Rational Environment
Reference Manual

Reference Summary

Document Control Number: 8001A-21 (803-002302)
Rev. 2.0, September 1985
Rev. 3.0, November 1985
Rev. 4.0, July 1986
Rev. 5.0, July 1987(Delta)

This document subject to change without notice.

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

Rational
1501 Salado Drive
Mountain View, California 94043

How to Use This Book

The Reference Summary book of the *Rational Environment Reference Manual* is intended to be used as a quick reference to the resources provided by the Rational Environment™. The information is intended for experienced users of the Rational Environment. The *Rational Environment Basic Operations* and the *Rational Environment User's Guide* provide a better introduction and quick reference for new users.

Products other than the Rational Environment (for example, Rational Networking—TCP/IP or Rational Target Build Utility) are not documented in the Reference Summary. If you have purchased such products, however, the documentation for them typically contains quick-reference pages and tabs that are intended to be inserted into this Reference Summary.

The Reference Summary provides the following information, organized into tabbed sections:

- **World !**: A map of the library system of the Rational Environment. The map includes an indication of the manual/book in which each unit is documented.
- **World !Commands**: Specifications for the units that provide the interactive command interfaces to the facilities provided by the Environment. The specifications are organized alphabetically by simple name.
- **World !Io**: Specifications for the I/O packages defined in Chapter 14 of the *Reference Manual for the Ada® Programming Language*. It also contains the specifications for other I/O packages provided by the Environment. The specifications are organized alphabetically by simple name.
- **World !Lrm**: Specifications for the predefined units required by the *Reference Manual for the Ada Programming Language* that are provided by the Environment. The specifications are organized aphabetically by simple name.
- **World !Tools**: Specifications for the software components provided by the Environment and the programmatic interfaces to the resources provided by the Environment. The specifications are organized alphabetically by simple name.
- **Abbreviations**: Definitions of the predefined abbreviations provided for frequently used Environment commands and unit names.
- **Model Definitions**: Definitions of the predefined models that can be used to initialize the imports (links) of newly created worlds and subsystem views.

- **Project Tools**: An empty section intended as a repository for quick-reference information describing locally developed tools.
- **Release Information**: An empty section intended as a repository for release notes distributed by Rational.
- **Symbols and Switches**: Information on library-naming syntax and special symbols (including wildcards and substitution characters), search pattern characters, and session and library switches.
- **System Programming**: An empty section intended as a repository for specifications of units used locally for system programming that are not included in any of the above sections.

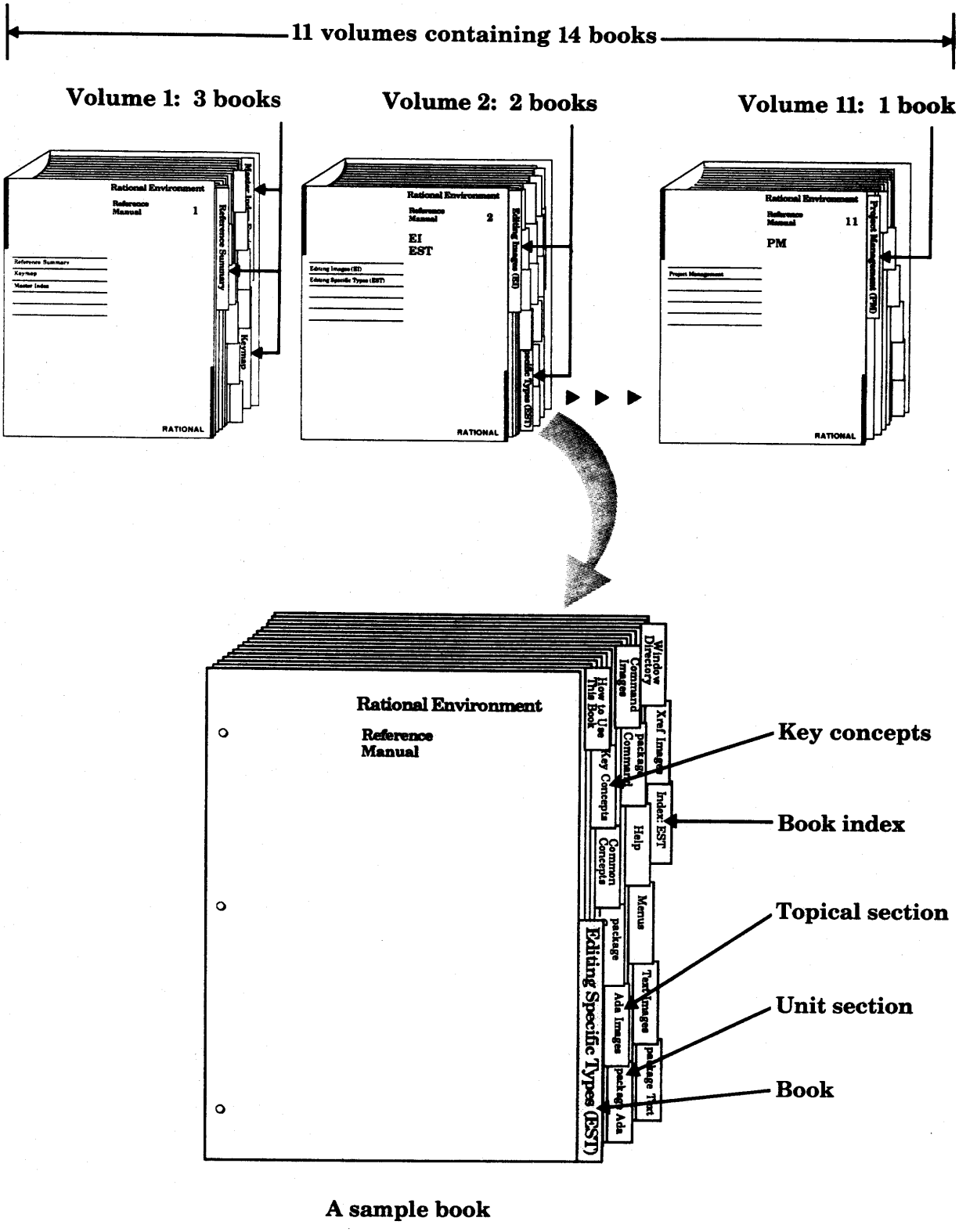
Organization of the Reference Manual

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

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

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

Organization of the *Rational Environment Reference Manual*



A sample book

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

Volume 1

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

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

Volumes 2–11

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

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

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

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

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

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

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

Suggestions for Finding Information

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

Learning about Environment Facilities

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

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

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

Finding Information on a Specific Item

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

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

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

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

Using the Index

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

Viewing Specifications On-Line

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

Using On-Line Help

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

Cross-Reference Conventions

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

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

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

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

Feedback to Rational: Reader's Comments Form

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

RATIONAL

MAP OF THE RATIONAL ENVIRONMENT LIBRARY SYSTEM

Name	Kind	Book Abbreviation For Documentation
Note: A blank book abbreviation indicates that documentation is not meaningful for the item. A n/a indicates that documentation is not currently available.		
! : Library (World);		
Commands	: Library (World);	
Compiler_Interface	: Library (World);	
Implementation	: Library (World);	
Io	: Library (World);	
Local	: Library (World);	
Lrm	: Library (World);	
Machine	: Library (World);	
Model	: Library (World);	
Software_Catalogs	: Library (World);	SW Library Catalog
Tools	: Library (World);	
Users	: Library (World);	
!Commands : Library (World);		
Abbreviations	: Library (World);	Reference Summary
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_Maintenance	: Ada (Pack_Spec);	PM
Command	: Ada (Pack_Spec);	EST
Common	: Ada (Pack_Spec);	EST
Compilation	: Ada (Pack_Spec);	LM
Daemon	: Ada (Pack_Spec);	SMU
Debug	: Ada (Pack_Spec);	DEB
Diana_Tree	: Ada (Pack_Spec);	n/a
Disk_Space	: Ada (Pack_Spec);	n/a
Editor	: Ada (Pack_Spec);	EI
File_Utilities	: Ada (Pack_Spec);	LM
Ftp	: Ada (Pack_Spec);	FTP
Job	: Ada (Pack_Spec);	SJM
Library	: Ada (Pack_Spec);	LM
Links	: Ada (Pack_Spec);	LM
Log	: Ada (Pack_Spec);	SJM
Message	: Ada (Pack_Spec);	SMU
Network	: Ada (Pack_Spec);	TRL
Operator	: Ada (Pack_Spec);	SMU
Program	: Ada (Pack_Spec);	SJM
Queue	: Ada (Pack_Spec);	SMU
Scheduler	: Ada (Pack_Spec);	SMU
Search_List	: Ada (Pack_Spec);	SJM
Sims	: Library (Subsystem);	System Manager's G.
Switches	: Ada (Pack_Spec);	LM
System_Backup	: Ada (Pack_Spec);	SMU

```
  System_Maintenance                  : Library (Subsystem);
    .Revn.Units.Check_Universe_Acls   : Ada (Proc_Spec);      n/a
    .Revn.Units.Find_Null_Acls        : Ada (Proc_Spec);      n/a
    .Revn.Units.Set_Universe_Acls     : Ada (Proc_Spec);      n/a
    .Revn.Units.Show_Groups           : Ada (Proc_Spec);      n/a
    .Revn.Units.Show_Identity         : Ada (Proc_Spec);      n/a
    .Revn.Units.Show_Jobs             : Ada (Proc_Spec);      n/a
    .Revn.Units.Show_Job_Names        : Ada (Proc_Spec);      n/a
    .Revn.Units.Show_Locks            : Ada (Proc_Spec);      n/a
    .Revn.Units.Show_Machine_Id       : Ada (Proc_Spec);      n/a
    .Revn.Units.Show_Memory_Hogs      : Ada (Proc_Spec);      n/a
    .Revn.Units.Show_Stats            : Ada (Proc_Spec);      n/a
    .Revn.Units.Show_Tasks            : Ada (Proc_Spec);      n/a
  Tape                                : Ada (Pack_Spec);      SMU
  Telnet                              : Ada (Pack_Spec);      TEL
  Terminal                            : Ada (Pack_Spec);      SMU
  Text                                : Ada (Pack_Spec);      EST
  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
--Standard                            : 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;
  Cg_Data                             : Library (World);
  Devices                             : Library (World);
  Editor_Data                         : Library (World);
    .Facit_Commands                   : Ada (Proc_Spec);
    .Facit_Commands                   : Ada (Proc_Body);      Facit Keymap
    .Facit_Keys                       : File;
    .Facit_User_Commands              : File;
    .Help_Data                        : Library (Directory);
    .Rational_Commands                : Ada (Proc_Spec);
    .Rational_Commands                : Ada (Proc_Body);      Rational Keymap
    .Rational_Keys                    : File;
    .Rational_User_Commands           : File;
    .Session_Switch_Help              : File;
```

```
    .Visible_Key_Names                : Ada (Pack_Spec);
    .Vt100_Commands                   : Ada (Proc_Spec);      Rational Keymap
    .Vt100_Commands                   : Ada (Proc_Body);
    .Vt100_Keys                       : File;
    .Vt100_User_Commands              : File;
  Error_Logs                          : Library (World);
  Groups                              : Library (World);
  Initialize                          : Ada (Proc_Spec);
  Initialize                          : Ada (Proc_Body);      System Manager's G.
  Initialize_Cross_Compilers          : Ada (Proc_Spec);
  Initialize_Cross_Compilers          : Ada (Proc_Body);
  Initialize_Daemons                  : Ada (Proc_Spec);
  Initialize_Daemons                  : Ada (Proc_Body);
  Initialize_Housekeeping             : Ada (Proc_Spec);
  Initialize_Housekeeping             : Ada (Proc_Body);
  Initialize_Network                  : Ada (Proc_Spec);
  Initialize_Network                  : Ada (Proc_Body);
  Initialize_Servers                  : Ada (Proc_Spec);
  Initialize_Servers                  : Ada (Proc_Body);
  Initialize_Site                     : Ada (Proc_Spec);
  Initialize_Site                     : Ada (Proc_Body);
  Initialize_Terminals                : Ada (Proc_Spec);
  Initialize_Terminals                : Ada (Proc_Body);
  Machine_Name                        : File;
  Operator_Capability                 : File (Text);
  Queues                              : Library (World);
  Release                             : Library (World);
    .Current.Activity                 : File (Activity);
    .Current.Commands                 : Library (World);
      .Login                          : Ada (Proc_Spec);
      .Login                          : Ada (Proc_Body);
  Search_Lists                        : Library (World);
    .Default                          : File;
  Shutdown_Help_File                  : File (Text);
  Tcp_Ip_Host_Id                      : File (Text);
  Temporary                           : Library (World);
  Transport_Name_Map                  : File (Text);
  Users                               : Library (World);
  User_Acl_Suffix                     : File (Text);
  User_Default_Acl_Suffix             : File (Text);

!Model : Library (World);
  R1000                               : Library (World);      Reference Summary
  R1000_Implementation                : Library (World);      Reference Summary
  R1000_Portable                      : Library (World);      Reference Summary

!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                                  : Library (Subsystem);
    .Revn.Units.Ci                    : Ada (Pack_Spec);      System Manager's G.
    .Revn.Units.Commands              : Library (Directory);
      .Run                            : Ada (Pack_Spec);      System Manager's G.
      .Show                           : Ada (Pack_Spec);      System Manager's G.
      .Typ                            : Ada (Proc_Spec);      System Manager's G.
```

```
Compatibility                    : Library (Subsystem);
   .Revn.Units.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);        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
Map_Generic                      : Ada (Gen_Pack);         PT
Networking                       : Library (Directory);
   .Byte_Defs                    : Ada (Pack_Spec);        TRL
   .Byte_String_Io               : Ada (Pack_Spec);        n/a
   .Exos_8010_3_Sd               : File;
   .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
   .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
   .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
Queue_Generic                    : Ada (Gen_Pack);         PT
Random                           : Ada (Pack_Spec);        n/a
Rpc_Servers                      : Library (Subsystem);
   .Revn.Units.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                 : Library (Subsystem);  System Manager's G.
  .Revn.Units.Log_Reader            : Ada (Pack_Spec);
  .Revn.Units.Outage_Information    : Ada (Pack_Spec);
  .Revn.Units.Sample_Logs           : Library (Directory);
  .Revn.Units.Show_Error_Log        : Ada (Proc_Spec);
  .Revn.Units.System_Information    : Ada (Pack_Spec);
  .Revn.Units.System_Report         : Ada (Pack_Spec);
System_Utilities                    : Ada (Pack_Spec);      SMU
Table_Formatter                     : Ada (Gen_Pack);       ST
Table_Sort_Generic                  : Ada (Gen_Proc);       PT
Tape_Tools                          : Ada (Pack_Spec);      n/a
Target_Build_Utility                : Library (Subsystem);
  .Revn.Units.Target_Builder        : Ada (Gen_Pack);       TBU
  .Revn.Units.Vax_Builder           : File (Text);          TBU
Time_Utilities                      : Ada (Pack_Spec);      PT
Unbounded_String                    : Ada (Gen_Pack);       ST
Unchecked_Conversions               : Ada (Pack_Spec);      PT
Xref_Utility                        : Library (Subsystem);
  .Revn.Units.Commands.Xref         : Ada (Pack_Spec);      LM
```

RATIONAL

```
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.

    pragma Subsystem (Os_Commands);
    pragma Module_Name (4, 3507);

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);

    pragma Subsystem (Os_Commands);
    pragma Module_Name (4, 3933);

end Action_Utilities;
```

Action_Utilities, !Commands

```
package Activity is

    pragma Subsystem (Commands);
    pragma Module_Name (4, 3940);

    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;
```

Activity, !Commands

```
-- 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;
    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

    pragma Subsystem (Command);
    pragma Module_Name (4, 2209);
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 => "[!HJL?,~!HJL.ABC?,~!HJL.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.
    --
    --   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.
    --      The NONRECURSIVE option does not affect the interpretation of the
    --      CDB specificaton even when it is obtained from the Objects
    --      parameter.
    --
    --      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.
    --
    --   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 and CDB
    --      option (if present) by expanding context-sensitive characters
    --      (such as ^ and @), expanding indirect file references, and
    --      removing all attributes.
    --
    -- For downward compatibility the following options are provided.
    --
    --   GAMMA0
    --      write a tape which can be read on a Gamma0 system.
    --
    --   GAMMA1
    --      write a tape which can be read on a Gamma1 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=210.


    -- 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.HJL.CLI.TEST
--     [!USERS.HJL.TESTS.@, !USERS.HJL.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.

-- If the name of the archived object does not have the For_Prefix as a
-- prefix, it is restored under its original name.

-- The For_Prefix may contain wildcard characters (#, @, ?) and the
-- Use_Prefix parameter may contain substitution characters (@ or #
-- only). (Not implemented in DO)

-- 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 worlds. 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 as worlds.
--
-- The following options are allowed in the Options parameter:
--
--   FORMAT and LABEL: options as in the save option.

--   COMPATIBILITY_DATABASE, (CDB) [=<Subsystems>]
--      Specifies that the Compatibility Databases for just the named
--      subsystems are to be restored.

--   NONRECURSIVE
--      prevents subcomponents of libraries and Ada units from being
--      implicitly restored. for example:

--       Restore
--          (Objects => "[!USERS.HJL, !USERS.HJL.CLI, !USERS.HJL.CLI.@]",
--           Options => "R1000 NONRECURSIVE");

--       will restore only the named objects and not their substructure.

--   OVERWRITE = ALL_OBJECTS | NEW_OBJECTS | UPDATED_OBJECTS | CHANGED_OB
--       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.

--   PROMOTE
--      After they are restored, any Ada units will be promoted to the
--      state they were in when they were archived.

--   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.
```

```
--        (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.
--
--    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 the special values INHERIT or ARCHIVED.
--       ARCHIVED means to use the ACL archived with the object and is
--       the default for all three ACL options.  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.

-------------------------------------------------------------------------------

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.

-------------------------------------------------------------------------------

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 Use_Prefix/For_Prefix parameters
-- specify where the objects are to go as in Archive.Restore.

-- Each name consists of an (optional) machine name followed directly
-- by a Objects parameter. A machine name has the form !!name.
-- the Objects part of the source name is like that given to the save
-- operation.
--
-- The Use_Prefix and the For_Prefix function as in the Restore command.

-- 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.
--
-- The Options parameter has the following options.
--
--   AFTER=<time_expression>
--      as in the save operation.
--
--   COMPATIBILITY_DATABASE, CDB
--   NONRECURSIVE
--      as in the save operation.
--
--   PROMOTE, REPLACE,
--   BECOME_OWNER,
--   OBJECT_ACL, WORLD_ACL, DEFAULT_ACL
--      as in the restore operation.
--

-- Examples of calls:
--
--  Copy (Objects    => "!USERS.HJL.CLI",
--        Use_Prefix => "!!M1");
--
--    will copy the CLI directory in !USERS.HJL on the
--    current machine to machine M1 !USERS.HJL.CLI.
--
--  Copy (Objects => "!!M2!USERS.JMK.CLI");
--
--    will copy !USERS.JMK.CLI on M2 to !USERS.JMK.CLI on the
--    current machine.
--
--  Copy (Objects    => "!!M3!USERS.HJL.CLI.CMD",
--        Use_Prefix => "!USERS.JMK",
--        For_Prefix => "!USERS.HJL.CLI");
--
--    will copy the file !USERS.HJL.CLI.CMD on M3 to
--    !USERS.JMK.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.HJL.ILFORD",
--        Use_Prefix => "!!M2!AGFA",
--        For_Prefix => "!USERS.HJL");
--
--    will copy !USERS.HJL.ILFORD from machine M1 to
--    machine M2 !AGFA!ILFORD
--
--  Copy (Objects    => "!USERS.HJL.CLI",
--        Use_Prefix => "!!M1",
--        Options    => "REPLACE AFTER=12/25/85");
--
--    will copy those files which have changed since 12/25/85 in
--    !USERS.HJL.CLI on the current machine to machine M1 !USERS.HJL.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.

    pragma Subsystem (Archive);
    pragma Module_Name (4, 3546);
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_Demotion : Boolean := False;
                         Allow_Implicit_Accept_Changes : Boolean := True;
                         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.

    -- 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;
```

```
                        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.

-- 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 ingored 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>";
                               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.

procedure Revert (What_Object : String := "<SELECTION>";
                  To_Generation : Integer := -1;
                  Make_Latest_Generation : Boolean := False;
                  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.

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 <IMAGE_TEXT>,
-- 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>";
              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 as the reservation token name.

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_Objects : String := "<SELECTION>";
                 New_Reservation_Token_Name : String := "<AUTO_GENERATE>";
                 Comments : String := "";
                 Work_Order : String := "<DEFAULT>";
                 Response : String := "<PROFILE>");
```

```
-- Make the object(s) in the given working view(s) have a separate
-- reservation.  This command severs the relationship between views
-- for objects.  When done, the views specified in this command will
-- have their own reservation to share.  All other views (not
-- specified) will share a different reservation.
-- A specific reservation token name can be provided, if desired.

procedure Join (What_Object : String := "<SELECTION>";
                To_Which_View : String := ">>VIEW NAME<<";
                Comments : String := "";
                Work_Order : String := "<DEFAULT>";
                Response : String := "<PROFILE>");

-- Make object in two or more working views share a reservation.  The
-- objects in the views must be identical (textually)  and controlled
-- for this command to succeed.

------------------------------------------------------------------------


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.  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
-- SUBSYSTEM.state.configurations.release_name.  It also creates an
-- import description file in the same place, named release_name &
-- "_imports".  This import description file lists the configuration
-- objects for all views that are imported.  It is maintained by
-- all commands that modify or adjust the imports.  These two objects
-- are used to reconstruct views from configurations.

-- 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 := "";
                Construct_Subpath_Name : Boolean := False;
                Create_Spec_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.  Units are not complied
-- to a state higher that they were in the source.

-- 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"'

-- 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;
                     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 := 0;
              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.

-- The import description file mentioned in the release command is
-- brought up to date by this command.

-- 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.

--------------------------------------------------------------------------
```

```
type Subsystem_Type_Enum is (Spec_Load, Combined, System);

procedure Initial (Subsystem : String := ">>SUBSYSTEM NAME<<";
                   Working_View_Base_Name : String := "Rev1";
                   Subsystem_Type : Subsystem_Type_Enum := Cmvc.Spec_Load;
                   View_To_Import : String := "";
                   Model : String := "R1000";
                   Comments : String := "";
                   Work_Order : String := "<DEFAULT>";
                   Volume : Natural := 0;
                   Response : String := "<PROFILE>");

-- Build a new subsystem 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 subsystem.

------------------------------------------------------------------------

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 Build (Configuration : String := ">>CONFIGURATION NAME<<";
                 View_To_Import : String := "";
                 Model : String := "R1000";
                 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 = "", and if a text file exists with the name "same
-- as configuration_object" & "_imports", that text file is opened
-- after the views are built and imports are constructed from the views
-- or configuration objects named in that file.  Please note that copy,
-- initial, import, and remove_import will create and maintain such a
-- text file, so it is probably there.

------------------------------------------------------------------------
--                        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>");

-- In this case, All_Units means all of the units in the current
-- view.  Naming a view means all units in that view.

procedure Show_All_Uncontrolled (Object_Or_View : String := "<CURSOR>";
                                 Response : String := "<PROFILE>");

-- List objects that are not controlled.  Produces output only if an
-- object listed (or one in the units directory if a view is supplied)
-- is not under CMVC control

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.

-- Object Name  Generation  Where  Chkd Out  By Who  Expected Check In
-- ===========  ==========  =====  ========  ======  ==================
-- UNITS.FOO      5 of 8    VIEW     Yes      MTD    Apr 7, 1987

-- 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.

-------------------------------------------------------------------------

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_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 and/or combined views.  If a load view is provided, no
-- specs are copied; all specs are copied for combined views.
-- This operation fails if any unit isn't coded, or any spec exists
-- for which a body is required and one doesn't exist.


pragma Subsystem (Cmvc);
pragma Module_Name (4, 3704);
end Cmvc;
```

```
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>");

    pragma Subsystem (Cmvc);
    pragma Module_Name (4, 3707);
end Cmvc_Maintenance;
```

```
package Command is

    procedure Diana_Edit (Name : String := "<IMAGE>");
    procedure Spawn;
    procedure Debug;

    pragma Subsystem (Command);
    pragma Module_Name (4, 2212);

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;
    pragma Subsystem (Object_Editor);
    pragma Module_Name (4, 2215);
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.

-- Specifying Scope => Subunits_Too will cause subunits of the
-- designated units to be promoted.  Specifying Scope => All_Parts is
-- equivalent to the Make procedure described below.

-- 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; To : String; 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.


    pragma Subsystem (Commands);
    pragma Module_Name (4, 3936);

end Compilation;
```

```
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;

   pragma Subsystem (Os_Commands);
   pragma Module_Name (4, 3932);

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 := "");

   pragma Subsystem (Native_Debugger);
   pragma Module_Name (4, 3801);

end Debug;
```

```
package Diana_Tree is

    procedure Ada_Edit (Name : String := "<IMAGE>");

    pragma Subsystem (Command);
    pragma Module_Name (4, 2211);

end Diana_Tree;
```

