stored in files and are consulted implicitly in many *Environment* operations. Switches can affect the operations of an entire *session,* or they can be limited in scope to a specific *library.*

**system:** In *CMVC,* the term specifies a group of *subsystems* that form a higher-level partition of an Ada software project. Systems define hierarchical relationships between subsystems and form the basis for defining project-level *releases.* Also used in reference to the R1000 computer system.

**system default profile:** The *response profile* defined by the *system.* The system default profile defines the initial *session response profile* for each *session.* The system default profile is indicated by the special value <DEFAULT>. See also *response profile, job response profile,* and *session response profile.*

**system manager:** The person designated to maintain *system* operation. The system manager is often the person who takes a system *backup* and transmits problem reports to Rational.

**target:** The computer on which application software is intended to execute.

**Target Build Utility:** An Environment mechanism that supports *host/target development* by automatically moving program components from a Rational *system* to a *target* system.

**target key:** Specifies both the *target* for which code should be generated and the particular *PDL* that should be enforced.

**TCP/IP:** Abbreviation for "transmission control protocol and Internet protocol"; TCP/IP is a standard transport layer established by the U.S. Department of Defense.

**traverse:** To move from one place in the Environment to another place. The *Definition command* is often used to traverse among *libraries* and *objects* in libraries. In an Ada program, the Definition command allows you to traverse from a reference to the defining occurrence of an object.

**type:** An Ada term; characterized by a set of values and a set of operations appropriate to *objects* of that type.

**undo:** The process of stepping backward through the sequence of *states* saved by the *editor* during modification of an *object.*

**unit:** See *compilation unit.*

**unit state:** See *state.*

**universe:** The name that often is used to refer to the *root world* (topmost node) of the Environment *library* hierarchy.

**venture:** An *object* that defines work tasks and policies associated with a project or part of a project. A venture serves as a template for creating *work orders.* Through a venture, a project manager specifies the kinds of information that will be collected by the work orders.

**version:** A numbered instance of an *object*; lower-numbered versions indicate earlier instances of an object.

**view:** A specific instance (or *release*) of a *subsystem* specification or implementation. There are three kinds of views: *spec views, load views,* and *combined views.*

**wildcards:** A set of special characters that represent some pattern of text. Naming wildcards can be embedded in *pathnames* to denote an *object* or set of objects without providing *fully qualified names.* Pattern-matching wildcards can be used in search commands from packages Editor.Search and File_Utilities to define a pattern (regular expression) that strings must match in order to be found.

**window:** A mechanism that displays some part of an *image.* There are three kinds of windows: major windows for displaying *objects, command windows* for entering *commands,* and minor windows for performing *incremental compilation* operations. Major windows are placed by the Environment in *frames* or sections of the terminal *screen* area. Command windows and minor windows can share the frame with the major window to which they are attached.

**window directory:** A special *window* containing a list of all windows in a *session.*

**withdraw:** To remove the declaration of an *object* from a *library.* An *insertion point* remains so that the object can be replaced at some future time.

**working view:** A *view* designated for making ongoing changes to a *subsystem.* By convention, working views are identified by "_Working" in their names. *Objects* in a working view can be baselined by creating a *released view* or a *configuration* object.

**work order:** An *object* that defines and collects information about a specific task within a particular project. Project managers can use work orders to assign tasks to individual developers. Specific work orders are created from a template defined by a *venture.*

**world:** A *subclass* of *libraries.* Worlds have *links* and *access lists* associated with them. Visibility to *objects* in a world can be achieved only by explicitly establishing a link to a specific object. In this sense, worlds can be considered to have a closed scope.

## DOD-STD-2167/2167A TERMS

**CDR:** Critical Design Review. A *design review* that occurs at the end of the *Detailed Design* phase. The purpose of the review is to ensure that the design is complete and consistent and that the project is ready to move to the *Coding and CSU Testing* phase. The *SDD* (2167A) or *SDDD* (2167) is delivered and reviewed during the CDR.

**Coding and CSU Testing:** A 2167A *lifecycle phase* following *Detailed Design*, during which CSUs are coded and tested. Called "Coding and Unit Testing" in 2167.

**CSC:** Computer Software Component. A 2167A subcomponent of a *CSCI*, further classified as *TLCSC* or *LLCSC* in 2167. CSCs are the only *software components* that can be decomposed into software components of the same kind.

**CSC Integration and Testing:** The 2167A *lifecycle phase* following *Coding and CSU Testing*, during which *CSUs* are integrated and tested to see whether they meet specified requirements.

**CSCI:** Computer Software Configuration Item. A major *software component* of a *Segment*; a separately deliverable collection of software.

**CSCI Testing:** The 2167A *lifecycle phase* following *CSC Integration and Testing*, during which CSUs are integrated and tested to see whether they meet specified requirements.

**CSU:** Computer Software Unit. A 2167A *software component* and a subcomponent of a *CSC*. It is the smallest component in a *CSCI* that is not further decomposed. A CSU is coded and tested independently of other CSUs.