Diana_Tree, !Commands

```
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, Edb_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;

   pragma Subsystem (Commands);
   pragma Module_Name (4, 3935);
end Disk_Space;
```

```
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;
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;

    pragma Subsystem (Command);
    pragma Module_Name (4, 2205);

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.


    pragma Subsystem (Os_Commands);
    pragma Module_Name (4, 3929);

end File_Utilities;
```

```
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 := "");

    pragma Subsystem (Command);
    pragma Module_Name (4, 2206);
end Job;
```

Job, !Commands

```
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 := "@";
```

Library, !Commands

```
                    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.  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.

   pragma Subsystem (Commands);
   pragma Module_Name (4, 3921);
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>");
    pragma Subsystem (Commands);
    pragma Module_Name (4, 3938);

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   XXX
--       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.

    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3218);
end Log;
```

```
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);

    pragma Subsystem (Command);
    pragma Module_Name (4, 2208);

end Message;
```

```
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 priv 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 capabilites 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

    pragma Subsystem (Os_Commands);
    pragma Module_Name (4, 3926);

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", "!New_Tool") returns:
   --
   -- "!Subsystem.Rev8_4_0.Units.Commands".View.Initial ("!New_Tool");

   pragma Subsystem (Commands);
   pragma Module_Name (4, 3930);

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).

   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.
--
------------------------------------------------------------------------------

   pragma Subsystem (Os_Commands);
   pragma Module_Name (4, 3922);

end Queue;
```

```
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.


pragma Subsystem (Os_Commands);
pragma Module_Name (4, 3923);

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

    pragma Subsystem (Commands);
    pragma Module_Name (4, 3939);

end Search_List;
```

```
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.

    pragma Subsystem (Commands);
    pragma Module_Name (4, 3934);
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.

    pragma Subsystem (Os_Commands);
    pragma Module_Name (4, 3924);

end System_Backup;
```

```
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);


    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3927);

end Tape;
```

```
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);

    pragma Subsystem (Os_Commands);
    pragma Module_Name (4, 3925);
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 be redirected to the named file.
    pragma Subsystem (Command);
    pragma Module_Name (4, 2210);
end Text;
```



```
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>");

    pragma Subsystem (Command);
    pragma Module_Name (4, 2217);

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>") 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_Comments_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>") 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 := 1;
                        On_Venture : String := "<VENTURE>";
                        Propagate : 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.
-- 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 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>") 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;

    pragma Subsystem (Cmvc);
    pragma Module_Name (4, 3781);

end Work_Order;
```

```
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.

    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3208);

    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

    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3203);

    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

    pragma Subsystem (Input_Output, Private_Part => Closed);
    pragma Module_Name (4, 3506);

    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_Independent_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

    pragma Subsystem (Miscellaneous);
    pragma Module_Name (4, 804);

end Io_Exceptions;
```

```
with Action;
with Directory;
with Polymorphic_Io;

package Object_Set is

    pragma Subsystem (Directory, Closed);
    pragma Module_Name (4, 1721);

    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 hov 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.

   pragma Subsystem (Input_Output, Private_Part => Closed);
   pragma Module_Name (4, 3223);
end Pipe;
```

```
with Action;
with Default;
with Directory;
with Io_Exceptions;
with System;

package Polymorphic_Io is

    pragma Subsystem (Directory);
    pragma Module_Name (4, 1706);

    -- 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

    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3210);

    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

    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3204);

    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);

    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3214);

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);

    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3215);
end Terminal_Specific;
```

```
with Io_Exceptions;

package Text_Io is

    pragma Subsystem (Input_Output, Private_Part => Closed);
    pragma Module_Name (4, 3201);

    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 := Out_File;
                    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

    pragma Subsystem (Object_Editor, Closed);
    pragma Module_Name (4, 2219);

    -- 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;
```

```
package Calendar is

    pragma Subsystem (Kernel, Private_Part => Closed);
    pragma Module_Name (4, 406);

    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 Standard is
    type Boolean is (False, True);
    for Boolean'Size use 1;

    type Integer       is range -2**31-1 .. 2**31-1;

    type Long_Integer  is range (-2**62 - 2**62) .. (2**62 - 1 + 2**62);
                            --       -2**63 .. 2**63-1

    type Float is digits 15 range (2.0**1023) - (2.0**97) + (2.0**1023)..
                                  - ((2.0**1023) - (2.0**97) + (2.0**1023));
                                   --  -1.7977E308 .. 1.7977E308;

    type Character is (Nul, ..., Del);
    for Character use (0, ..., 127);
    for Character'Size use 8;

    package Ascii is ... end Ascii;

    subtype Natural  is Integer range 0 .. Integer'Last;
    subtype Positive is Integer range 1 .. Integer'Last;

    type String is array (Positive range <>) of Character;

    type Duration is delta 2.0**(-15)
                      -- -3.051757812500E-05
                    range -(2.0**32) .. (2.0**32) - (2.0**(-15));
                      -- -4.294967296000E+09 .. 4.294967296000E+09

    Constraint_Error : exception;
    Numeric_Error    : exception;
    Program_Error    : exception;
    Storage_Error    : exception;
    Tasking_Error    : exception;

end Standard;
```

```
package System is

    pragma Read_Only;
    pragma Open_Private_Part;
    pragma Subsystem (Ada_Base);
    pragma Module_Name (4, 66);

    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;

pragma Subsystem (Miscellaneous);
pragma Module_Name (4, 824);
```

```
generic
    type Object is limited private;
    type Name is access Object;
procedure Unchecked_Deallocation (X : in out Name);

pragma Subsystem (Miscellaneous);
pragma Module_Name (4, 825);
```

```
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.


    pragma Subsystem (Os_Commands);
    pragma Module_Name (4, 3508);

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.

    pragma Subsystem (Ada_Os);
    pragma Module_Name (4, 2200);
end Ada_Object_Editor;
```

```
with Diana;
with Directory;
with Directory_Tools;

package Ada_Text is
    pragma Subsystem (Tools);
    pragma Module_Name (4, 3548);

    -- 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;
pragma Subsystem (Miscellaneous);
pragma Module_Name (4, 827);
```

```
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;

    --/R1000 pragma Module_Name (4, 828);
    --/R1000 pragma Subsystem (Miscellaneous);

end Bit_Operations;
```



```
package Bounded_String is

    pragma Subsystem (Tools);
    pragma Module_Name (4, 3976);

    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;
```

```
with Profile;
package Check is

    -- Check runtime compatibility on the R1000 of individual units,
    -- lists of units, views, directories, or activities.

    type Status is (Identical, Upward_Compatible, Incompatible, Error);

    subtype Unit_Name is String;
    subtype Activity_Name is String;

    subtype Response_Profile is Profile.Response_Profile;

    Current_Activity : constant Activity_Name := "<ACTIVITY>";

--    Check whether the exported functionality of Name1 is
--    implemented by the corresponding implementation.

--    If Name2 is specified, check Name1 against Name2; otherwise, check
--    Name1 against its correspondant in the provided activity.
--    Wildcards specified in Name1 cause checking of against Name2 of each
--    object specified by Name1.

--    Name1 may be fully qualified (starts with ! or some other character
--    recognized by the environment) or may be relative to the current context
--    or to Name2.

--    Activity => "" or "<ACTIVITY>" (default) means to use the current activ
--    otherwise, Activity specifies the name of an activity file.

--    If Closure is true, also compatibility check all units used by the
--    requested objects.

--    Examples:

--     Check_Units (Name1 => "Check", Closure => True);
--     Determine whether Check (and everything required for it to run) is
--     compatible with the implementation specified in the current activity

--     Check_Units (Name1 => "!Tools.Compatibility.Rev9_2_Spec",
--                  Name2 => "!Tools.Compatibility.Rev9_1_Working");
--     Since both names are specified, the activity is unused.  If Closure
--     were specified, the activity would be used for determining the
--     compatibility of units in the closure.  Specifying a view name is
--     equivalent to specifiying "view_name?'Spec"; it means "check
--     all ADA specs in the view".

--     Check_Units (Name1 => "!Subsys.Rev9_1_spec.Pack1",
--                  Name2 => "!Subsys.Rev9_1_Working")
--     Specifying names of a unit and a view means "check the unit in the
--     spec view against the corresponding unit in the load view".

--     Check_Units (Name1 => "Pack1",
--                  Name2 => "!Subsys.Rev9_1_Working");
--     Assuming that we are not in a context where "Pack1" is recognized:
--     The name "Pack1" is resolved in the context specified by Name2, resulting
--     in the object "!Subsys.Rev9_1_Working.Pack1" which is a unit in a load
--     view.  Name2 is also used to specify the object to check Name1 against.
--     However, instead of checking the load view unit "Pack1" against itself
```

```
--      compatibility checker substitutes the unit "Pack1" in the spec view,
--      which it check against the unit "Pack1" in the load view.  The activity
--      is used to find the spec view corresponding to the load view.

--     Check_Units (Name1 => "!Machine.Release.Current.Activity");
--      Since Name1 specifies an activity, check the consistency of the
--      activity's view pairs.  In this case, Closure, Name2, and Activity
--      are meaningless.

--     Check_Units (Name1 => "<ACTIVITY>");
--      Check the consistency of the current default activity's view pairs.

--     Check_Units (Name1 => "");
--      The standard resolution of "" is the enclosing library, so this
--      checks @'Spec against the current activity. (Specifying a directory
--      is equivalent to specifying all ADA specs in the directory).

    procedure Check_Units (Name1 : Unit_Name := ">>Name<<";
                           Name2 : Unit_Name := "";
                           Activity : Activity_Name := "<ACTIVITY>";
                           Closure : Boolean := False;
                           Response : Response_Profile := Profile.Get);

    function Check_Units
                (Name1 : Unit_Name := ">>Name<<";
                 Name2 : Unit_Name := "";
                 Activity : Activity_Name := "<ACTIVITY>";
                 Closure : Boolean := False;
                 Response : Response_Profile := Profile.Get) return Status;

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

    pragma Subsystem (Tools);
    pragma Module_Name (4, 3984);

    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 Cocurrent_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.

    pragma Subsystem (Native_Debugger);
    pragma Module_Name (4, 3802);

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;

    pragma Subsystem (Object_Editor);
    pragma Module_Name (4, 2202);

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
-- ation 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 a
-- 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, additi
-- 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 fai

    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

    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 ret
-- 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.Co
        --                               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;

    pragma Subsystem (Cmvc, Closed);
    pragma Module_Name (4, 3525);
end Directory_Tools;
```

```
with Calendar;
with Machine;

package Disk_Daemon is
    pragma Subsystem (Disk_Cleaner);
    pragma Module_Name (4, 3401);

    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;

    pragma Subsystem (Abstract_Types);
    pragma Module_Name (4, 731);

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_Redrav is true, the image will be redravn even if it
    -- already exists.

    procedure Display (Library : Directory.Object;
                       Selection : Directory.Object := Directory.Nil;
                       Force_Redrav : 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<<");

    pragma Subsystem (Object_Editor);
    pragma Module_Name (4, 2221);

end Library_Object_Editor;
```

```
with Diana;
with Directory;
with Links_Implementation;
with Io_Exceptions;

package Link_Tools is

    pragma Subsystem (Tools, Private_Part => Closed);
    pragma Module_Name (4, 3975);

    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

    pragma Subsystem (Abstract_Types, Private_Part => Open);
    pragma Module_Name (4, 710);

    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;
```

```
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

    pragma Subsystem (Tools);
    pragma Module_Name (4, 3977);

    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 Default;
with Directory;
with Machine;

package Object_Editor is

    pragma Subsystem (Object_Editor, Closed);
    pragma Module_Name (4, 2223);

    -- 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


    pragma Subsystem (Directory);
    pragma Module_Name (4, 3528);

    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

    pragma Subsystem (Directory);
    pragma Module_Name (4, 3219);

    -- A collection of job-related utilities for control of log generation,
    -- Activity files, and error reactions.  These facilities are used by
    -- Commands.Compilation, .Directory, and .Tape, Source_Archive, 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, XXX              "
-- 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.
--
-- <DEFAULT>                    System default, ignores user profile.
--                              PERSEVERE, :::, ~???, ---..###, Width=>77
--                              Prefix=>YR_MN_DY, HR_MN_SC SYMBOLS, USE_OUTPUT
--
-- <ERRORS>                     +++, ***..XXX, <PROFILE>
--
-- <IGNORE>                     ~::::..###. <PROFILE>
--
-- <NIL>                        Empty, ignores user profile.
--                              Quit, ~::::..###, Prefix=>, Width=>77, USE_OU
--
-- <PROFILE>                    Values set by job/session profile.
--
-- <QUIET>                      Equivalent of <NIL>
--
-- <PROGRESS>                   +++, +++, ***..XXX, <PROFILE>
--
-- <RAISE_EXCEPTION>            PROPAGATE, <PROFILE>
--
-- <SESSION_PROFILE>            Use session profile rather than job.
--
-- <WARN>                       +++..XXX, <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;
```

```
generic
    type Element is private;

    pragma Must_Be_Constrained (Yes => Element);

package Queue_Generic is

    pragma Subsystem (Abstract_Types, Private_Part => Open);
    pragma Module_Name (4, 712);

    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.

    pragma Subsystem (Tools);
    pragma Module_Name (4, 3563);
end Random;
```

Random, !Tools

```
package Script is

    procedure Pretty_Print (Script_File : String; Command_File : String);

    pragma Subsystem (Command);
    pragma Module_Name (4, 2026);
end Script;
```

Script, !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 Set_Generic is

    pragma Subsystem (Tools);
    pragma Module_Name (4, 3978);

    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.


    pragma Subsystem (Miscellaneous);
    pragma Module_Name (4, 810);

end Simple_Status;
```

```
generic
    type Element is private;

    pragma Must_Be_Constrained (Yes => Element);
package Stack_Generic is

    pragma Subsystem (Abstract_Types, Private_Part => Open);
    pragma Module_Name (4, 714);

    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;
```

Stack_Generic, !Tools

```
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

    pragma Subsystem (Tools);
    pragma Module_Name (4, 3980);

    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 --
```

String_Map_Generic, !Tools

```
    ------------------------------------------------------

    -- := 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

    pragma Subsystem (Tools, Private_Part => Closed);
    pragma Module_Name (4, 3981);

    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;

    pragma Subsystem (Tools);
    pragma Module_Name (4, 303);

end String_Utilities;
```

```
with Directory;
with Machine;
with Calendar;
with System;

package System_Utilities is

    pragma Subsystem (Tools, Closed);
    pragma Module_Name (4, 3973);

    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/IOC
   --      1 : SYS
   --      2 : SEQ
   --      3 : VAL
   --      4 : TYP
   --      5 : FIU
   --      100 : MEM0
   --      101 : MEM1
   --      102 : MEM2
   --      103 : MEM3
   --    etc.
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);

    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3217);
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);
pragma Subsystem (Interface);
pragma Module_Name (4, 3979);with Device_Independent_Io;
```

```
package Tape_Tools is

    pragma Subsystem (Input_Output);
    pragma Module_Name (4, 3974);

    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;

    pragma Subsystem (Tools);
    pragma Module_Name (4, 3972);

end Time_Utilities;
```

```
generic
    Default_Maximum_Length : Natural := 20;
package Unbounded_String is

    pragma Subsystem (Tools, Private_Part => Open);
    pragma Module_Name (4, 3983);

    ------------------------------------------------------------------------
    --  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, ma
        -- calls to this convert faster than to an equivalent Unchecked_Conve
        -- 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.


    pragma Subsystem (Miscellaneous);
    pragma Module_Name (4, 3543);
end Unchecked_Conversions;
```

Unchecked_Conversions, !Tools

```
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;
```

Xref, !Tools

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
COMP	=>	!COMMANDS.COMPILATION
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 main 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 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 System_Backup;
procedure Do_Backup (Variety : System_Backup.Kind := System_Backup.Full;
                     Starting_At : String := "");
-- System_Backup.Backup; preforms various other housekeeping functions


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 (Name : String := "<IMAGE>";
                 Options : String := "<DEFAULT>";
                 Banner : String := "<DEFAULT>";
                 Header : String := "<DEFAULT>";
                 Footer : String := "<DEFAULT>");
-- Queue.Print


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 Send (Who : String := ">>USER or all<<";
                Contents : String := "");
-- Message.Send


procedure Sledit (Session : String := ""; User : String := "");
-- Search_List.Show_List (allows editing)
```

```
procedure Ssedit (Switch_File : String := "<SESSION>");
-- Switches.Edit; by default edits session switches


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
```

!Model.R1000 Links

KIND	LINK	=>	SOURCE
EXT:	ACCESS_LIST	=>	!COMMANDS.ACCESS_LIST
EXT:	ACCESS_LIST_TOOLS	=>	!TOOLS.ACCESS_LIST_TOOLS
EXT:	ACTION_UTILITIES	=>	!COMMANDS.ACTION_UTILITIES
EXT:	ACTIVITY	=>	!COMMANDS.ACTIVITY
EXT:	ADA	=>	!COMMANDS.ADA
EXT:	ADA_OBJECT_EDITOR	=>	!TOOLS.ADA_OBJECT_EDITOR
EXT:	ADA_TEXT	=>	!TOOLS.ADA_TEXT
EXT:	ALLOWS_DEALLOCATION	=>	!TOOLS.ALLOWS_DEALLOCATION
EXT:	ARCHIVE	=>	!COMMANDS.ARCHIVE
EXT:	BOUNDED_STRING	=>	!TOOLS.BOUNDED_STRING
EXT:	BYTE_STRING_IO	=>	!TOOLS.NETWORKING.BYTE_STRING_IO
EXT:	CALENDAR	=>	!LRM.CALENDAR
EXT:	CMVC	=>	!COMMANDS.CMVC
EXT:	CMVC_MAINTENANCE	=>	!COMMANDS.CMVC_MAINTENANCE
EXT:	COMMAND	=>	!COMMANDS.COMMAND
EXT:	COMMON	=>	!COMMANDS.COMMON
EXT:	COMPILATION	=>	!COMMANDS.COMPILATION
EXT:	CONCURRENT_MAP_GENERIC	=>	!TOOLS.CONCURRENT_MAP_GENERIC
EXT:	DAEMON	=>	!COMMANDS.DAEMON
EXT:	DEBUG	=>	!COMMANDS.DEBUG
EXT:	DEBUG_TOOLS	=>	!TOOLS.DEBUG_TOOLS
EXT:	DEVICE_INDEPENDENT_IO	=>	!IO.DEVICE_INDEPENDENT_IO
EXT:	DIANA_OBJECT_EDITOR	=>	!TOOLS.DIANA_OBJECT_EDITOR
EXT:	DIANA_TREE	=>	!COMMANDS.DIANA_TREE
EXT:	DIRECTORY_TOOLS	=>	!TOOLS.DIRECTORY_TOOLS
EXT:	DIRECT_IO	=>	!IO.DIRECT_IO
EXT:	DISK_DAEMON	=>	!TOOLS.DISK_DAEMON
EXT:	DISK_SPACE	=>	!COMMANDS.DISK_SPACE
EXT:	EDITOR	=>	!COMMANDS.EDITOR
EXT:	FILE_TRANSFER	=>	!TOOLS.NETWORKING.FILE_TRANSFER
EXT:	FILE_UTILITIES	=>	!COMMANDS.FILE_UTILITIES
EXT:	FTP	=>	!COMMANDS.FTP
EXT:	FTP_DEFS	=>	!TOOLS.NETWORKING.FTP_DEFS
EXT:	FTP_NAME_MAP	=>	!TOOLS.NETWORKING.FTP_NAME_MAP
EXT:	FTP_PROFILE	=>	!TOOLS.NETWORKING.FTP_PROFILE
EXT:	HASH	=>	!TOOLS.HASH
EXT:	HOST_ID_IO	=>	!TOOLS.NETWORKING.HOST_ID_IO
EXT:	INCREMENTAL	=>	!COMMANDS.INCREMENTAL
EXT:	INTERCHANGE	=>	!TOOLS.NETWORKING.INTERCHANGE
EXT:	INTERCHANGE_DEFS	=>	!TOOLS.NETWORKING.INTERCHANGE_DEFS
EXT:	IO	=>	!IO.IO
EXT:	IO_EXCEPTIONS	=>	!IO.IO_EXCEPTIONS
EXT:	JOB	=>	!COMMANDS.JOB
EXT:	LIBRARY	=>	!COMMANDS.LIBRARY
EXT:	LIBRARY_OBJECT_EDITOR	=>	!TOOLS.LIBRARY_OBJECT_EDITOR
EXT:	LINKS	=>	!COMMANDS.LINKS
EXT:	LINK_TOOLS	=>	!TOOLS.LINK_TOOLS
EXT:	LIST_GENERIC	=>	!TOOLS.LIST_GENERIC
EXT:	LOG	=>	!COMMANDS.LOG
EXT:	MACHINE_CODE	=>	!LRM.MACHINE_CODE
EXT:	MAP_GENERIC	=>	!TOOLS.MAP_GENERIC
EXT:	MESSAGE	=>	!COMMANDS.MESSAGE
EXT:	OBJECT_EDITOR	=>	!TOOLS.OBJECT_EDITOR
EXT:	OBJECT_SET	=>	!IO.OBJECT_SET
EXT:	OPERATOR	=>	!COMMANDS.OPERATOR
EXT:	PARAMETER_PARSER	=>	!TOOLS.PARAMETER_PARSER
EXT:	PIPE	=>	!IO.PIPE
EXT:	POLYMORPHIC_IO	=>	!IO.POLYMORPHIC_IO
EXT:	POLYMORPHIC_SEQUENTIAL_IO	=>	!IO.POLYMORPHIC_SEQUENTIAL_IO
EXT:	PROFILE	=>	!TOOLS.PROFILE
EXT:	PROGRAM	=>	!COMMANDS.PROGRAM
EXT:	QUEUE	=>	!COMMANDS.QUEUE
EXT:	QUEUE_GENERIC	=>	!TOOLS.QUEUE_GENERIC
EXT:	RANDOM	=>	!TOOLS.RANDOM
EXT:	RPC	=>	!TOOLS.NETWORKING.RPC
EXT:	RPC_CLIENT	=>	!TOOLS.NETWORKING.RPC_CLIENT
EXT:	RPC_SERVER	=>	!TOOLS.NETWORKING.RPC_SERVER
EXT:	SCHEDULER	=>	!COMMANDS.SCHEDULER
EXT:	SCRIPT	=>	!TOOLS.SCRIPT
EXT:	SEARCH_LIST	=>	!COMMANDS.SEARCH_LIST
EXT:	SEQUENTIAL_IO	=>	!IO.SEQUENTIAL_IO
EXT:	SET_GENERIC	=>	!TOOLS.SET_GENERIC
EXT:	SIMPLE_STATUS	=>	!TOOLS.SIMPLE_STATUS
EXT:	STACK_GENERIC	=>	!TOOLS.STACK_GENERIC
EXT:	STRING_MAP_GENERIC	=>	!TOOLS.STRING_MAP_GENERIC
EXT:	STRING_TABLE	=>	!TOOLS.STRING_TABLE
EXT:	STRING_UTILITIES	=>	!TOOLS.STRING_UTILITIES
EXT:	SWITCHES	=>	!COMMANDS.SWITCHES
EXT:	SYSTEM	=>	!LRM.SYSTEM
EXT:	SYSTEM_BACKUP	=>	!COMMANDS.SYSTEM_BACKUP
EXT:	SYSTEM_UTILITIES	=>	!TOOLS.SYSTEM_UTILITIES
EXT:	TABLE_FORMATTER	=>	!TOOLS.TABLE_FORMATTER
EXT:	TABLE_SORT_GENERIC	=>	!TOOLS.TABLE_SORT_GENERIC
EXT:	TAPE	=>	!COMMANDS.TAPE
EXT:	TAPE_SPECIFIC	=>	!IO.TAPE_SPECIFIC
EXT:	TAPE_TOOLS	=>	!TOOLS.TAPE_TOOLS
EXT:	TCP_IP_BOOT	=>	!TOOLS.NETWORKING.TCP_IP_BOOT
EXT:	TCP_IP_DUMP	=>	!TOOLS.NETWORKING.TCP_IP_DUMP
EXT:	TERMINAL	=>	!COMMANDS.TERMINAL
EXT:	TERMINAL_SPECIFIC	=>	!IO.TERMINAL_SPECIFIC
EXT:	TEXT	=>	!COMMANDS.TEXT
EXT:	TEXT_IO	=>	!IO.TEXT_IO
EXT:	TIME_UTILITIES	=>	!TOOLS.TIME_UTILITIES
EXT:	TRANSFER_GENERIC	=>	!TOOLS.NETWORKING.TRANSFER_GENERIC
EXT:	TRANSPORT	=>	!TOOLS.NETWORKING.TRANSPORT
EXT:	TRANSPORT_DEFS	=>	!TOOLS.NETWORKING.TRANSPORT_DEFS
EXT:	TRANSPORT_INTERCHANGE	=>	!TOOLS.NETWORKING.TRANSPORT_INTERCHANGE
EXT:	TRANSPORT_NAME	=>	!TOOLS.NETWORKING.TRANSPORT_NAME
EXT:	TRANSPORT_SERVER	=>	!TOOLS.NETWORKING.TRANSPORT_SERVER
EXT:	TRANSPORT_SERVER_PROC	=>	!TOOLS.NETWORKING.TRANSPORT_SERVER_PROC
EXT:	TRANSPORT_STREAM	=>	!TOOLS.NETWORKING.TRANSPORT_STREAM
EXT:	UNBOUNDED_STRING	=>	!TOOLS.UNBOUNDED_STRING
EXT:	UNCHECKED_CONVERSION	=>	!LRM.UNCHECKED_CONVERSION
EXT:	UNCHECKED_CONVERSIONS	=>	!TOOLS.UNCHECKED_CONVERSIONS
EXT:	UNCHECKED_DEALLOCATION	=>	!LRM.UNCHECKED_DEALLOCATION
EXT:	WHAT	=>	!COMMANDS.WHAT
EXT:	WINDOW_IO	=>	!IO.WINDOW_IO
EXT:	WORK_ORDER	=>	!COMMANDS.WORK_ORDER

!Model.R1000_Implementation Links

KIND	LINK	=>	SOURCE
EXT:	ACCESS_LIST	=>	!COMMANDS.ACCESS_LIST
EXT:	ACCESS_LIST_TOOLS	=>	!TOOLS.ACCESS_LIST_TOOLS
EXT:	ACTION	=>	!IMPLEMENTATION.ACTION
EXT:	ACTION_UTILITIES	=>	!COMMANDS.ACTION_UTILITIES
EXT:	ACTIVITY	=>	!COMMANDS.ACTIVITY
EXT:	ACTIVITY_IMPLEMENTATION	=>	!IMPLEMENTATION.ACTIVITY_IMPLEMENTATION
EXT:	ADA	=>	!COMMANDS.ADA
EXT:	ADA_OBJECT_EDITOR	=>	!TOOLS.ADA_OBJECT_EDITOR
EXT:	ADA_TEXT	=>	!TOOLS.ADA_TEXT
EXT:	ALLOWS_DEALLOCATION	=>	!TOOLS.ALLOWS_DEALLOCATION
EXT:	ARCHIVE	=>	!COMMANDS.ARCHIVE
EXT:	BOUNDED_STRING	=>	!TOOLS.BOUNDED_STRING
EXT:	BYTE_STRING_IO	=>	!TOOLS.NETWORKING.BYTE_STRING_IO
EXT:	CALENDAR	=>	!LRM.CALENDAR
EXT:	CMVC	=>	!COMMANDS.CMVC
EXT:	CMVC_IMPLEMENTATION	=>	!IMPLEMENTATION.CMVC_IMPLEMENTATION
EXT:	CMVC_IMPLEMENTATION_ERRORS	=>	!IMPLEMENTATION.CMVC_IMPLEMENTATION_ER
EXT:	CMVC_IMPLEMENTATION_UTILITIES	=>	!IMPLEMENTATION.CMVC_IMPLEMENTATION_UT
EXT:	CMVC_MAINTENANCE	=>	!COMMANDS.CMVC_MAINTENANCE
EXT:	COMMAND	=>	!COMMANDS.COMMAND
EXT:	COMMON	=>	!COMMANDS.COMMON
EXT:	COMPILATION	=>	!COMMANDS.COMPILATION
EXT:	CONCURRENT_MAP_GENERIC	=>	!TOOLS.CONCURRENT_MAP_GENERIC
EXT:	DAEMON	=>	!COMMANDS.DAEMON
EXT:	DEBUG	=>	!COMMANDS.DEBUG
EXT:	DEBUG_TOOLS	=>	!TOOLS.DEBUG_TOOLS
EXT:	DEFAULT	=>	!IMPLEMENTATION.DEFAULT
EXT:	DEPENDENCY_DATA_BASE	=>	!IMPLEMENTATION.DEPENDENCY_DATA_BASE
EXT:	DEVICE_INDEPENDENT_IO	=>	!IO.DEVICE_INDEPENDENT_IO
EXT:	DIANA	=>	!IMPLEMENTATION.DIANA
EXT:	DIANA_OBJECT_EDITOR	=>	!TOOLS.DIANA_OBJECT_EDITOR
EXT:	DIANA_RENAMES	=>	!IMPLEMENTATION.DIANA_RENAMES
EXT:	DIANA_TREE	=>	!COMMANDS.DIANA_TREE
EXT:	DIRECTORY	=>	!IMPLEMENTATION.DIRECTORY
EXT:	DIRECTORY_TOOLS	=>	!TOOLS.DIRECTORY_TOOLS
EXT:	DIRECT_IO	=>	!IO.DIRECT_IO
EXT:	DISK_DAEMON	=>	!TOOLS.DISK_DAEMON
EXT:	DISK_SPACE	=>	!COMMANDS.DISK_SPACE
EXT:	EDITOR	=>	!COMMANDS.EDITOR
EXT:	ERROR_MESSAGES	=>	!IMPLEMENTATION.ERROR_MESSAGES
EXT:	ERROR_REPORTING	=>	!IMPLEMENTATION.ERROR_REPORTING
EXT:	FILE_TRANSFER	=>	!TOOLS.NETWORKING.FILE_TRANSFER
EXT:	FILE_UTILITIES	=>	!COMMANDS.FILE_UTILITIES
EXT:	FTP	=>	!COMMANDS.FTP
EXT:	FTP_DEFS	=>	!TOOLS.NETWORKING.FTP_DEFS
EXT:	FTP_NAME_MAP	=>	!TOOLS.NETWORKING.FTP_NAME_MAP
EXT:	FTP_PROFILE	=>	!TOOLS.NETWORKING.FTP_PROFILE
EXT:	HASH	=>	!TOOLS.HASH
EXT:	HOST_ID_IO	=>	!TOOLS.NETWORKING.HOST_ID_IO
EXT:	INCREMENTAL	=>	!COMMANDS.INCREMENTAL
EXT:	INTERCHANGE	=>	!TOOLS.NETWORKING.INTERCHANGE
EXT:	INTERCHANGE_DEFS	=>	!TOOLS.NETWORKING.INTERCHANGE_DEFS
EXT:	IO	=>	!IO.IO
EXT:	IO_EXCEPTIONS	=>	!IO.IO_EXCEPTIONS
EXT:	JOB	=>	!COMMANDS.JOB
EXT:	JOB_SEGMENT	=>	!IMPLEMENTATION.JOB_SEGMENT
EXT:	LIBRARY	=>	!COMMANDS.LIBRARY
EXT:	LIBRARY_OBJECT_EDITOR	=>	!TOOLS.LIBRARY_OBJECT_EDITOR
EXT:	LIMIT_OPERATIONS	=>	!IMPLEMENTATION.LIMIT_OPERATIONS
EXT:	LINKS	=>	!COMMANDS.LINKS
EXT:	LINKS_IMPLEMENTATION	=>	!IMPLEMENTATION.LINKS_IMPLEMENTATION
EXT:	LINK_TOOLS	=>	!TOOLS.LINK_TOOLS
EXT:	LIST_GENERIC	=>	!TOOLS.LIST_GENERIC
EXT:	LOAD_VIEW	=>	!IMPLEMENTATION.LOAD_VIEW
EXT:	LOG	=>	!COMMANDS.LOG
EXT:	LOW_LEVEL_ACTION	=>	!IMPLEMENTATION.LOW_LEVEL_ACTION
EXT:	MACHINE	=>	!IMPLEMENTATION.MACHINE
EXT:	MACHINE_CODE	=>	!LRM.MACHINE_CODE
EXT:	MAP_GENERIC	=>	!TOOLS.MAP_GENERIC
EXT:	MESSAGE	=>	!COMMANDS.MESSAGE
EXT:	OBJECT_EDITOR	=>	!TOOLS.OBJECT_EDITOR
EXT:	OBJECT_SET	=>	!IO.OBJECT_SET
EXT:	OPERATOR	=>	!COMMANDS.OPERATOR
EXT:	PARAMETER_PARSER	=>	!TOOLS.PARAMETER_PARSER
EXT:	PIPE	=>	!IO.PIPE
EXT:	POLYMORPHIC_IO	=>	!IO.POLYMORPHIC_IO
EXT:	POLYMORPHIC_SEQUENTIAL_IO	=>	!IO.POLYMORPHIC_SEQUENTIAL_IO
EXT:	PRODUCT_AUTHORIZATION	=>	!IMPLEMENTATION.PRODUCT_AUTHORIZATION
EXT:	PROFILE	=>	!TOOLS.PROFILE
EXT:	PROGRAM	=>	!COMMANDS.PROGRAM
EXT:	QUEUE	=>	!COMMANDS.QUEUE
EXT:	QUEUE_GENERIC	=>	!TOOLS.QUEUE_GENERIC
EXT:	RANDOM	=>	!TOOLS.RANDOM
EXT:	RPC	=>	!TOOLS.NETWORKING.RPC
EXT:	RPC_CLIENT	=>	!TOOLS.NETWORKING.RPC_CLIENT
EXT:	RPC_SERVER	=>	!TOOLS.NETWORKING.RPC_SERVER
EXT:	SCHEDULER	=>	!COMMANDS.SCHEDULER
EXT:	SCRIPT	=>	!TOOLS.SCRIPT
EXT:	SEARCH_LIST	=>	!COMMANDS.SEARCH_LIST
EXT:	SEMANTIC_ATTRIBUTES	=>	!IMPLEMENTATION.SEMANTIC_ATTRIBUTES
EXT:	SEQUENTIAL_IO	=>	!IO.SEQUENTIAL_IO
EXT:	SET_GENERIC	=>	!TOOLS.SET_GENERIC
EXT:	SIMPLE_STATUS	=>	!TOOLS.SIMPLE_STATUS
EXT:	SNAPSHOT_NOTIFICATION	=>	!IMPLEMENTATION.SNAPSHOT_NOTIFICATION
EXT:	STACK_GENERIC	=>	!TOOLS.STACK_GENERIC
EXT:	STRING_MAP_GENERIC	=>	!TOOLS.STRING_MAP_GENERIC
EXT:	STRING_TABLE	=>	!TOOLS.STRING_TABLE
EXT:	STRING_UTILITIES	=>	!TOOLS.STRING_UTILITIES
EXT:	SWITCHES	=>	!COMMANDS.SWITCHES
EXT:	SWITCH_IMPLEMENTATION	=>	!IMPLEMENTATION.SWITCH_IMPLEMENTATION
EXT:	SYSTEM	=>	!LRM.SYSTEM
EXT:	SYSTEM_BACKUP	=>	!COMMANDS.SYSTEM_BACKUP
EXT:	SYSTEM_UTILITIES	=>	!TOOLS.SYSTEM_UTILITIES
EXT:	TABLE_FORMATTER	=>	!TOOLS.TABLE_FORMATTER
EXT:	TABLE_SORT_GENERIC	=>	!TOOLS.TABLE_SORT_GENERIC
EXT:	TAPE	=>	!COMMANDS.TAPE
EXT:	TAPE_SPECIFIC	=>	!IO.TAPE_SPECIFIC
EXT:	TAPE_TOOLS	=>	!TOOLS.TAPE_TOOLS
EXT:	TCP_IP_BOOT	=>	!TOOLS.NETWORKING.TCP_IP_BOOT
EXT:	TCP_IP_DUMP	=>	!TOOLS.NETWORKING.TCP_IP_DUMP
EXT:	TERMINAL	=>	!COMMANDS.TERMINAL
EXT:	TERMINAL_SPECIFIC	=>	!IO.TERMINAL_SPECIFIC
EXT:	TEXT	=>	!COMMANDS.TEXT
EXT:	TEXT_IO	=>	!IO.TEXT_IO
EXT:	TIME_UTILITIES	=>	!TOOLS.TIME_UTILITIES
EXT:	TRANSFER_GENERIC	=>	!TOOLS.NETWORKING.TRANSFER_GENERIC

```
EXT: TRANSPORT                    => !TOOLS.NETWORKING.TRANSPORT
EXT: TRANSPORT_DEFS               => !TOOLS.NETWORKING.TRANSPORT_DEFS
EXT: TRANSPORT_INTERCHANGE        => !TOOLS.NETWORKING.TRANSPORT_INTERCHANGE
EXT: TRANSPORT_NAME               => !TOOLS.NETWORKING.TRANSPORT_NAME
EXT: TRANSPORT_SERVER             => !TOOLS.NETWORKING.TRANSPORT_SERVER
EXT: TRANSPORT_SERVER_PROC        => !TOOLS.NETWORKING.TRANSPORT_SERVER_PROC
EXT: TRANSPORT_STREAM             => !TOOLS.NETWORKING.TRANSPORT_STREAM
EXT: UNBOUNDED_STRING             => !TOOLS.UNBOUNDED_STRING
EXT: UNCHECKED_CONVERSION         => !LRM.UNCHECKED_CONVERSION
EXT: UNCHECKED_CONVERSIONS        => !TOOLS.UNCHECKED_CONVERSIONS
EXT: UNCHECKED_DEALLOCATION       => !LRM.UNCHECKED_DEALLOCATION
EXT: UNIVERSAL                    => !IMPLEMENTATION.UNIVERSAL
EXT: WHAT                         => !COMMANDS.WHAT
EXT: WINDOW_IO                    => !IO.WINDOW_IO
EXT: WORK_ORDER                   => !COMMANDS.WORK_ORDER
EXT: WORK_ORDER_ERRORS            => !IMPLEMENTATION.WORK_ORDER_ERRORS
EXT: WORK_ORDER_IMPLEMENTATION    => !IMPLEMENTATION.WORK_ORDER_IMPLEMENTATION
```

!Model.R1000_Portable Links

KIND	LINK	=>	SOURCE
EXT:	CALENDAR	=>	!LRM.CALENDAR
EXT:	DIRECT_IO	=>	!IO.DIRECT_IO
EXT:	IO_EXCEPTIONS	=>	!IO.IO_EXCEPTIONS
EXT:	SEQUENTIAL_IO	=>	!IO.SEQUENTIAL_IO
EXT:	SYSTEM	=>	!LRM.SYSTEM
EXT:	TEXT_IO	=>	!IO.TEXT_IO
EXT:	UNCHECKED_CONVERSION	=>	!LRM.UNCHECKED_CONVERSION
EXT:	UNCHECKED_DEALLOCATION	=>	!LRM.UNCHECKED_DEALLOCATION

This section intentionally left blank.