**DBDD:** Data Base Design Document. The 2167 document that describes the design of any databases in a *CSCI*. This information can be incorporated into the *SDDD*. The DBDD is reviewed during the *CDR*.

**design review:** The review held at the end of the current *lifecycle phase*, at which the design is reviewed by the contracting agency for conformance to requirements and constraints. Specific design reviews are the *SRR*, *SSR*, *PDR*, and *CDR*.

**Detailed Design:** The *lifecycle phase* following *preliminary design*, during which the structure of the *CSCI* is finalized down to the *CSU* level.

**DID:** Data Item Description. The specification of a deliverable document produced under *DOD-STD-2167* or *DOD-STD-2167A*. Each type of document has its own DID.

**DOD-STD-2167 (2167):** A software procurement standard mandated by the U.S. Department of Defense that defines *lifecycle phases*, *design reviews*, *software components*, and documents to be used by the contractor in developing software.

**DOD-STD-2167A (2167A):** An updated standard superseding DOD-STD-2167. All software designed for the U.S. Department of Defense after February 1988 must follow 2167A rather than 2167.

**HWCI:** Hardware Configuration Item. A hardware component of a *Segment*; a separately deliverable piece of hardware.

**IDD:** Interface Design Document. The document that describes the design of the external interfaces of each *CSCI*.

**IRS:** Interface Requirements Specification. The specification that describes a preliminary set of interface requirements for all *CSCI* external interfaces.

**IV&V:** Independent Verification and Validation. A *design review* to determine whether the system meets all requirements. It is performed by a contractor or government agency not responsible for developing the product or performing the activity being evaluated.

**lifecycle phase:** A discrete part of the software-development process, such as *Software Requirements Analysis* or *Preliminary Design.* Each phase is characterized by milestones and deliverables.

**LLCSC:** Low-Level Computer Software Component. A 2167 subcomponent of a *TLCSC.* LLCSCs are the only *software components* that can be decomposed into software components of the same kind.

**Maintenance:** The *lifecycle phase* following formal delivery of a *CSCI,* during which the software is updated or changed as needed.

**NDS:** Non-Developmental Software. Deliverable software that is not developed under the contract; it may be reusable software or vendor-supplied software.

**PDR:** Preliminary Design Review. A *design review* of the preliminary *SDD* (2167A) or *STLDD* (2167) that occurs at the end of the *Preliminary Design* phase. The purpose of the review is to ensure that the design is complete and consistent and that the project is ready to move to the *Detailed Design* phase.

**Preliminary Design:** The *lifecycle phase* following *Software Requirements Analysis,* during which the overall structure of the software down to the *CSC* level is developed.

**SDD:** Software Design Document. A preliminary version of the 2167A SDD is a product of the *Preliminary Design* phase and is required as a deliverable at the *PDR*; the final version of the SDD is a product of the *Detailed Design* phase and is required as a deliverable at the *CDR.*

**SDDD:** Software Detailed Design Document. A 2167 document that describes the design of a *CSCI* down to the Unit level. The SDDD is a product of the *Detailed Design* phase and is required at the *CDR.*

**Segment:** A major piece or subsystem of a *System* composed of *HWCIs* and *CSCIs.* Usually all HWCIs and CSCIs of a Segment are delivered together.

**software component:** An entity that is used to describe the structure of software under 2167A and 2167.

**Software Requirements Analysis:** An early *lifecycle phase* following *System Requirements Analysis/Design,* during which final engineering requirements are defined for each *CSCI.*

**SRR:** System Requirements Review. A *design review* that occurs at the end of the *System Analysis* phase. The purpose of the review is to ensure that the system requirements are complete and consistent and that the project is ready to move to the *Software Requirements Analysis* phase.

**SRS:** Software Requirements Specification. A document describing the requirements of a *CSCI.* The SRS is a product of the *Software Requirements Analysis* phase and is required at the *SSR.*

**SSR:** Software Specification Review. A *design review* that occurs at the end of the *Software Requirements Analysis* phase. The purpose of the review is to ensure that the SRS is complete and consistent and that the project is ready to move to the *Preliminary Design* phase.

**STLDD:** Software Top-Level Design Document. A 2167 document that describes the design of a *CSCI* down to the *CSC* level. The STLDD is a product of the *Preliminary Design* phase and is required at the *PDR*.

**System:** The entire application, including both hardware and software.

**System Analysis:** The first *lifecycle phase*, during which the system requirements are analyzed and allocated to *HWCIs* and *CSCIs*, and preliminary engineering requirements are defined for each CSCI.

**System Integration and Testing:** The *lifecycle phase* following *CSCI Testing and Integration*, during which the software and hardware are integrated and tested to see if the complete *System* performs according to system requirements.

**TLCSC:** Top-Level Computer Software Component. A 2167 subcomponent of a *CSCI*. TLCSCs are decomposed into *LLCSCs* and/or *Units*.

**Unit:** The smallest possible 2167 *software component* and a subcomponent of *CSC*. Units are not further decomposed. A Unit is coded and tested independently of other Units.