Please use this section as a repository for quick-reference information describing locally-developed tools you use.

This section intentionally left blank.

Please use this section as a repository for release notes distributed by Rational.

Naming in the Rational Environment Library System

Two distinct forms of names are used in the library system: string names and Ada names. (A subset of the string naming mechanism is used only during debugging.) Every directory object has a string name; only Ada units have Ada names. String names must be enclosed in quotation marks.

String names include special names, wildcards, substitution characters, special characters, and attributes. String names assume a closed scope by default.

The following sections describe string names and Ada names.

STRING NAMES

Special Names

Special names are used as parameter values for many Environment operations to specify text, objects, and regions. Special names allow you to designate without providing a pathname. Listed below are the special names used in the Environment and their references:

"<SELECTION>"	References the highlighted object, if the cursor is in the highlighted area. Otherwise, an error will result.
"<REGION>"	References the highlighted object.
"<CURSOR>"	References the highlighted object, if the cursor is in the highlighted area. If the cursor is not located in the highlighted area, this special name refers to the object associated with the smallest selectable region surrounding the cursor.
"<IMAGE>"	References the highlighted object, if the cursor is in the highlighted area. If the cursor is not located in the highlighted area, this special name references the object corresponding to the image in which the cursor is located.
"<TEXT>"	References the object named in the highlighted text in the image in the window. This is equivalent to typing the contents of the highlighted area as the name.
"<ACTIVITY>"	References the default activity. If an activity is highlighted and the cursor is in the highlight, this special name references that activity rather than the default activity.

Wildcards

Wildcards allow for both the abbreviation of names and the specifying of several objects with one name. The wildcards are: pound sign (#), at sign (@), question mark (?), and double question mark (??).

The Wildcard

The pound sign (#) matches any single identifier character in a name, including the underscore (_). It can be used several times within a single name. For example, F### will match the name Food.

The Wildcard @

The at sign (@) matches zero or more identifier characters in a name, including the underscore (_). It does not match a period (.). The wildcard can be used several times within a single name. For example, the name !U@.@.Food matches the name !Users.Fred.Food.

The Wildcard ?

The question mark (?) matches zero or more components in a name that are not worlds or objects contained by those worlds. For example, consider the world below:

```
!Users.Stooges :   Library (World);
  Curly    : I Ada (Proc_Spec);
  Curly    : I Ada (Proc_Body);
    .Foo   : I Ada (Proc_Body);
  Larry    : I Ada (Proc_Spec);
  Larry    : I Ada (Proc_Body);
    .Bar   : I Ada (Proc_Body);
  Moe      : C Ada (Proc_Spec);
  Moe      : C Ada (Proc_Body);
```

The name !Users.Stooges? matches !Users.Stooges itself and units within that world called Larry, Curly, Moe, and any of their subunits. It would not match any worlds within !Users.Stooges.

The periods before and after this wildcard are optional. For example, the name A.?.B is equivalent to the name A?B.

The Wildcard ??

The double question mark () matches zero or more components in a name, including worlds or objects contained by those worlds. For example, the name !Users?? matches the home worlds of all users and the contents of those worlds. !Users.Bill?? matches everything in Bill's home world, including worlds and the objects within those worlds, and !Users.Bill itself. As another example, consider that !?? matches all objects in the directory system on a given machine.

The periods before and after this wildcard are optional. For example, the name A.??.B is equivalent to the name A??B.

Substitution Characters

The Environment also offers support for parameterization within a string name designating the target of an operation--for example, the destination of a Library.Copy. The target string parameters are called substitution characters and represent placeholders in the target string for portions of the source name. The substitution characters are pound sign (#), at sign (@), and question mark (?). Matching is performed from right to left, substituting character against wildcard.

The Substitution Character

The pound sign (#) is replaced by the next complete (right to left) segment in the source name.

The Substitution Character @

The at sign (@) is replaced by the portion of the source name that is matched by a wildcard in the source name. If there is more than one wildcard in the source name, a separate @ character is needed in the target to match each one.

The Substitution Character ?

The question mark (?) is replaced by successive full segments of the source name, working right to left, until the segment for a world is encountered.

Using the example:

```
!Users.Foo
  ...
  TRW      : Session;
  Foo_File : File;
  Gen_Data : Ada (Gen_Pack);
  Packet   : Ada (Pack_Spec);
  Formula  : Ada (Proc_Inst);
  Sub_Foo  : Library (Directory);
  ...

!User.Foo.Sub_Foo
  ...
  Gen_Data : File;
  Actor    : Ada (Func_Spec);
  Actor    : Ada (Func_Body);
  Steamer  : Ada (Pack_Spec);
  ...
```

the following uses of substitution characters can be demonstrated:

Source String	Target String	Target Names
Foo.Packet	Bar.#	Bar.Packet
Foo?Gen_@	#.Old_@_Save	Foo.Old_Data_Save Sub_Foo.Old_Data_Save
Foo?@m@	Bar?	Bar.Formula Bar.Sub_Foo.Steamer

Special Characters

Special characters can be used in names to specify either relative or absolute contexts or to specify indirect files of names. These special characters apply to names used throughout the Environment.

A special character in a name determines the context in which the remaining portion of the name will be interpreted. A special character of exclamation (!), caret (^), dollar sign ($), double dollar sign ($$), percent (%), underscore (_), period (.), backslash (\), or grave (`) causes explicit interpretations of the remainder of the name, as described below.

Character pairs are also used to enclose a name and to give that name an additional meaning. Character pairs are brackets ([]) and braces ({}), which are also described below.

The Special Character !

The exclamation mark (!) specifies that the context for resolving the remainder of the name should be set to the root of the library system. This creates a fully qualified name. This character represents the root of the library system in any context.

The Special Character ^

The caret (^) specifies that the context should be set to the immediately enclosing object. The caret permits naming to climb the hierarchy of objects and eventually reach the root of the library system. The caret prefix can be used repeatedly to define the context to be several units above the current context. The parent object of the root of the directory system is itself.

A special use of this character occurs in combination with a bracketed name. A name component of the form ^[some_object] resolves to the closest containing object whose simple name is Some_Object.

The caret is frequently used as a shorthand method for referring to objects in a parent unit.

The Special Character $

The dollar sign ($) specifies that the context should be set to the immediately enclosing library. If the current context is a library, this character has no effect.

A special use of this character occurs in combination with a bracketed name. A name component of the form $[some_library] resolves to the closest containing library whose simple name is Some_Library.

The Special Character $$

The double dollar sign ($$) specifies that the context should be set to the immediately enclosing world. This is more restrictive than the

single dollar sign ($), which is either a world or a directory. If the current context is a world, this character has no effect.

A special use of this character occurs in combination with a bracketed name. A name component of the form $$[some_world] resolves to the closest containing world whose simple name is Some_World.

The Special Character %

The percent (%), used only in the Rational Debugger, can be used only as the first character of a name. It specifies that the next name component is a task name. Task names are either string names assigned to tasks by calls to the !Commands.Debug.Set_Task_Name or the !Tools.Debug_Tools.Set_Task_Name procedure or task numbers assigned by the Environment. The !Commands.Debug.Task_Display procedure lists all tasks and their names and numbers.

The components of a name that follow the task name are interpreted as objects declared in the named task. If the task name is followed by _n (where n is a number), the name refers to a stack frame of the named task. Stack frame names are further discussed in "The Special Character _," below.

The Special Character _

The underscore (_) is interpreted as an indirect file prefix when used in Environment commands. If the first character after the underscore is an alphabetic character, it is assumed to be the first character of the name of a file that contains other names. This provides a way of building lists of objects and referring to that list in a name. The file may be a text or an activity file.

The underscore character is also interpreted as a stack frame prefix when used in the Rational Debugger. If the value of an object declared in a subprogram is to be named, the frame on the run-time stack that contains an activation of that subprogram must be named. Stack frames are numbered for each task, starting at the top with 1.

The Special Character .

The period (.) is used both as a name component separator and as a name prefix. As a separator, it is used just as in Ada names to separate components of a name.

As a prefix character, the period specifies that the first component of the name is a library unit name. This is used only in the Rational Debugger. A second component of the name would be an object declared in the named library unit.

The Special Character \

The backslash (\) specifies that the next name component be evaluated in the current searchlist.

The Special Character `

The grave (`) is used to evaluate names using the current context and the set of links associated with the current context. The grave evaluates the name as if it were the name of an Ada unit in a "with" clause of a unit in the library that contains the current context. For example, the name `Moe resolves to an Ada unit called Moe in the containing library. Moe could be a link to some other library.

The Special Characters []

Brackets ([]) define a set notation. Sets are created by enclosing a series of name components, separated by commas, in brackets. The semicolon character in set notation can also be used to separate name components. Commas and semicolons cannot be mixed. If semicolons are used, each name component in the set must resolve to at least one object.

Names can also be excluded from a set with the tilde (~).

The special string [] represents the current context, whether that context is a directory, world, Ada unit, or other object.

The Special Characters {}

Braces ({}) denote objects that have been deleted but not expunged.

Attributes

Attributes are special strings that specify a restriction on the evaluation of the name. Syntactically like Ada attributes, these strings are a postfix notation that specifies some restriction on the interpretation of the name. Specific versions of an object, specific classes of objects, either the visible part or the body of an Ada unit, or a nickname can be specified with attributes to remove ambiguity or to specify something other than the default interpretation of the name.

The following is a list of attributes and their meanings:

'body	Matches the body of the specified Ada object. The 'spec and 'body attributes are mutually exclusive and cover the space of all Ada objects.
'spec	Matches the specification part of a declaration. Single-part declaration objects are also matched by this attribute.
'L(External)link_name	Matches the named external link in the set of links associated with a world.
'L(Internal)link_name	Matches the named internal link in the set of links associated with a world.
'L(Any)link_name	Matches the named internal or external link in the set of links associated with a world.
'N(nickname)	Matches the object whose assigned nickname is the same as the nickname specified.

State Attributes

'S(Archived)	Matches units in the archived state.
'S(Source)	Matches units in the source state.
'S(Installed)	Matches units in the installed state.
'S(Coded)	Matches units in the coded state.

Version Attributes

'V(ALL)	Matches all versions of the object.
'V(ANY)	Matches the default version of the object.
'V(MAX)	Matches the newest version of the object.
'V(MIN)	Matches the oldest version of the object.
'V(n)	Matches the version with that version number.
'V(-n)	Matches the nth version preceding the current version.

Class Attributes

'C(ADA)	Matches any Ada program unit.
'C(ARCHIVED_CODE)	Matches objects appearing in a subsystem view for a code-only unit.
'C(FILE)	Matches any file.
'C(GROUP)	Matches any group in the system.
'C(LIBRARY)	Matches any directory, world, or subsystem.
'C(NULL_DEVICE)	Matches a device that accepts output and discards it.
'C(PIPE)	Matches any pipe.
'C(SESSION)	Matches any user's session object.
'C(TAPE)	Matches any tape drive in the system.
'C(TERMINAL)	Matches any terminal in the system.
'C(USER)	Matches any user in the system.

Subclass Attributes of Library Class Objects

'C(Comb_Ss)	Matches any subsystem containing combined view that cannot contain spec or load views.
'C(Comb_View)	Matches any combined view of a subsystem.
'C(Directory)	Matches any directory.
'C(Load_View)	Matches any load view of a subsystem.
'C(Mailbox)	Matches any library containing Mail and Mail_Db files for the Rational Mail Utility.
'C(Spec_Load)	Matches any subsystem that cannot contain combined views.
'C(Spec_View)	Matches any spec view of a subsystem.
'C(Subsystem)	Matches any subsystem.
'C(World)	Matches any world.

Subclass Attributes of Ada Class Objects

'C(Alt_List)	Matches any alternative list insertion point.
'C(Comp_Unit)	Matches any compilation unit that has not been semanticized.
'C(Context)	Matches any context clause insertion point.
'C(Decl_List)	Matches any declaration list insertion point.
'C(Func_Body)	Matches any function body.
'C(Func_Inst)	Matches any generic function instantiation.
'C(Func_Ren)	Matches any function rename.
'C(Func_Spec)	Matches any function specification.
'C(Gen_Func)	Matches any generic function.
'C(Gen_Pack)	Matches any generic package.
'C(Gen_Param)	Matches any generic parameter insertion point.
'C(Gen_Proc)	Matches any generic procedure.
'C(Insertion)	Matches any insertion point.
'C(Load_Func)	Matches any code-only function.
'C(Load_Proc)	Matches any code-only procedure.
'C(Main_Body)	Matches any main function body.
'C(Main_Body)	Matches any main procedure body.
'C(Main_Func)	Matches any main function specification.
'C(Main_Proc)	Matches any main procedure specification.
'C(Pack_Body)	Matches any package body.
'C(Pack_Inst)	Matches any generic package instantiation.
'C(Pack_Ren)	Matches any package rename.
'C(Pack_Spec)	Matches any package specification.
'C(Pragma)	Matches any pragma insertion point.
'C(Proc_Body)	Matches any procedure body.
'C(Proc_Inst)	Matches any generic procedure instantiation.
'C(Proc_Ren)	Matches any procedure rename.
'C(Proc_Spec)	Matches any procedure spec.
'C(Statement)	Matches any statement insertion point.
'C(Subp_Body)	Matches any subprogram body.
'C(Subp_Inst)	Matches any generic subprogram instantiation.
'C(Subp_Ren)	Matches any subprogram rename.
'C(Subp_Spec)	Matches any subprogram specification.
'C(Task_Body)	Matches any task body.

Subclass Attributes of File Class Objects

'C(Activity)	Matches any activity file.
'C(Binary)	Matches any binary file.
'C(Cmvc_Db)	Matches any CMVC database.
'C(Code_Db)	Matches any code saved for a subsystem load view.
'C(Compat_Db)	Matches any compatibility database for a subsystem.
'C(Config)	Matches any configuration pointer for CMVC.
'C(Dictionary)	For future development.
'C(Documents)	Matches any document database.
'C(File_Map)	Matches any file map.
'C(Log)	Matches any log file.
'C(Mail)	Matches any collection of messages.
'C(Mail_Db)	Matches any user'C mailbox.
'C(Msg_In)	For future development.
'C(Msg_Out)	For future development.
'C(Objects)	Matches any object set.
'C(Ps)	Matches any PostScript file.
'C(Search)	Matches any searchlist file.
'C(Switch)	Matches any switch file.
'C(Swtch_Def)	Matches any switch definition file.
'C(Text)	Matches any text file.
'C(Venture)	Matches any collection of work orders for CMVC.
'C(Work)	Matches any work order for CMVC.
'C(Work_List)	Matches any work order list for CMVC.

A list of subclasses can also be found in!Implementation.Object_Subclass.

ADA NAMES

Ada names are implemented as defined by the LRM. Link names can be used to reference library units across world boundaries. Worlds, directories, and files cannot be referenced by Ada names.

In locations other than context clauses, Ada names are always resolved in the context of a library unit and its associated "with" clauses. The simple names in "with" clauses are resolved in a two-step process. First, the immediately containing directory or world is searched for a unit of the same name. If that search fails, the link pac of the closest containing world is searched for the name. If that search fails, the name is undefined.

Search Characters

Packages File_Utilities and Editor.Search both contain procedures that allow regular expression pattern matching. The patterns may include the use of the special wildcard characters below. The special interpretation of these characters is enabled by a parameter in packages File_Utilities and Editor.Search procedures. When the special characters are not enabled, they are interpreted literally, just like other characters.

The special wildcard characters for regular expression pattern matching are described below:

? Matches any single character.
% Matches any single character that is legal in an Ada identifier.
# Matches the following characters, which are frequently used as Ada delimiters:
&'()*+,-./:;<=>|
\ When not at the end of the pattern, causes the character immediately following this wildcard to be interpreted as a normal (not a wildcard) character.
{ When at the beginning of the pattern, requires the pattern to match the beginning of the line.
} When at the end of the pattern, requires the pattern to match the end of the line.
[] Used around a string of characters, matches any one of the enclosed characters. Each character to be compared must be specified explicitly or by a range (for example, [A-Z]).
^ Matches anything except the character (or characters, if inside brackets ([])) following this wildcard. If used inside brackets ([]), this wildcard must be the first character in the list.
* Matches zero or more occurrences of the previous characters or set of characters.

Example 1

The command File_Utilities.Find("section 3","Test"); prints the line(s) containing the string "section 3" in the specified file.

Example 2

The command File_Utilites.Find("%*","Test",True); prints the line(s) containing a string of any length that is a legal Ada identifier.

Example 3

The command File_Utilities.Find("[^abc]","Test",True); prints the line(s) that start with characters other than 'a', 'b', or 'c' in the specified file.

Banner Symbols

The Editor maintains the banner of each window it displays. On the left end of the banner is a symbol. This single character represents the state of the image in the window. The following table provides the typical interpretation of these symbols:

Symbol	Lock	Committed?	Formatted?
=	Read	N.A.	True
(Blank)	Write	True	Can be either
*	Write	False	False
#	Write	False	True
!	(The Editor or a job has temporarily locked the image.)		

Note that, for I/O windows, the * and # symbols mean that a job that is requesting input is running. Because such images are considered uncommitted, you cannot log off without terminating these jobs or ignoring changed images.

Library Switch Definitions and Default Values

PROCESSOR	SWITCH	TYPE	VALUE
Ftp	Account	String	:= ""
Format	Alignment_Threshold	Line_Range	:= 0
R1000_Cg	Asm_Listing	Boolean	:= False
Cross_Cg	Asm_Source	Boolean	:= False
Cross_Cg	Auto_Assemble	Boolean	:= True
Cross_Cg	Auto_Link	Boolean	:= True
Ftp	Auto_Login	Boolean	:= False
Semantics	Closed_Private_Part	Boolean	:= False
Format	Comment_Column	Line_Range	:= 1
Parser	Configuration	Switch_Set	:= ()
Format	Consistent_Breaking	Integer	:= 1
Directory	Create_Internal_Links	Boolean	:= True
Directory	Create_Subprogram_Specs	Boolean	:= True
Cross_Cg	Debugging_Level	Debug_Level	:= None
R1000_Cg	Elab_Order_Listing	Boolean	:= False
R1000_Cg	Enable_Deallocation	Boolean	:= False
Cross_Cg	Flags	String	:= ""
Format	Id_Case	Letter_Case	:= Capitalized
Semantics	Ignore_Interface_Pragmas	Boolean	:= False
Semantics	Ignore_Minor_Errors	Boolean	:= False
Semantics	Ignore_Unsupported_Rep_Specs	Boolean	:= False
Format	Keyword_Case	Letter_Case	:= Lower
Format	Line_Length	Line_Range	:= 80
Cross_Cg	Linker_Command_File	String	:= ""
Cross_Cg	Listing	Boolean	:= False
Cross_Cg	Magic	String	:= ""
Format	Major_Indentation	Indent_Range	:= 4
Format	Minor_Indentation	Indent_Range	:= 4
Format	Number_Case	Letter_Case	:= Upper
Cross_Cg	Object_Libraries	String	:= ""
Cross_Cg	Optimization_Level	Integer	:= 0
R1000_Cg	Page_Limit	Integer	:= 0
Ftp	Password	String	:= ""
Ftp	Prompt_For_Account	Boolean	:= False
Ftp	Prompt_For_Password	Boolean	:= False
Ftp	Remote_Directory	String	:= ""
Ftp	Remote_Machine	String	:= ""
Ftp	Remote_Roof	String	:= ""
Ftp	Remote_Type	String	:= ""
Directory	Require_Internal_Links	Boolean	:= True
R1000_Cg	Seg_Listing	Boolean	:= False
Ftp	Send_Port_Enabled	Boolean	:= True
Format	Statement_Indentation	Indent_Range	:= 3
Format	Statement_Length	Line_Range	:= 35
Semantics	Subsystem_Interface	Boolean	:= False
Cross_Cg	Suppress_All_Checks	Boolean	:= False
R1000_Cg	Terminal_Echo	Boolean	:= False
Ftp	Transfer_Mode	Mode_Code	:= Nil
Ftp	Transfer_Structure	Structure_Code	:= Nil
Ftp	Transfer_Type	Type_Code	:= Nil
Ftp	Username	String	:= ""
Format	Wrap_Indentation	Line_Range	:= 16

Session Switch Definitions and Default Values

PROCESSOR	SWITCH	TYPE	VALUE
Session_Ftp	Account	String	:= ""
Session_Ftp	Auto_Login	Boolean	:= False
Queue	Banner	String	:= "<user_id>"
Session	Beep_On_Errors	Boolean	:= True
Session	Beep_On_Interrupt	Boolean	:= True
Session	Beep_On_Messages	Boolean	:= False
Session	Cmvc_Break_Long_Lines	Boolean	:= True
Session	Cmvc_Capitalize	Boolean	:= True
Session	Cmvc_Comment_Extent	Integer	:= 4
Session	Cmvc_Configuration_Extent	Integer	:= 0
Session	Cmvc_Enable_Relocation	Boolean	:= True
Session	Cmvc_Field_Extent	Integer	:= 4
Session	Cmvc_Indentation	Integer	:= 2
Session	Cmvc_Line_Length	Integer	:= 80
Session	Cmvc_Shorten_Name	Boolean	:= True
Session	Cmvc_Shorten_Unit_State	Boolean	:= False
Session	Cmvc_Show_Add_Date	Boolean	:= True
Session	Cmvc_Show_Add_Time	Boolean	:= True
Session	Cmvc_Show_All_Default_Lists	Boolean	:= False
Session	Cmvc_Show_All_Default_Orders	Boolean	:= False
Session	Cmvc_Show_Deleted_Objects	Boolean	:= False
Session	Cmvc_Show_Deleted_Versions	Boolean	:= False
Session	Cmvc_Show_Display_Position	Boolean	:= False
Session	Cmvc_Show_Edit_Info	Boolean	:= False
Session	Cmvc_Show_Field_Default	Boolean	:= True
Session	Cmvc_Show_Field_Max_Index	Boolean	:= False
Session	Cmvc_Show_Field_Type	Boolean	:= False
Session	Cmvc_Show_Frozen	Boolean	:= False
Session	Cmvc_Show_Hidden_Fields	Boolean	:= False
Session	Cmvc_Show_Retention	Boolean	:= False
Session	Cmvc_Show_Size	Boolean	:= False
Session	Cmvc_Show_Unit_State	Boolean	:= True
Session	Cmvc_Show_Users	Boolean	:= False
Session	Cmvc_Show_Version_Number	Boolean	:= False
Session	Cmvc_Uppercase	Boolean	:= False
Session	Cmvc_Version_Extent	Integer	:= 0
Session	Cursor_Bottom_Offset	Integer	:= 33
Session	Cursor_Left_Offset	Integer	:= 33
Session	Cursor_Right_Offset	Integer	:= 33
Session	Cursor_Top_Offset	Integer	:= 33
Session	Cursor_Transpose_Moves	Boolean	:= False
Session	Debug_Addresses	Boolean	:= False
Session	Debug_Break_At_Creation	Boolean	:= False
Session	Debug_Declaration_Display	Boolean	:= True
Session	Debug_Delete_Temporary_Breaks	Boolean	:= False
Session	Debug_Display_Count	Integer	:= 10
Session	Debug_Display_Creation	Boolean	:= False
Session	Debug_Display_Level	Integer	:= 3
Session	Debug_Echo_Commands	Boolean	:= True
Session	Debug_Element_Count	Integer	:= 25
Session	Debug_First_Element	Integer	:= 0
Session	Debug_Freeze_Tasks	Boolean	:= False
Session	Debug_History_Count	Integer	:= 10
Session	Debug_History_Entries	Integer	:= 1000
Session	Debug_History_Start	Integer	:= 10
Session	Debug_Include_Packages	Boolean	:= False
Session	Debug_Interpret_Control_Words	Boolean	:= False
Session	Debug_Kill_Old_Jobs	Boolean	:= True
Session	Debug_Machine_Level	Boolean	:= False
Session	Debug_Memory_Count	Integer	:= 3
Session	Debug_No_History_Timestamps	Boolean	:= True
Session	Debug_Optimize_Generic_History	Boolean	:= True
Session	Debug_Permanent_Breakpoints	Boolean	:= True
Session	Debug_Pointer_Level	Integer	:= 3
Session	Debug_Put_Locals	Boolean	:= False
Session	Debug_Qualify_Stack_Names	Boolean	:= False
Session	Debug_Require_Debug_Off	Boolean	:= False
Session	Debug_Save_Exceptions	Boolean	:= False
Session	Debug_Show_Location	Boolean	:= True
Session	Debug_Stack_Count	Integer	:= 10
Session	Debug_Stack_Start	Integer	:= 1
Session	Debug_Timestamps	Boolean	:= False
Session	Default_Job_Page_Limit	Integer	:= 8000
Cmvc	Default_Venture	Venture_Name	:= <>
Telnet	Escape	String	:= ""
Telnet	Escape_On_Break	Boolean	:= True
Queue	Footer	String	:= ""
Queue	Header	String	:= ""
Session	Image_Fill_Column	Integer	:= 72
Session	Image_Fill_Extra_Space	Text	:= "!.?"
Session	Image_Fill_Indent	Integer	:= -1
Session	Image_Fill_Mode	Boolean	:= False
Session	Image_Fill_Prefix	Text	:= ""
Session	Image_Insert_Mode	Boolean	:= True
Session	Image_Tab_Stops	Text	:= ""
Session	Job_Context_First	Boolean	:= True
Session	Job_Context_Length	Integer	:= 3
Session	Job_Name_Length	Integer	:= 2
Session	Job_Name_Separator	Text	:= " % "
Session	Key_Directory	Text	:= ""
Session	Library_Break_Long_Lines	Boolean	:= True
Session	Library_Capitalize	Boolean	:= True
Session	Library_Indentation	Integer	:= 2
Session	Library_Lazy_Realignment	Boolean	:= True
Session	Library_Line_Length	Integer	:= 80
Session	Library_Misc_Show_Edit_Info	Boolean	:= True
Session	Library_Misc_Show_Frozen	Boolean	:= True
Session	Library_Misc_Show_Retention	Boolean	:= True
Session	Library_Misc_Show_Size	Boolean	:= True
Session	Library_Misc_Show_Subclass	Boolean	:= False
Session	Library_Misc_Show_Unit_State	Boolean	:= True
Session	Library_Misc_Show_Volume	Boolean	:= True
Session	Library_Shorten_Names	Boolean	:= True
Session	Library_Shorten_Subclass	Boolean	:= True
Session	Library_Shorten_Unit_State	Boolean	:= True
Session	Library_Show_Deleted_Objects	Boolean	:= False
Session	Library_Show_Deleted_Versions	Boolean	:= False
Session	Library_Show_Miscellaneous	Boolean	:= False
Session	Library_Show_Standard	Boolean	:= False
Session	Library_Show_Subunits	Boolean	:= True
Session	Library_Show_Version_Number	Boolean	:= False
Session	Library_Std_Show_Class	Boolean	:= True
Session	Library_Std_Show_Subclass	Boolean	:= True
Session	Library_Std_Show_Unit_State	Boolean	:= False

```
     Session . Library_Uppercase              : Boolean         := False
     Profile . Log_At_Sign_Msgs               : Boolean         := True
     Profile . Log_Auxiliary_Msgs             : Boolean         := True
     Profile . Log_Diagnostic_Msgs            : Boolean         := False
     Profile . Log_Dollar_Msgs                : Boolean         := True
     Profile . Log_Error_Msgs                 : Boolean         := True
     Profile . Log_Exception_Msgs             : Boolean         := True
     Profile . Log_File                       : Output_File     := Use_Output
     Profile . Log_Line_Width                 : Natural         := 77
     Profile . Log_Negative_Msgs              : Boolean         := True
     Profile . Log_Note_Msgs                  : Boolean         := True
     Profile . Log_Position_Msgs              : Boolean         := True
     Profile . Log_Positive_Msgs              : Boolean         := True
     Profile . Log_Prefix_1                   : Log_Prefix      := Yr_Mn_Dy
     Profile . Log_Prefix_2                   : Log_Prefix      := Hr_Mn_Sc
     Profile . Log_Prefix_3                   : Log_Prefix      := Symbols
     Profile . Log_Sharp_Msgs                 : Boolean         := True
     Profile . Log_Warning_Msgs               : Boolean         := True
     Session . Mail_Multiple_Message_Windows  : Boolean         := False
     Session . Notify_Warnings                : Boolean         := False
       Queue . Options                        : String          :=
                                                "Format=>(Wrap, System_Header)"
  Session_Ftp . Password                      : String          := ""
     Session . Prompt_Delimiters              : Text            := """:"
  Session_Ftp . Prompt_For_Account            : Boolean         := False
  Session_Ftp . Prompt_For_Password           : Boolean         := False
     Profile . Reaction                       : Error_Reaction  := Persevere
  Session_Ftp . Remote_Directory              : String          := ""
  Session_Ftp . Remote_Machine                : String          := ""
       Telnet . Remote_Machine                : String          := ""
*    Profile . Remote_Passwords               : Object          := <Nil>
  Session_Ftp . Remote_Roof                   : String          := ""
*    Profile . Remote_Sessions                : Object          := <Nil>
  Session_Ftp . Remote_Type                   : String          := ""
     Session . Screen_Dump_File               : Text            :=
                                                  "SCREEN_DUMP"
     Session . Search_Ignore_Case             : Boolean         := True
     Session . Search_Preserve_Case           : Boolean         := False
     Session . Search_Regular_Expr            : Boolean         := False
  Session_Ftp . Send_Port_Enabled             : Boolean         := True
     Session . Terminal_Line_Speed            : Integer         := 9600
     Session . Terminal_Padding               : Integer         := 10
     Session . Text_Bottom_Stripe             : Text            :=
"-------------------------------------------------------------------------------"
     Session . Text_Convert_Tabs              : Boolean         := True
     Session . Text_Header                    : Boolean         := True
     Session . Text_Print_Name                : Boolean         := True
     Session . Text_Print_Number              : Boolean         := True
     Session . Text_Print_Time                : Boolean         := True
     Session . Text_Reuse_Window              : Boolean         := True
     Session . Text_Scroll_Output             : Boolean         := True
     Session . Text_Top_Stripe                : Text            :=
"-------------------------------------------------------------------------------"
  Session_Ftp . Transfer_Mode                 : Mode_Code       := Nil
  Session_Ftp . Transfer_Structure            : Structure_Code  := Nil
  Session_Ftp . Transfer_Type                 : Type_Code       := Nil
  Session_Ftp . Username                      : String          := ""
     Session . Window_Command_Size            : Integer         := 0
     Session . Window_Frames                  : Integer         := 0
     Session . Window_Frames_Startup          : Integer         := 0
```

```
     Session . Window_Have_Sides              : Boolean         := False
     Session . Window_Is_Staggered            : Boolean         := True
     Session . Window_Message_Life            : Integer         := 2
     Session . Window_Message_Size            : Integer         := 2
     Session . Window_Scroll_Overlap          : Integer         := 20
     Session . Window_Shift_Overlap           : Integer         := 80
     Session . Word_Breaks                    : Text            :=
                                                " ""#%&'()*+,-./:;<=>?[]_`{|}~"
```

This section intentionally left blank.

Please use this section as a repository for specifications of units you use for system programming not included elsewhere in the Reference Summary.

RATIONAL

READER'S COMMENTS

Note: This form is for documentation comments only. You can also submit problem reports and comments electronically by using the SIMS problem-reporting system. If you use SIMS to submit documentation comments, please indicate the manual name, book name, and page number.

Did you find this book understandable, usable, and well organized? Please comment and list any suggestions for improvement.

_____
_____
_____
_____
_____
_____

If you found errors in this book, please specify the error and the page number. If you prefer, attach a photocopy with the error marked.

_____
_____
_____
_____

Indicate any additions or changes you would like to see in the index.

_____
_____
_____
_____

How much experience have you had with the Rational Environment?

6 months or less _____ 1 year _____ 3 years or more _____

How much experience have you had with the Ada programming language?

6 months or less _____ 1 year _____ 3 years or more _____

Name (optional)_____ Date_____
Company _____
Address _____
City _____ State _____ ZIP Code _____

Please return this form to:

Publications Department
Rational
1501 Salado Drive
Mountain View, CA 94043

Rational Environment
Reference Manual

Keymap: Facit Terminal

Document Control Number: 8001A-51 (803-002324)

Rev. 6.0, November 1985
Rev. 6.1, March 1986
Rev. 6.2, July 1986
Rev. 7.0, July 1987 (Delta)

This document subject to change without notice.

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

Rational
1501 Salado Drive
Mountain View, California 94043

Contents

RATIONAL

How to Use the Keymap

The Rational Environment Keymap is the primary reference guide describing the keys that have been bound to Environment commands. Users have the option of modifying these key bindings for their own use, following procedures described in Rational Environment Basic Operations.

Note that there is a more basic reference to Environment key bindings in the Rational Environment Basic Keymap, in the Rational Environment Basic Operations. It is intended as the primary key reference for new Environment users.

Keymap Overview

The Keymap has been divided into the following three sections. The first and third sections apply to the Facit terminal only. The second section includes key bindings for both the Facit terminal and the Rational Terminal.

Detailed Reference to Key Bindings

The Detailed Reference provides a nearly complete list of key combinations, organized by topic and subtopic. The Detailed Reference entry for each key combination includes:

- A brief description of what the combination does
- The full name of the command that is bound to it
- Alternative key bindings, including accelerated key combinations (see "Basic and Accelerated Keystrokes," below

Master Reference to Key Bindings by Command

This section provides a complete, alphabetic list of the commands that are bound to keys on both the Facit terminal and the Rational Terminal. Each entry includes:

- The full name of an Environment command
- The key combination(s) to which the command is bound on the Facit terminal
- The key combination(s) to which the command is bound on the Rational Terminal

Master Reference to Key Bindings by Key

This section provides a complete table of the commands that are bound to keys on the Facit terminal. This table is in the form of an Ada procedure with nested case statements describing each of the key combinations. It is the same as the online keymap definition stored in !Machine.Editor_Data.Facit_Commands.

Environment Key Combinations

Environment commands are bound to two types of key combinations:

- Item-operation combinations
- Modified key combinations

These two types of key combinations differ in how they are executed.

Item-Operation Key Combinations

Each item-operation key combination contains an item key ([Esc], [Object], [Region], [Window], [Image], [Line], [Word], or [Mark]) followed by an operation key (either alphabetic or nonalphabetic). The item key identifies the item affected by the operation; the operation key identifies the action that applies to the indicated item.

The keystrokes must be sequential in an item-operation key combination. To execute an item-operation key:

1. Press and release the item key.
2. Press and release the operation key.

The notation indicates sequential keystrokes by separating them with a hyphen: [item key] - [operation key].

Patterns among Item-Operation Combinations

In general, commands that execute similar operations are bound to combinations that contain a common operation key. Some examples include:

[item] - [C] — Commands that copy items are bound to combinations such as [Line] - [C], [Region] - [C], and [Object] - [C], which share the operation key [C].

[item] - [D] — Commands that delete items are bound to combinations such as [Line] - [D], [Word] - [D], and [Window] - [D], which share the operation key [D].

[item] - [T] — Commands that transpose items are bound to combinations such as [Word] - [T], [Line] - [T], and [Window] - [T], which share the operation key [T].

Modified Key Combinations

Each modified key combination contains one or more modifier keys ([Shift], [Control]), along with another key (either alphabetic or nonalphabetic). Modifier keys are never used with item keys.

The keystrokes must overlap in a modified key combination. To execute a modified combination:

1. Press and hold the modifier key(s).
2. While holding down the modifier key(s), press the key to be modified.

The notation indicates overlapping keystrokes by naming the keys adjacently: [modifier key][other key].

Basic and Accelerated Keystrokes

Certain key combinations (namely, item-operation combinations and modified function keys) are considered *basic* combinations because they involve explicitly labeled keys, such as [Word] or [Definition]. Basic key bindings are recommended if you are new to the Environment, because they are easy to remember.

However, experienced users may find *accelerated* key bindings more convenient. Accelerated bindings generally involve the modifier keys in combination with keys on the main keyboard so that you can use them without moving your hands away from normal typing position.

Many commands are bound to both basic and accelerated key combinations. As an example, you can delete a word using either [Word] - [D] or the corresponding accelerated key combination, [Esc] - [D].

Keymap Notation

The following notations apply to the "Detailed Reference to Key Bindings" sections of the Keymap.

Symbols

[key1] - [key2]	Press and release [key1]; then press [key2].
[key1][key2]	Press and hold [key1] while pressing [key2].
[numeric 1]	Press [1] on the numeric keypad.

Numeric Arguments

You can give a numeric argument to many of the commands that are bound to keys. Indicate the desired number using the numeric keypad, and then press the key combination bound to the command. For example, [Word] - [D] deletes one word; the following combination deletes four words: [numeric 4] - [Word] - [D].

Indicate negative numbers by pressing [numeric -] first. For example, the following combination shrinks a window by seven lines ("expands" it by −7 lines):

[numeric -] - [numeric 7] - [Window] - [!]

Case Sensitivity of Key Bindings

Although keys are shown as uppercase, the unshifted equivalent also works. This is true for the nonalphabetic characters as well. For example, [Object] - [d] is equivalent to [Object] - [D] and [Object] - [!] is equivalent to [Object] - [1].

Detailed Reference to Key Bindings

Getting Help and Other Information

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Determine what help is available	Help on Help		What.Does
Get help on item	Help		What.Does
Get help on key	Help on Key	Esc · ⊕	Editor.Key.Name
Display Help window	Help Window		Editor.Image.Find
Explain underlined error	Object · ?		Common.Explain
Show time and date	What Time		What.Time
Show system load	What Load		What.Load (True)
Show current users	What Users		What.Users (True)
Show lock information for object in window	What Locks		What.Locks
Show full name of object in window	What Object		What.Object
Show access list for designated object	Show Access List		Access_List.Display

Traversing the Environment

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Display the Window Directory	Window · Definition	Window · ?	Editor.Window.Directory
Display object cursor is on	Definition		Common.Definition
Display object, same window	Definition In Place		Common.Definition
Display parent object	Enclosing		Common.Enclosing
Display parent object, same window	Enclosing In Place		Common.Enclosing
Display parent library, same window	Enclosing Library		Common.Enclosing
Display your home library	Esc · ↑		What.Home_Library
Set mark at current location	Mark · ↓		Editor.Mark.Push
Cycle through marks in stack	Mark · →	Esc · M	Editor.Mark.Next
Cycle back through marks in stack	Mark · ←		Editor.Mark.Previous
Return to most recent mark	Mark · ↑		Editor.Mark.Top

Logging Off

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Log off, unless changes aren't saved	–	–	Editor.Quit
Log off, ignoring unsaved changes	–	–	Editor.Quit(True)

Selecting Items

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Select successively larger structures	Object · ←		Common.Object.Parent
Select successively smaller structures	Object · →		Common.Object.Child
Select previous structure, same level	Object · ↑		Common.Object.Previous
Select next structure, same level	Object · ↓		Common.Object.Next
Select first structure	Object · Begin Of	Object · B	Common.Object.First_Child
Select last structure	Object · End Of	Object · E	Common.Object.Last_Child
Turn off selection cursor is in	Control X		Editor.Set.Designation_Off

Executing Commands

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Create a Command window	Create Command		Common.Create_Command
Complete command name and parameters	Complete		Common.Complete
Execute a command	Promote		Common.Promote
Execute command in background	Shift Promote		Command.Spawn
Move to the next parameter prompt	Esc · N		Editor.Cursor.Next
Move to the previous parameter prompt	Esc · U		Editor.Cursor.Previous
Turn a prompt into text	Control X		Editor.Set.Designation_Off
Redisplay the previous command (undo)	Object · U		Common.Undo
Redisplay the next command (redo)	Object · R		Common.Redo
Provide prompts for the next key pressed	Esc · Q		Editor.Key.Prompt

Managing Windows

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Moving between Windows			
Move to the next window	Window · ↓	Esc · V, Shift ↓	Editor.Window.Next
Move to the previous window	Window · ↑	Esc · Z, Shift ↑	Editor.Window.Previous
Move to next attached window	Window · →		Editor.Window.Child
Move to previous attached window	Window · ←		Editor.Window.Parent

Managing Windows *(Continued)*

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Resising and Repositioning Windows			
Join with the next window	Window · J		**Editor.Window.Join (1)**
Join with the previous window	Window · Delete		**Editor.Window.Join (-1)**
Expand a window 4 lines	Window · !		**Editor.Window.Expand**
Shrink a window 4 lines	Window · -		**Editor.Window.Expand (-4)**
Transpose current window with previous	Window · T		**Editor.Window.Transpose**
Realign windows	Window · Format		**Editor.Window.Focus**
Copy a window	Window · C		**Editor.Window.Copy**
Redrawing the Screen			
Redraw the screen	Control L		Editor.Screen.Redraw
Erase the screen, resetting the terminal	Esc · L		Editor.Screen.Clear
Retaining Windows			
Lock a window on the screen	Window · Promote		Editor.Window.Promote
Release a locked window	Window · Demote	Window · Edit	Editor.Window.Demote
Removing Windows			
Remove a window temporarily	Window · D, Window · K, Window · X		Editor.Window.Delete
Release image, discarding changes	Object · G		Common.Abandon
Release image, saving changes	Object · X		Common.Release
Delete selected Window Directory entry	Object · D		Common.Object.Delete
Finding Windows			
Display Window Directory	Window · Definition	Window · ?	Editor.Window.Directory
Display Window Directory entry	Definition		Common.Definition

Moving within an Image

Description	*Basic Keys*	*Accelerated Keys*	*Command*
By Character			
Move right 1 character	→	Control J	Editor.Cursor.Right
Move right 8 characters	numeric 8 · →	Esc · Control J	Editor.Cursor.Right(8)
Move left 1 character	←	Control H	Editor.Cursor.Left
Move left 8 characters	numeric 8 · ←	Esc · Control H	Editor.Cursor.Left(8)
By Word			
Move to next word	Word · →	Esc · J	Editor.Word.Next
Move to previous word	Word · ←	Esc · H	Editor.Word.Previous
Move to beginning of word	Word · Begin Of	Esc · A, Esc · B	Editor.Word.Beginning_Of
Move to end of word	Word · End Of	Esc · E	Editor.Word.End_Of
By Underline or Prompt			
Move to next underline or prompt	Esc · N		Editor.Cursor.Next
Move to previous underline or prompt	Esc · U		Editor.Cursor.Previous
By Line			
Move up 1 line	↑	Control U	Editor.Cursor.Up
Move up 8 lines	numeric 8 · ↑	Esc · Control U	Editor.Cursor.Up(8)
Move down 1 line	↓	Control N	Editor.Cursor.Down
Move down 8 lines	numeric 8 · ↓	Esc · Control N	Editor.Cursor.Down(8)
Move to beginning of line	Line · Begin Of	Control B	Editor.Line.Beginning_Of
Move to end of line	Line · End Of	Control E	Editor.Line.End_Of

Moving within an Image *(Continued)*

Description	*Basic Keys*	*Accelerated Keys*	*Command*
In a Region			
Move to beginning of region	Region · Begin Of	Region · B	Editor.Region.Beginning_Of
Move to end of region	Region · End Of	Region · E	Editor.Region.End_Of
By Tabs			
Tab forward	Control I		Editor.Char.Tab_Forward
Tab backward	Esc · Control I		Editor.Char.Tab_Backward
By Scrolling			
Scroll up	Image · ↑	Control Z	Editor.Image.Up
Scroll down	Image · ↓	Control V	Editor.Image.Down
Scroll right	Image · →		Editor.Image.Right
Scroll left	Image · ←		Editor.Image.Left
Scroll to top of image	Image · Begin Of	Image · B	Editor.Image.Beginning_Of
Scroll to end of image	Image · End Of	Image · E	Editor.Image.End_Of
Scroll current line to top	Window · Begin Of	Window · B	Editor.Window.Beginning_Of
Scroll current line to bottom	Window · End Of	Window · E	Editor.Window.End_Of
By Marking Your Place			
Set mark at cursor position	Mark · ↓	Control @	Editor.Mark.Push
Cycle through marks in stack	Mark · →	Esc · M	Editor.Mark.Next
Cycle back through marks in stack	Mark · ←		Editor.Mark.Previous
Return to most recent mark	Mark · ↑		Editor.Mark.Top

General Editing Operations

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Selecting an Arbitrary Region			
Select start of region	Region · [	Esc · [	Editor.Region.Start
Select end of region	Region ·]	Esc ·]	Editor.Region.Finish
Unselect a region	Region · X		Editor.Region.Off
Moving and Copying Text			
Copy a selected item	Region · C		Editor.Region.Copy
Move a selected item	Region · M		Editor.Region.Move
Duplicate a single line	Line · C	Esc · Control C	Editor.Line.Copy
Deleting Text			
Delete character — forward	Control D		Editor.Char.Delete_Forward
Delete character — backward	Delete		Editor.Char.Delete_Backward
Reduce multiple blanks to one	Control Delete		Editor.Char.Delete_Spaces
Delete word	Word · D	Esc · D	Editor.Word.Delete
Delete to end of word	Word · K	Esc · K	Editor.Word.Delete_Forward
Delete to beginning of word	Word · Delete	Esc · Delete	Editor.Word.Delete_Backward
Delete line	Line · D	Esc · Control D	Editor.Line.Delete
Delete to end of line	Line · K	Control K	Editor.Line.Delete_Forward
Delete to beginning of line	Line · Delete	Esc · Control F	Editor.Line.Delete_Backward
Delete selected item	Region · D, Region · K		Editor.Region.Delete
Searching and Replacing Text			
Search for next occurrence	Control F		Editor.Search.Next
Search for previous occurrence	Control R		Editor.Search.Previous
Replace next occurrence	Esc · F		Editor.Search.Replace_Next
Replace previous occurrence	Esc · R		Editor.Search.Replace_Previous

General Editing Operations *(Continued)*

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Entering Text			
Quote a special character	Esc · '		Editor.Char.Quote
Split line, cursor on new line	Line · I		Editor.Line.Insert
Split line, cursor on old line	Line · O	Control O	Editor.Line.Open
Join 2 lines	Line · J	Esc · O, Esc · Control O	Editor.Line.Join
Enter text in insert mode	Image · I		Editor.Set.Insert_Mode(True)
Enter text in overwrite mode	Image · O		Editor.Set.Insert_Mode(False)
Show current line number	Line · ?		What.Line
Transposing Text			
Transpose with previous character	Control T		Editor.Char.Transpose
Transpose with previous word	Word · T	Esc · T	Editor.Word.Transpose
Transpose with previous line	Line · T	Esc · Control T	Editor.Line.Transpose
Controlling Case			
Capitalize to end of word	Word · ^	Esc · ^	Editor.Word.Capitalize
Capitalize words to end of line	Line · ^		Editor.Line.Capitalize
Capitalize every word in region	Region · ^		Editor.Region.Capitalize
Make lowercase to end of word	Word · <	Esc · <	Editor.Word.Lower_Case
Make lowercase to end of line	Line · <		Editor.Line.Lower_Case
Convert entire region to lowercase	Region · <		Editor.Region.Lower_Case
Make uppercase to end of word	Word · >	Esc · >	Editor.Word.Upper_Case
Make uppercase to end of line	Line · >		Editor.Line.Upper_Case
Convert entire region to uppercase	Region · >		Editor.Region.Upper_Case

General Editing Operations *(Continued)*

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Holding and Retrieving Text			
Hold selected text	Region . ↓	Control C	Editor.Hold_Stack.Push
Retrieve most recently held text	Region . ↑	Control Y	Editor.Hold_Stack.Top
Retrieve previous held text	Region . ←		Editor.Hold_Stack.Previous
Retrieve next held text	Region . →	Esc . C, Esc . Y	Editor.Hold_Stack.Next
Formatting Text			
Center the line cursor is on	Line . $		Editor.Line.Center
Fill text in selected region	Region . Format		Editor.Region.Fill
Justify text in selected region	Region . Complete		Editor.Region.Justify
Automatically wrap lines	Image . F		Editor.Set.Fill_Mode(True)
Do not wrap lines	Image . X		Editor.Set.Fill_Mode(False)

Writing Text Files

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Accessing Text Files			
Create a new text file	Create Text		Text.Create
Display existing text file	Definition		Common.Definition
Open text file for editing	Edit		Common.Edit
Revert to last saved version	Object . L		Common.Revert
Saving Changes			
Save, leaving open for editing	Enter		Common.Commit
Save, making read only	Promote		Common.Promote
Terminating Edit			
Remove image, discarding changes	Object . G		Common.Abandon
Remove image, saving changes	Object . X		Common.Release
Selecting Substructures within Text			
Select current word	Object . ←		Common.Object.Parent
Select current sentence	numeric 2 . Object . ←		Common.Object.Parent
Select current paragraph	numeric 3 . Object . ←		Common.Object.Parent
Select smaller structure	Object . →		Common.Object.Child
Select previous structure, same level	Object . ↑		Common.Object.Previous
Select next structure, same level	Object . ↓		Common.Object.Next
Turn off selection	Control X		Editor.Set.Designation_Off

Writing Ada Programs

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Creating Ada Programs			
Create an Ada unit in library	Object . I		Common.Object.Insert
Build a body	Create Body		Ada.Create_Body
Build a private part	Create Private		Ada.Create_Private
Put temporary name in library	–		Ada.Install_Stub
Accessing Ada Programs			
Display Ada unit, read only	Definition		Common.Definition
Demote to source, open for editing	Edit		Common.Edit
Saving Changes and Terminating Edit			
Save, leaving open for editing	Enter		Common.Commit
Release image, discarding changes	Object . G		Common.Abandon
Release image, saving changes	Object . X		Common.Release
Revert to last version	Object . L		Common.Revert
Checking for Errors			
Complete and check syntax	Format		Common.Format
Check for semantic errors	Semanticize		Common.Semanticize
Explain underlined error	Object . ?		Common.Explain
Move to next underlined error	Esc . N		Editor.Cursor.Next
Move to previous underlined error	Esc . U		Editor.Cursor.Previous
Remove underline from error	Control X		Editor.Set.Designation_Off
Clear all underlined errors	Underlines Off		Common.Clear_Underlining
Redisplay cleared errors	Show Errors		Ada.Get_Errors

Writing Ada Programs *(Continued)*

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Changing the Compilation State			
Change unit to source state from any state	Source Unit		Ada.Source_Unit
Change unit to installed state from any state	Install Unit		Ada.Install_Unit
Change unit to coded state from any state	Code Unit		Ada.Code_Unit
Changing to a Higher Compilation State			
Promote unit to next higher state	Promote		Common.Promote
Code unit and those it depends on			
In this world only	Code (This World)		Compilation.Make
Across worlds	Code (All Worlds)		Compilation.Make
Install unit and those it depends on			
In this world only	Install (This World)		Compilation.Promote
Changing to a Lower Compilation State			
Demote unit to next lower state	Demote		Common.Demote
Demote unit and dependents to source			
In this world only	Source (This World)		Compilation.Demote
Selecting Structures within Ada Programs			
Select successively larger structures	Object . ←		Common.Object.Parent
Select successively smaller structures	Object . →		Common.Object.Child
Select previous structure, same level	Object . ↑		Common.Object.Previous
Select next structure, same level	Object . ↓		Common.Object.Next
Select first structure	Object . Begin Of	Object . B	Common.Object.First_Child
Select last structure	Object . End Of	Object . E	Common.Object.Last_Child
Turn off selection cursor is in	Control X		Editor.Set.Designation_Off

Writing Ada Programs *(Continued)*

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Modifying Ada Programs			
Edit selected Ada structure	Edit		Common.Edit
Insert Ada structures(s) in program	Object · I		Common.Object.Insert
Delete selected Ada structure	Object · D,		Common.Object.Delete
	Object · K		
Copy selected Ada structure	Object · C		Common.Object.Copy
Move selected Ada structure	Object · M		Common.Object.Move
Withdraw Ada unit stub	Withdraw Unit		Ada.Withdraw
Entering Comments and Special Strings			
Comment selected item or region	–		Region.Comment
Uncomment selected item or region	–		Region.Uncomment
Tab forward to comment	–		Editor.Char.Tab_To_Comment
Browsing Ada Programs			
Display other part of Ada unit	Other Part		Ada.Other_Part
Display other part, same window	Other Part In Place		Ada.Other_Part
Display Ada unit cursor is on	Definition		Common.Definition
Display parent object	Enclosing		Common.Enclosing
Set mark at current location	Mark · ↓		Editor.Mark.Push
Cycle through marks in stack	Mark · –	Esc · M	Editor.Mark.Next
Cycle back through marks in stack	Mark · ←		Editor.Mark.Previous
Return to most recent mark	Mark · ↑		Editor.Mark.Top
Checking Using Occurrences			
Show uses of selected identifier			
In this unit only	Show Usage (Unit)		Ada.Show_Usage
In any unit	Show Usage		Ada.Show_Usage
Show unused declarations			
In this unit only	Show Unused (Unit)		Ada.Show_Unused
Check other units	Show Unused		Ada.Show_Unused

Debugging Ada Programs

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Execute program with Debugger on	Esc · Promote		Command.Debug
Display Debugger window	Debugger Window		Debug.Current_Debugger
Show current statement in source	Show Source		Debug.Source
Stepping and Executing			
Continue program execution	Execute		Debug.Execute
Step one statement	Run		Debug.Run
Step one statement at same level	Run Local		Debug.Run (Local)
Stop task execution	Stop		Debug.Stop
Display information about tasks	Task Display		Debug.Task_Display
Display task rendezvous info	–		Debug.Information
Setting and Removing Breakpoints			
Set breakpoints with default lifetime	Break		Debug.Break
Display breakpoints	Show Breaks		Debug.Show
Reactivate existing breakpoints	Activate		Debug.Activate
Remove breakpoints	Remove Breaks		Debug.Remove
Viewing Stacks			
Display calling stack	Stack		Debug.Stack
Displaying and Modifying Variables			
Display values of selected variables	Put		Debug.Put
Modify value of selected variable	Modify		Debug.Modify
Handling Exceptions			
Stop execution when exception raised	Catch		Debug.Catch
Do not stop when exception raised	Propagate		Debug.Propagate
Remove handling for this exception	–		Debug.Forget

Managing Libraries

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Creating Libraries			
Create a directory	Create Directory		Library.Create_Directory
Create a world	Create World		Library.Create_World
Manipulating Objects in Libraries			
Create an Ada unit in library	Object . I		Common.Object.Insert
Create a text file in library	Create Text		Text.Create
Delete selected object from library	Object . D, Object . K		Common.Object.Delete
Undelete selected object from library	Object . U		Common.Object.Undo
Print selected object	Print		Queue.Print
Show access list for designated object	Show Access List		Access_List.Display
Controlling Library Display			
Toggle information on library objects	Object . ?		Common.Explain
Show more detail	Object . !		Common.Expand
Show less detail	Object . -		Common.Elide

Using CMVC

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Check out designated object	–		Cmvc.Check_Out
Check in designated object	–		Cmvc.Check_In
Accept changes for designated object	–		Cmvc.Accept_Changes
Show objects that are checked out			
In this view	–		Cmvc.Show_Checked_Out_In_View
By you, any view	–		Cmvc.Show_Checked_Out_By_User
Show info about designated object	–		Cmvc.Show
Show out-of-date objects in this view	–		Cmvc.Show_Out_Of_Date_Objects

Managing Links

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Accessing Links			
List links	–		Links.Display
Edit links display	–		Links.Edit
Refresh link image	Object . L		Common.Revert
Removing the Link Editor			
Remove window temporarily	Window . D		Editor.Window.Delete
Release image permanently	Object . X		Common.Release
Selecting Links			
Select link cursor is on	Object . →		Common.Object.Child
Select all links	Object . ←		Common.Object.Parent
Select previous link	Object . ↑		Common.Object.Previous
Select next link	Object . ↓		Common.Object.Next
Select first link in image	Object . Begin Of	Object . B	Common.Object.First_Child
Select last link in image	Object . End Of	Object . E	Common.Object.Last_Child
Modifying Links			
Add a new link—simple method	–		Links.Add
Add a new link	Object . I		Common.Object.Insert
Give selected link another source	Edit		Common.Edit
Delete selected link	Object . D, Object . K		Common.Object.Delete
Traversing Linked Ada Units			
Go to source unit of current link	Definition		Common.Definition
Go to world associated with links	Enclosing		Common.Enclosing
List Ada units that use current link	Object . ?		Common.Explain
Controlling the Display			
Toggle order of kind of link	Object . !		Common.Expand
Toggle classes of source of link	Object . ·		Common.Elide

Managing Searchlists

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Accessing the Searchlist			
Edit or view searchlist	–		Search_List.Edit
Refresh searchlist image	Object . L		Common.Revert
Removing the Searchlist Editor			
Remove window temporarily	Window . D		Editor.Window.Delete
Release image permanently	Object . X		Common.Release
Selecting Entries			
Select entry cursor is on	Object . →		Common.Object.Child
Select all entries	Object . ←		Common.Object.Parent
Select next entry	Object . ↓		Common.Object.Next
Select previous entry	Object . ↑		Common.Object.Previous
Select first entry on list	Object . Begin Of	Object . B	Common.Object.First_Child
Select last entry on list	Object . End Of	Object . E	Common.Object.Last_Child
Go to world named by current entry	Definition		Common.Definition
Modifying the Searchlist			
Add a new entry	Object . I		Common.Object.Insert
Delete selected entry	Object . D		Common.Object.Delete
	Object . K		
Move selected entry	Object . M		Common.Object.Move

Using Keyboard Macros

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Start macro definition	Mark · Begin Of	Mark · [	Editor.Macro.Start
End macro definition	Mark · End Of	Mark ·]	Editor.Macro.Finish
Execute macro	Mark · Promote	Esc · X	Editor.Macro.Execute
Bind macro to key	Mark · Definition		Editor.Macro.Bind

Using Environment I/O Resources

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Indicate end of input to program	numeric .		Text.End_Of_Input
Commit interactive input	Promote	Enter	Common.Commit

Managing Jobs

Description	*Basic Keys*	*Accelerated Keys*	*Command*
Disconnect job from terminal	Control G		Job.Interrupt
Kill job	Job Kill	Esc · G	Job.Kill(0)
Stop running jobs	Job Disable		Job.Disable(0)
Resume stopped jobs	Job Enable		Job.Enable(0)
Reconnect job	Job Connect		Job.Connect(0)

Master Reference to Key Bindings by Command

Legend

C =	Control	X1 =	Object
ESC =	Esc	X2 =	Region
ESC_C =	Esc - Control	X3 =	Window
S =	Shift	PF1 =	Image
X4 =	Promote	PF2 =	Line
X5 =	Complete	PF3 =	Word
X6 =	Format	PF4 =	Mark

Command	Facit	Rational
Access_List.Display	ESC_C_F5	
Access_List.Display (For_Object...		CMS_F20
Ada.Code_Unit	S_F6	C_F13
Ada.Create_Body	S_F8	S_F15
Ada.Create_Private	ESC_C_F9	CMS_F15
Ada.Delete_Blank_Line		CMS_K
		CM_K
Ada.Get_Errors	S_F10	S_F16
Ada.Insert_Blank_Line		CMS_I
		CM_I
Ada.Install_Stub		S_F13
Ada.Install_Unit	F6	F13
Ada.Make_Inline		CMS_F17
Ada.Make_Separate		CMS_F18
Ada.Other_Part (Name => "<Image...	F9	C_F10
Ada.Other_Part (Name => "<Image...	S_F9	CS_F10
Ada.Show_Unused (In_Unit => "<I...	C_F10	M_F17
Ada.Show_Unused (In_Unit => "<I...	ESC_C_F10	CM_F17
Ada.Show_Usage (Name => "<Curso...	ESC_F9	M_F16
Ada.Show_Usage (Name => "<Curso...	C_F9	CM_F16
Ada.Show_Usage (Name => "<Curso...		CMS_F16
Ada.Source_Unit	C_F7	C_F14
Ada.Withdraw	ESC_C_F7	M_F14
Cmvc.Accept_Changes (Destinatio...		CM_F12
Cmvc.Check_In (What_Object => "...		M_F12
Cmvc.Check_Out (What_Object => ...		C_F12
Cmvc.Show (Objects => "<Cursor>...		CS_F12
Cmvc.Show_Checked_Out_By_User (...		MS_F12
Cmvc.Show_Checked_Out_In_View (...		CMS_F12
Cmvc.Show_Out_Of_Date_Objects (...		S_F12
Command.Debug	ESC_C_M	M_PROMOT
		MS_CARRIAGE_RETURN
		M_CARRIAGE_RETURN
		MS_ENTER
		M_ENTER
Command.Spawn	S_X4	C_PROMOT
Common.Abandon	X1.'g'	OBJECT.'G'
	X1.'G'	OBJECT.'g'
Common.Clear_Underlining	ESC_F10	C_F16
Common.Commit	ENTER	S_ENTER
		C_CARRIAGE_RETURN
		ENTER
		CS_CARRIAGE_RETURN
Common.Complete	X5	COMPLT
Common.Create_Command	F8	F15
Common.Definition	X1.F4	OBJECT.F10
		CM_RIGHT
Common.Definition (In_Place => ...		S_RIGHT
Common.Definition (Name => "<Cu...	F4	F10
Common.Definition (Name => "<Cu...	S_F4	
Common.Definition (Name => "<Cu...		S_F10
Common.Demote	S_F7	S_F14
Common.Edit	F7	F14
Common.Elide	X1.'.'	OBJECT.'.'
		OBJECT.'>'
Common.Enclosing	C_F4	M_F10
		CM_LEFT
Common.Enclosing (In_Place => F...	ESC_F4	
Common.Enclosing (In_Place => T...		S_LEFT
Common.Enclosing (In_Place => T...	ESC_C_F4	MS_F10
Common.Enclosing (In_Place => T...		CMS_F10
Common.Expand	X1.'1'	C_EXCLAM
	X1.'!'	OBJECT.'!'
		C_1
		OBJECT.'1'
Common.Explain	ESC_SLASH	OBJECT.'?'
	ESC_PLUS	F17
	X1.'/'	C_QUERY
	X1.'+'	OBJECT.'/'
	ESC_QUERY	C_SLASH
	X1.'?'	
Common.Format	X6	FORMAT
Common.Object.Child	X1.RIGHT	OBJECT.RIGHT
		C_RIGHT
Common.Object.Copy	X1.'c'	OBJECT.'c'
	X1.'C'	OBJECT.'C'
Common.Object.Delete	X1.'K'	OBJECT.'d'
	X1.'D'	OBJECT.'D'
	X1.'k'	OBJECT.'k'
	X1.'d'	OBJECT.'K'
Common.Object.First_Child	X1.'B'	OBJECT.BEGIN_OF
	X1.'b'	C_BEGIN_OF
	X1.BEGIN_OF	
Common.Object.Insert	X1.'i'	OBJECT.'i'
	X1.'I'	C_F15
		OBJECT.'I'
Common.Object.Last_Child	X1.'e'	C_END_OF
	X1.END_OF	OBJECT.END_OF
	X1.'E'	
Common.Object.Move	X1.'M'	OBJECT.'M'
	X1.'m'	OBJECT.'m'
Common.Object.Next	X1.DOWN	C_DOWN
		OBJECT.DOWN
Common.Object.Parent	X1.LEFT	OBJECT.LEFT
		C_LEFT
Common.Object.Previous	X1.UP	OBJECT.UP
		C_UP
Common.Promote	X4	PROMOT

Common.Redo	X1.'R'	OBJECT.'r'
	X1.'r'	OBJECT.'R'
Common.Release	X1.'X'	OBJECT.'x'
	X1.'x'	OBJECT.'X'
Common.Revert	X1.'L'	OBJECT.'l'
	X1.'l'	OBJECT.'L'
Common.Semanticize	F10	F16
Common.Sort_Image		OBJECT.'S'
		OBJECT.'s'
Common.Undo	X1.'u'	OBJECT.'U'
	X1.'U'	OBJECT.'u'
Compilation.Demote (Unit => "<S...		MS_F14
Compilation.Demote (Unit => "<S...		CS_F14
Compilation.Demote (Unit => "<S...		CMS_F14
Compilation.Demote (Unit => "<S...	ESC_F7	CM_F14
Compilation.Make (Unit => "<Ima...	C_F6	CM_F13
Compilation.Make (Unit => "<Ima...	ESC_C_F6	CMS_F13
Compilation.Promote (Unit => "<...		MS_F13
Compilation.Promote (Unit => "<...	ESC_F6	M_F13
Debug.Activate (Breakpoint => 0)	ESC_F2	M_F7
Debug.Break	S_F2	C_F7
Debug.Break (Default_Lifetime =...		S_F7
Debug.Catch	S_F3	C_F8
Debug.Current_Debugger ("")		CMS_F9
Debug.Execute	S_F1	S_F6
Debug.Forget		S_F8
Debug.Information (Debug.Rendez...		CMS_F8
Debug.Modify (New_Value => "", ...	ESC_C_F3	CM_F9
Debug.Propagate	ESC_F3	M_F8
Debug.Put	F3	F9
Debug.Remove (Breakpoint => 0)	C_F2	CM_F7
Debug.Run	F1	F6
Debug.Run (Debug.Returned)		M_F6
Debug.Run (Stop_At => Debug.Loc...	ESC_F1	C_F6
Debug.Set_Value (Variable => De...		C_F9
Debug.Set_Value (Variable => De...		S_F9
Debug.Set_Value (Variable => De...		M_F9
Debug.Show	ESC_C_F2	CMS_F7
Debug.Show (Debug.Exceptions)		CM_F8
Debug.Source	F2	
Debug.Source (Location => "", S...		F7
Debug.Stack	C_F3	F8
Debug.Stop	C_F1	
Debug.Stop (Name => "")		CM_F6
Debug.Task_Display	ESC_C_F1	CMS_F6
Editor.Char.Capitalize		C_6
		C_CIRCUMFLEX
Editor.Char.Delete_Backward	DELETE	DELETE
Editor.Char.Delete_Forward	C_D	C_D
		CS_D
Editor.Char.Delete_Spaces	ESC_BACKSLASH	C_DELETE
		CS_DELETE
Editor.Char.Insert_Character (1...		MS_SPACE
		CS_SPACE
		CM_SPACE
		M_SPACE
		CMS_SPACE
		C_SPACE
		S_SPACE
Editor.Char.Insert_String ("")")		C_RIGHT_PAREN
		C_0
Editor.Char.Insert_String ("("")		C_9
		C_LEFT_PAREN
Editor.Char.Insert_String (":=")		C_COLON
		C_SEMICOLON
Editor.Char.Insert_String ("=>")		C_EQUAL
		C_PLUS
Editor.Char.Lower_Case		C_LESS_THAN
		C_COMMA
Editor.Char.Quote	ESC_TICK	C_TICK
	ESC_STAR	C_QUOTE
		CS_TICK
Editor.Char.Tab_Backward	ESC_C_I	CS_TAB
		C_TAB
Editor.Char.Tab_Forward	C_I	S_TAB
		TAB
Editor.Char.Tab_To_Comment		MS_TAB
		OBJECT.TAB
		M_TAB
Editor.Char.Transpose	C_T	C_T
		CS_T
Editor.Char.Upper_Case		C_PERIOD
		C_GREATER_THAN
Editor.Cursor.Backward		CS_B
		C_B
Editor.Cursor.Down	C_N	C_N
	DOWN	LINE.DOWN
		CS_N
		DOWN
Editor.Cursor.Down (8)	ESC_C_N	CM_N
		CMS_N
Editor.Cursor.Forward		C_F
		CS_F
Editor.Cursor.Left	C_H	CS_H
	LEFT	C_H
		LEFT
Editor.Cursor.Left (8)	ESC_C_H	CM_H
		CMS_H
Editor.Cursor.Next	ESC_S_N	M_DOWN
	ESC_N	
Editor.Cursor.Next (Prompt => F...		S_F18
Editor.Cursor.Next (Prompt => T...		M_F18
Editor.Cursor.Next (Prompt => T...		F18
		M_N
		MS_N
Editor.Cursor.Previous	ESC_U	M_U
	ESC_S_U	MS_U
		M_UP
Editor.Cursor.Previous (Prompt ...		CS_F18
Editor.Cursor.Previous (Prompt ...		CM_F18
Editor.Cursor.Previous (Prompt ...		C_F18
Editor.Cursor.Right	C_J	C_J
	RIGHT	CS_J
		RIGHT
Editor.Cursor.Right (8)	ESC_C_J	CMS_F
		CM_F
		CMS_J
		CM_J
Editor.Cursor.Up	C_U	C_U
	UP	CS_U

		UP
		LINE.UP
Editor.Cursor.Up (8)	ESC_C_U	CMS_U
		CM_U
Editor.Hold_Stack.Copy_Top		REGION.'p'
		REGION.'P'
Editor.Hold_Stack.Delete_Top		REGION.DELETE
Editor.Hold_Stack.Next	ESC_Y	M_Y
	ESC_C	MS_Y
	X2.RIGHT	M_C
	ESC_S_Y	REGION.RIGHT
	ESC_S_C	MS_C
Editor.Hold_Stack.Previous	X2.LEFT	REGION.LEFT
Editor.Hold_Stack.Push	C_C	C_C
	X2.DOWN	REGION.DOWN
		CS_C
Editor.Hold_Stack.Rotate		REGION.'r'
		REGION.'R'
Editor.Hold_Stack.Swap		REGION.'T'
		REGION.'t'
Editor.Hold_Stack.Top	C_Y	REGION.UP
	X2.UP	CS_Y
		C_Y
Editor.Image.Beginning_Of	PF1.'b'	S_BEGIN_OF
	PF1.BEGIN_OF	IMAGE.BEGIN_OF
	PF1.'B'	
Editor.Image.Down	C_V	C_V
	PF1.DOWN	S_DOWN
		IMAGE.DOWN
		CS_V
Editor.Image.End_Of	PF1.END_OF	IMAGE.END_OF
	PF1.'E'	S_END_OF
	PF1.'e'	
	PF1.TAB	
Editor.Image.Find ("")		IMAGE.'?'
		IMAGE.'/'
Editor.Image.Find ("Help Window")	ESC_F5	C_F11
Editor.Image.Find (Name => "Nam...	PF1.'+'	
	PF1.'/'	
	PF1.'?'	
Editor.Image.Left	PF1.LEFT	MS_LEFT
		IMAGE.LEFT
Editor.Image.Right	PF1.RIGHT	MS_RIGHT
		IMAGE.RIGHT
Editor.Image.Up	PF1.UP	IMAGE.UP
	C_Z	CS_Z
		S_UP
		C_Z
Editor.Key.Name	C_F5	C_Q
	ESC_AT_SIGN	CS_Q
	ESC_QUOTATION	M_F11
	ESC_2	
Editor.Key.Prompt	ESC_S_Q	M_Q
	ESC_Q	MS_Q
Editor.Key.Prompt (Key_Code => "")		F12
Editor.Line.Beginning_Of	ESC_C_B	CMS_B
	PF2.'b'	CM_B
	C_B	CM_A
	C_A	CS_A
	ESC_C_A	C_A

	PF2.'B'	LINE.BEGIN_OF
	PF2.BEGIN_OF	CMS_A
		BEGIN_OF
Editor.Line.Capitalize	PF2.'^'	CM_CIRCUMFLEX
	PF2.'6'	CM_6
	PF2.'~'	LINE.'~'
		LINE.'6'
Editor.Line.Center	PF2.'4'	LINE.'$'
	PF2.'$'	LINE.'4'
Editor.Line.Copy	ESC_C_C	LINE.'C'
	PF2.'c'	CM_C
	PF2.'C'	LINE.'c'
		CMS_C
Editor.Line.Delete	ESC_C_D	CM_D
	PF2.'D'	CMS_D
	PF2.'d'	LINE.'D'
		LINE.'d'
Editor.Line.Delete_Backward	ESC_C_F	LINE.DELETE
	PF2.DELETE	CM_DELETE
		CMS_DELETE
Editor.Line.Delete_Forward	PF2.'k'	CS_K
	C_K	LINE.'K'
	ESC_C_K	LINE.'k'
	PF2.'K'	C_K
Editor.Line.End_Of	PF2.END_OF	CM_E
	PF2.TAB	CS_E
	PF2.'e'	CMS_E
	PF2.'E'	END_OF
	C_E	C_E
	ESC_C_E	LINE.END_OF
Editor.Line.Indent	C_M	CARRIAGE_RETURN
Editor.Line.Insert	PF2.'i'	S_CARRIAGE_RETURN
	PF2.'I'	LINE.'i'
		LINE.'I'
Editor.Line.Insert (- (1))		C_I
		CS_I
Editor.Line.Join	PF2.'J'	LINE.'J'
	ESC_C_O	M_O
	ESC_S_O	CM_O
	ESC_O	MS_O
	PF2.'j'	CMS_O
		LINE.'j'
Editor.Line.Lower_Case	PF2.'<'	LINE.','
		LINE.'<'
		CM_LESS_THAN
		CM_COMMA
Editor.Line.Open	PF2.'o'	C_O
	C_O	CS_O
	PF2.'O'	LINE.'o'
		LINE.'O'
Editor.Line.Transpose	ESC_C_T	CM_T
	PF2.'T'	LINE.'t'
	PF2.'t'	LINE.'T'
		CMS_T
Editor.Line.Upper_Case	PF2.'>'	CM_GREATER_THAN
		LINE.'.'
		CM_PERIOD
		LINE.'>'
Editor.Macro.Bind	PF4.F4	MARK.F10
Editor.Macro.Execute	PF4.ENTER	M_X

	PF4.X4	MARK.ENTER
	ESC_S_X	MARK.PROMOT
	ESC_X	MARK.CARRIAGE_RETURN
		MS_X
Editor.Macro.Finish	PF4.'e'	MARK.']'
	PF4.'E'	M_RIGHT_BRACE
	PF4.'}'	MARK.'}'
	PF4.END_OF	MARK.END_OF
	PF4.']'	M_RIGHT_BRACKET
Editor.Macro.Start	PF4.BEGIN_OF	M_LEFT_BRACKET
	PF4.'b'	MARK.'['
	PF4.'{'	M_LEFT_BRACE
	PF4.'B'	MARK.'{'
	PF4.'['	MARK.BEGIN_OF
Editor.Mark.Copy_Top		MARK.'p'
		MARK.'P'
Editor.Mark.Delete_Top		MARK.DELETE
Editor.Mark.Next	ESC_M	MS_M
	PF4.RIGHT	M_M
	ESC_S_M	MARK.RIGHT
Editor.Mark.Previous	PF4.LEFT	MARK.LEFT
Editor.Mark.Push	NUL	MARK.DOWN
	PF4.DOWN	CS_M
		C_M
Editor.Mark.Rotate		MARK.'R'
		MARK.'r'
Editor.Mark.Swap		MARK.'t'
		MARK.'T'
Editor.Mark.Top	PF4.UP	MARK.UP
Editor.Region.Beginning_Of	X2.'b'	REGION.BEGIN_OF
	X2.'B'	
	X2.BEGIN_OF	
Editor.Region.Capitalize	X2.'6'	REGION.'6'
	X2.'~'	REGION.'~'
	X2.'^'	
Editor.Region.Comment		REGION.'_'
		REGION.'-'
Editor.Region.Copy	X2.'c'	REGION.'c'
	X2.'C'	REGION.'C'
Editor.Region.Delete	X2.'D'	REGION.'K'
	X2.'d'	REGION.'k'
	X2.'K'	REGION.'d'
	X2.'k'	REGION.'D'
Editor.Region.End_Of	X2.'E'	REGION.END_OF
	X2.END_OF	
	X2.'e'	
Editor.Region.Fill	X2.X6	REGION.FORMAT
Editor.Region.Finish	X2.'}'	REGION.']'
	X2.']'	C_RIGHT_BRACKET
	ESC_RIGHT_BRACE	C_RIGHT_BRACE
		REGION.'}'
Editor.Region.Justify	X2.X5	REGION.COMPLT
Editor.Region.Lower_Case	X2.'<'	REGION.'<'
		REGION.','
Editor.Region.Move	X2.'m'	REGION.'M'
	X2.'M'	REGION.'m'
Editor.Region.Off	X2.'x'	REGION.'x'
	X2.'X'	REGION.'X'
Editor.Region.Start	ESC_LEFT_BRACE	C_LEFT_BRACKET
	X2.'{'	REGION.'['

	X2.'['	REGION.'{'
		C_LEFT_BRACE
Editor.Region.Uncomment		REGION.'='
		REGION.'+'
Editor.Region.Upper_Case	X2.'>'	REGION.'.'
		REGION.'>'
Editor.Screen.Clear	ESC_L	M_L
	ESC_S_L	MS_L
Editor.Screen.Down		CS_DOWN
Editor.Screen.Left		CS_LEFT
Editor.Screen.Next		CMS_RIGHT
Editor.Screen.Previous		CMS_LEFT
Editor.Screen.Push		CMS_DOWN
Editor.Screen.Redraw	C_L	CS_L
		C_L
Editor.Screen.Right		CS_RIGHT
Editor.Screen.Top		CMS_UP
		CMS_BEGIN_OF
Editor.Screen.Up		CS_UP
Editor.Search.Next	C_F	C_S
		CS_S
Editor.Search.Previous	C_R	CS_R
		C_R
Editor.Search.Replace_Next	ESC_S_F	M_S
	ESC_F	MS_S
Editor.Search.Replace_Previous	ESC_R	M_R
	ESC_S_R	MS_R
Editor.Set.Argument_Digit (0)	NUMERIC_0	M_NUMERIC_0
		C_NUMERIC_0
		NUMERIC_0
		S_NUMERIC_0
Editor.Set.Argument_Digit (1)	NUMERIC_1	M_NUMERIC_1
		C_NUMERIC_1
		NUMERIC_1
		S_NUMERIC_1
Editor.Set.Argument_Digit (2)	NUMERIC_2	NUMERIC_2
		M_NUMERIC_2
		S_NUMERIC_2
		C_NUMERIC_2
Editor.Set.Argument_Digit (3)	NUMERIC_3	C_NUMERIC_3
		M_NUMERIC_3
		S_NUMERIC_3
		NUMERIC_3
Editor.Set.Argument_Digit (4)	NUMERIC_4	NUMERIC_4
		C_NUMERIC_4
		M_NUMERIC_4
		S_NUMERIC_4
Editor.Set.Argument_Digit (5)	NUMERIC_5	C_NUMERIC_5
		M_NUMERIC_5
		S_NUMERIC_5
		NUMERIC_5
Editor.Set.Argument_Digit (6)	NUMERIC_6	C_NUMERIC_6
		S_NUMERIC_6
		M_NUMERIC_6
		NUMERIC_6
Editor.Set.Argument_Digit (7)	NUMERIC_7	C_NUMERIC_7
		NUMERIC_7
		M_NUMERIC_7
		S_NUMERIC_7
Editor.Set.Argument_Digit (8)	NUMERIC_8	C_NUMERIC_8

		S_NUMERIC_8
		M_NUMERIC_8
		NUMERIC_8
Editor.Set.Argument_Digit (9)	NUMERIC_9	M_NUMERIC_9
		S_NUMERIC_9
		NUMERIC_9
		C_NUMERIC_9
Editor.Set.Argument_Minus	DASH	C_DASH
		DASH
		S_DASH
		M_DASH
Editor.Set.Argument_Prefix	NUMERIC_COMMA	NUMERIC_COMMA
		M_NUMERIC_COMMA
		C_NUMERIC_COMMA
		S_NUMERIC_COMMA
Editor.Set.Designation_Off	C_X	CS_X
		C_F17
		C_X
Editor.Set.Fill_Mode (False)	PF1.'X'	IMAGE.'X'
	PF1.'x'	IMAGE.'x'
Editor.Set.Fill_Mode (True)	PF1.'f'	IMAGE.'f'
	PF1.'F'	IMAGE.'F'
Editor.Set.Insert_Mode (False)	PF1.'O'	IMAGE.'o'
	PF1.'o'	IMAGE.'O'
Editor.Set.Insert_Mode (True)	PF1.'i'	IMAGE.'I'
	PF1.'I'	IMAGE.'i'
Editor.Window.Beginning_Of	X3.'b'	WINDOW.BEGIN_OF
	X3.BEGIN_OF	CM_BEGIN_OF
	X3.'B'	
Editor.Window.Child	X3.RIGHT	WINDOW.RIGHT
Editor.Window.Copy	X3.'c'	WINDOW.'c'
	X3.'C'	WINDOW.'C'
Editor.Window.Delete	X3.'D'	WINDOW.'x'
	X3.'X'	WINDOW.'X'
	X3.'K'	WINDOW.'k'
	X3.'d'	WINDOW.'K'
	X3.'k'	WINDOW.'d'
	X3.'x'	WINDOW.'D'
Editor.Window.Demote	X3.F7	WINDOW.S_F14
	X3.S_F7	WINDOW.F14
Editor.Window.Directory	X3.'/'	WINDOW.F10
	X3.F4	WINDOW.'?'
	X3.'+'	WINDOW.'/'
	X3.'?'	
Editor.Window.End_Of	X3.'e'	CM_END_OF
	X3.END_OF	WINDOW.END_OF
	X3.'E'	
Editor.Window.Expand	X3.'!'	WINDOW.'!'
	X3.'1'	WINDOW.'1'
Editor.Window.Expand (- (4))	X3.'.'	WINDOW.'>'
		WINDOW.'.'
Editor.Window.Focus	X3.X6	WINDOW.FORMAT
Editor.Window.Join (- (1))	X3.DELETE	WINDOW.DELETE
Editor.Window.Join (1)	X3.'J'	WINDOW.'j'
	X3.'j'	WINDOW.'J'
Editor.Window.Next	ESC_V	CM_DOWN
	S_DOWN	WINDOW.DOWN
	X3.DOWN	MS_V
	ESC_S_V	M_V
Editor.Window.Parent	X3.LEFT	WINDOW.LEFT

Editor.Window.Previous	ESC_S_Z	WINDOW.UP
	ESC_Z	M_Z
	S_UP	MS_Z
	X3.UP	CM_UP
Editor.Window.Promote	X3.X4	WINDOW.PROMOT
Editor.Window.Transpose	X3.'t'	WINDOW.'T'
	X3.'T'	WINDOW.'t'
Editor.Word.Beginning_Of	PF3.BEGIN_OF	M_A
	ESC_A	WORD.BEGIN_OF
	PF3.'B'	MS_A
	PF3.'b'	M_BEGIN_OF
	ESC_S_B	MS_B
	ESC_S_A	M_B
	ESC_B	
Editor.Word.Capitalize	PF3.'6'	WORD.'6'
	PF3.'^'	M_6
	PF3.'~'	M_CIRCUMFLEX
	ESC_TILDE	WORD.'^'
	ESC_6	
	ESC_CIRCUMFLEX	
Editor.Word.Delete	PF3.'D'	MS_D
	ESC_S_D	M_D
	PF3.'d'	WORD.'d'
	ESC_D	WORD.'D'
Editor.Word.Delete_Backward	PF3.DELETE	M_DELETE
	ESC_DEL	WORD.DELETE
		MS_DELETE
Editor.Word.Delete_Forward	ESC_K	MS_K
	PF3.'k'	WORD.'k'
	ESC_S_K	M_K
	PF3.'K'	WORD.'K'
Editor.Word.End_Of	PF3.TAB	M_E
	ESC_S_E	WORD.END_OF
	PF3.'e'	MS_E
	ESC_E	M_END_OF
	PF3.'E'	
	PF3.END_OF	
Editor.Word.Lower_Case	PF3.'<'	M_LESS_THAN
	ESC_LESS_THAN	WORD.'<'
		M_COMMA
		WORD.','
Editor.Word.Next	PF3.RIGHT	M_J
	ESC_S_J	M_RIGHT
	ESC_J	WORD.RIGHT
		MS_J
Editor.Word.Previous	PF3.LEFT	M_LEFT
	ESC_S_H	WORD.LEFT
	ESC_H	M_H
		MS_H
Editor.Word.Transpose	ESC_S_T	WORD.'t'
	PF3.'t'	WORD.'T'
	PF3.'T'	MS_T
	ESC_T	M_T
Editor.Word.Upper_Case	PF3.'>'	WORD.'.'
	ESC_GREATER_THAN	WORD.'>'
		M_GREATER_THAN
		M_PERIOD
Job.Connect (0)	ESC_F11	M_F19
Job.Disable (0)	ESC_C_F11	F19
Job.Enable (0)	S_F11	S_F19

Job.Interrupt	C_G	C_G
	C_G	CS_G
Job.Kill (0)	ESC_G	C_F19
	C_F11	M_G
	ESC_S_G	MS_G
	ESC_G	
Library.Create_Directory (Name ...	ESC_F8	M_F15
Library.Create_World (Name => "")	ESC_C_F8	CS_F15
Queue.Print	F11	CM_F11
Text.Create (Image_Name => "")	C_F8	CM_F15
Text.End_Of_Input	DOT	DOT
		CM_F19
What.Does ("")	S_F5	F11
What.Does ("Help_On_Help")	F5	S_F11
What.Home_Library	ESC_UP	CM_F10
What.Line	PF2.'?'	LINE.'/'
	PF2.'+'	LINE.'?'
	PF2.'/'	
What.Load (Verbose => True)	S_F12	S_F20
What.Locks (Name => "<Image>")	ESC_C_F12	M_F20
What.Object	C_F12	CM_F20
What.Tabs		CMS_TAB
		CM_TAB
What.Time	F12	F20

Master Reference to Key Bindings by Key

```
with Access_List;
with Ada;
with Command;
with Common;
with Compilation;
with Daemon;
with Debug;
with Editor;
with File_Utilities;
with Job;
with Library;
with Operator;
with Profile;
with Queue;
with Text;
with Visible_Key_Names;
with What;

procedure Facit_Commands is

    -- Updated for Delta under A_9_5_0 by KR, 3/28/87
    -- Updated for Delta under A_9_16_0 for LLB, 6/24/87

    use Visible_Key_Names;

    type Intent is (Prompt, Execute, Interrupt);

    Action : Intent;

    Key_1 : Facit_Key_Names;
    Key_2 : Facit_Key_Names;


    -- Interrupt and Prompt are first because they are shorter and can be moved
    -- around very easily.  Each of the keys is replicated in comments in the
    -- main Execute case statement.

begin
    case Action is
        when Interrupt =>
            case Key_1 is

                when C_G =>
                    Job.Interrupt;

                when Esc_G =>
                    Job.Kill (0);

                when C_F1 =>
                    Debug.Stop;
                when Esc_C_F11 =>
                    Job.Disable (0);

                when C_F11 =>
                    Job.Kill (0);

                when others =>
                    null;
            end case;

        when Prompt =>

            case Key_1 is
                when Esc_C_F3 =>
                    Debug.Modify (New_Value => "",
                                  Variable => "<SELECTION>",
                                  Stack_Frame => 0);
                when S_F5 =>
                    What.Does ("");

                when Esc_F8 =>
                    Library.Create_Directory (Name => "");

                when C_F8 =>
                    Text.Create (Image_Name => "");
                when Esc_C_F8 =>
                    Library.Create_World (Name => "");

                when Pf1 =>
                    case Key_2 is
                        when '?' | '/' | '+' =>
                            Editor.Image.Find (Name => "name or name fragment");
                        when others =>
                            null;
                    end case;

                when others =>
                    null;
            end case;

        when Execute =>
            case Key_1 is

                when X4 =>
                    Common.Promote;
                when S_X4 =>
                    Command.Spawn;
                when X5 =>
                    Common.Complete;
                when X6 =>
                    Common.Format;

                when C_A =>
                    Editor.Line.Beginning_Of;
                when C_B =>
                    Editor.Line.Beginning_Of;
                when C_C =>
                    Editor.Hold_Stack.Push;
                when C_D =>
                    Editor.Char.Delete_Forward;
                when C_E =>
                    Editor.Line.End_Of;
                when C_F =>
                    Editor.Search.Next;
                when C_G =>
                    Job.Interrupt;
                when C_H =>
                    Editor.Cursor.Left;
                when C_I =>
                    Editor.Char.Tab_Forward;
```

```
when C_J =>
    Editor.Cursor.Right;
when C_K =>
    Editor.Line.Delete_Forward;
when C_L =>
    Editor.Screen.Redraw;
when C_M =>
    Editor.Line.Indent;
when C_N =>
    Editor.Cursor.Down;
when C_O =>
    Editor.Line.Open;
when C_R =>
    Editor.Search.Previous;
when C_T =>
    Editor.Char.Transpose;
when C_U =>
    Editor.Cursor.Up;
when C_V =>
    Editor.Image.Down;
when C_X =>
    Editor.Set.Designation_Off;
when C_Y =>
    Editor.Hold_Stack.Top;
when C_Z =>
    Editor.Image.Up;
when Nul =>
    Editor.Mark.Push;

when Esc_Left_Brace =>
    Editor.Region.Start;
when Esc_Right_Brace =>
    Editor.Region.Finish;

when Esc_Query | Esc_Slash | Esc_Plus =>
    Common.Explain;
when Esc_Tick | Esc_Star =>
    Editor.Char.Quote;

when Esc_A | Esc_S_A | Esc_B | Esc_S_B =>
    Editor.Word.Beginning_Of;
when Esc_C | Esc_S_C =>
    Editor.Hold_Stack.Next;
when Esc_D | Esc_S_D =>
    Editor.Word.Delete;
when Esc_E | Esc_S_E =>
    Editor.Word.End_Of;
when Esc_F | Esc_S_F =>
    Editor.Search.Replace_Next;
when Esc_G | Esc_S_G =>
    Job.Kill (0);
when Esc_H | Esc_S_H =>
    Editor.Word.Previous;
when Esc_J | Esc_S_J =>
    Editor.Word.Next;
when Esc_K | Esc_S_K =>
    Editor.Word.Delete_Forward;
when Esc_L | Esc_S_L =>
    Editor.Screen.Clear;
when Esc_M | Esc_S_M =>
    Editor.Mark.Next;
when Esc_N | Esc_S_N =>
    Editor.Cursor.Next;
when Esc_O | Esc_S_O =>
    Editor.Line.Join;
when Esc_Q | Esc_S_Q =>
    Editor.Key.Prompt;
when Esc_R | Esc_S_R =>
    Editor.Search.Replace_Previous;
when Esc_T | Esc_S_T =>
    Editor.Word.Transpose;
when Esc_U | Esc_S_U =>
    Editor.Cursor.Previous;
when Esc_V | Esc_S_V =>
    Editor.Window.Next;
when Esc_X | Esc_S_X =>
    Editor.Macro.Execute;
when Esc_Y | Esc_S_Y =>
    Editor.Hold_Stack.Next;
when Esc_Z | Esc_S_Z =>
    Editor.Window.Previous;

when Esc_Circumflex | Esc_6 | Esc_Tilde =>
    Editor.Word.Capitalize;
when Esc_Greater_Than =>
    Editor.Word.Upper_Case;
when Esc_Less_Than =>
    Editor.Word.Lower_Case;

when Esc_2 | Esc_At_Sign | Esc_Quotation =>
    Editor.Key.Name;
when Esc_C_A | Esc_C_B =>
    Editor.Line.Beginning_Of;
when Esc_C_C =>
    Editor.Line.Copy;
when Esc_C_D =>
    Editor.Line.Delete;
when Esc_C_E =>
    Editor.Line.End_Of;
when Esc_C_H =>
    Editor.Cursor.Left (8);
when Esc_C_I =>
    Editor.Char.Tab_Backward;
when Esc_C_J =>
    Editor.Cursor.Right (8);
when Esc_C_K =>
    Editor.Line.Delete_Forward;
when Esc_C_M =>
    Command.Debug;
when Esc_C_N =>
    Editor.Cursor.Down (8);
when Esc_C_O =>
    Editor.Line.Join;
when Esc_C_T =>
    Editor.Line.Transpose;
when Esc_C_U =>
    Editor.Cursor.Up (8);
when Esc_Up =>
    What.Home_Library;
```

```
                when Delete =>
                    Editor.Char.Delete_Backward;
                when Esc_Del =>
                    Editor.Word.Delete_Backward;
                when Esc_Backslash =>
                    Editor.Char.Delete_Spaces;
                when Esc_C_F =>
                    Editor.Line.Delete_Backward;

                when Enter =>
                    Common.Commit;

--    -- F1 - F3 Debugger Control

--    -- F1 Debugger Execution Control

                when F1 =>
                    Debug.Run;
                when S_F1 =>
                    Debug.Execute;
                when Esc_F1 =>
                    Debug.Run (Stop_At => Debug.Local_Statement);
--             -- when C_F1 => Debug.Stop; -- Interrupts
                when Esc_C_F1 =>
                    Debug.Task_Display;

--    -- F2 Debugger source and breakpoints

                when F2 =>
                    Debug.Source;
                when S_F2 =>
                    Debug.Break;
                when Esc_F2 =>
                    Debug.Activate (Breakpoint => 0);
                when C_F2 =>
                    Debug.Remove (Breakpoint => 0);
                when Esc_C_F2 =>
                    Debug.Show;

--   -- F3 Debugger stack, exceptions and values

                when F3 =>
                    Debug.Put;
                when S_F3 =>
                    Debug.Catch;
                when Esc_F3 =>
                    Debug.Propagate;
                when C_F3 =>
                    Debug.Stack;
--             -- when Esc_C_F3 => Debug.Modify; -- Prompt

--   -- F4 Definition and traversal

                when F4 =>
                    Common.Definition
                       (Name => "<CURSOR>", In_Place => False, Visible => True);

                when S_F4 =>
                    Common.Definition
                       (Name => "<CURSOR>", In_Place => True, Visible => False);
```

```
                when Esc_F4 =>
                    Common.Enclosing (In_Place => False, Library => True);
                when C_F4 =>
                    Common.Enclosing;
                when Esc_C_F4 =>
                    Common.Enclosing (In_Place => True, Library => False);

--   -- F5  Help and access list display

                when F5 =>
                    What.Does ("Help_On_Help");
--             -- when S_F5 => What.Does (""); -- Prompt
                when Esc_F5 =>
                    Editor.Image.Find ("Help Window");
                when C_F5 =>
                    Editor.Key.Name;
                when Esc_C_F5 =>
                    Access_List.Display;

--   -- F6   Ada promotion


                when F6 =>
                    Ada.Install_Unit;
                when S_F6 =>
                    Ada.Code_Unit;
                when Esc_F6 =>
                    Compilation.Promote (Unit => "<Image>",
                                         Scope => Compilation.All_Parts,
                                         Goal => Compilation.Installed,
                                         Limit => "<WORLDS>",
                                         Effort_Only => False,
                                         Response => "<PROFILE>");
                when C_F6 =>
                    Compilation.Make (Unit => "<IMAGE>",
                                      Scope => Compilation.All_Parts,
                                      Goal => Compilation.Coded,
                                      Limit => "<WORLDS>",
                                      Effort_Only => False,
                                      Response => "<PROFILE>");
                when Esc_C_F6 =>
                    Compilation.Make (Unit => "<IMAGE>",
                                      Scope => Compilation.Load_Views,
                                      Goal => Compilation.Coded,
                                      Limit => "<ALL_WORLDS>",
                                      Effort_Only => False,
                                      Response => "<PROFILE>");

--    -- F7  Ada edit/demotion

                when F7 =>
                    Common.Edit;
                when S_F7 =>
                    Common.Demote;
                when Esc_F7 =>
                    Compilation.Demote (Unit => "<SELECTION>",
                                        Goal => Compilation.Source,
                                        Limit => "<WORLDS>",
                                        Effort_Only => False,
```

```
                                        Response => "<PROFILE>");
                when C_F7 =>
                    Ada.Source_Unit;
                when Esc_C_F7 =>
                    Ada.Withdraw;

--  -- F8 Creations

                when F8 =>
                    Common.Create_Command;
                when S_F8 =>
                    Ada.Create_Body;

--          -- when Esc_F8 =>   Library.Create_Directory  -- Prompt
--          -- when ESC_C_F8 => Library.Create_World      -- Prompt
--          -- when C_F8 =>     Text.Create;              -- Prompt

--  -- F9  Ada traversal and show usage

                when F9 =>
                    Ada.Other_Part (Name => "<IMAGE>", In_Place => False);
                when S_F9 =>
                    Ada.Other_Part (Name => "<IMAGE>", In_Place => True);

                when Esc_F9 =>
                    Ada.Show_Usage (Name => "<CURSOR>",
                                    Global => False,
                                    Limit => "<WORLDS>",
                                    Closure => False);
                when Esc_C_F9 =>
                    Ada.Create_Private;
                when C_F9 =>
                    Ada.Show_Usage (Name => "<CURSOR>",
                                    Global => True,
                                    Limit => "<ALL_WORLDS>",
                                    Closure => False);

--  -- F10 Ada semanticize, underlinings, and show unused

                when F10 =>
                    Common.Semanticize;
                when S_F10 =>
                    Ada.Get_Errors;
                when Esc_F10 =>
                    Common.Clear_Underlining;
                when C_F10 =>
                    Ada.Show_Unused (In_Unit => "<IMAGE>",
                                     Check_Other_Units => False);
                when Esc_C_F10 =>
                    Ada.Show_Unused (In_Unit => "<IMAGE>",
                                     Check_Other_Units => True);

--  -- F11  Jobs and print

                when F11 =>
                    Queue.Print;
                when S_F11 =>
                    Job.Enable (0);
                when Esc_F11 =>
                    Job.Connect (0);
```

```
--          -- when Esc_C_F11 => Job.Disable (0); -- Interrupt
--          -- when C_F11 =>     Job.Kill (0);    -- Interrupt

--  -- F12 Information

                when F12 =>
                    What.Time;
                when S_F12 =>
                    What.Load (Verbose => True);
                when Esc_F12 =>
                    What.Users (All_Users => True);
                when C_F12 =>
                    What.Object;
                when Esc_C_F12 =>
                    What.Locks (Name => "<Image>");


                when Up =>
                    Editor.Cursor.Up;
                when S_Up =>
                    Editor.Window.Previous;

                when Down =>
                    Editor.Cursor.Down;
                when S_Down =>
                    Editor.Window.Next;

                when Right =>
                    Editor.Cursor.Right;

                when Left =>
                    Editor.Cursor.Left;

--  -- PF3 => Word
                when Pf3 =>
                    case Key_2 is
                        when '<' =>
                            Editor.Word.Lower_Case;
                        when '>' =>
                            Editor.Word.Upper_Case;
                        when 'd' | 'D' =>
                            Editor.Word.Delete;
                        when 'K' | 'k' =>
                            Editor.Word.Delete_Forward;
                        when 't' | 'T' =>
                            Editor.Word.Transpose;
                        when '^' | '6' | '~' =>
                            Editor.Word.Capitalize;
                        when Delete =>
                            Editor.Word.Delete_Backward;
                        when End_Of | 'e' | 'E' | Tab =>
                            Editor.Word.End_Of;
                        when Begin_Of | 'b' | 'B' =>
                            Editor.Word.Beginning_Of;
                        when Left =>
                            Editor.Word.Previous;
                        when Right =>
                            Editor.Word.Next;
                        when others =>
                            null;
```

```
                end case;

--  -- PF2 => Line
            when Pf2 =>
                case Key_2 is
                    when '<' =>
                        Editor.Line.Lower_Case;
                    when '>' =>
                        Editor.Line.Upper_Case;
                    when '?' | '/' | '+' =>
                        What.Line;
                    when Begin_Of | 'b' | 'B' =>
                        Editor.Line.Beginning_Of;
                    when 'c' | 'C' =>
                        Editor.Line.Copy;
                    when 'd' | 'D' =>
                        Editor.Line.Delete;
                    when End_Of | 'e' | 'E' | Tab =>
                        Editor.Line.End_Of;
                    when 'i' | 'I' =>
                        Editor.Line.Insert;
                    when 'j' | 'J' =>
                        Editor.Line.Join;
                    when 'k' | 'K' =>
                        Editor.Line.Delete_Forward;
                    when 'o' | 'O' =>
                        Editor.Line.Open;
                    when 't' | 'T' =>
                        Editor.Line.Transpose;
                    when '^' | '6' | '~' =>
                        Editor.Line.Capitalize;
                    when Delete =>
                        Editor.Line.Delete_Backward;
                    when '@' | '4' =>
                        Editor.Line.Center;
                    when others =>
                        null;
                end case;

--  -- X2 => Region/Hold_Stack
            when X2 =>
                case Key_2 is
                    when 'X' | 'x' =>
                        Editor.Region.Off;
                    when '<' =>
                        Editor.Region.Lower_Case;
                    when '>' =>
                        Editor.Region.Upper_Case;
                    when 'c' | 'C' =>
                        Editor.Region.Copy;
                    when 'd' | 'D' | 'k' | 'K' =>
                        Editor.Region.Delete;
                    when 'm' | 'M' =>
                        Editor.Region.Move;
                    when '[' | '{' =>
                        Editor.Region.Start;
                    when ']' | '}' =>
                        Editor.Region.Finish;
                    when '^' | '6' | '~' =>
                        Editor.Region.Capitalize;
                    when End_Of | 'e' | 'E' =>
                        Editor.Region.End_Of;
                    when Begin_Of | 'b' | 'B' =>
                        Editor.Region.Beginning_Of;
                    when Left =>
                        Editor.Hold_Stack.Previous;
                    when Right =>
                        Editor.Hold_Stack.Next;
                    when Up =>
                        Editor.Hold_Stack.Top;
                    when Down =>
                        Editor.Hold_Stack.Push;
                    when X6 =>
                        Editor.Region.Fill;
                    when X5 =>
                        Editor.Region.Justify;
                    when others =>
                        null;
                end case;

--  -- X3 => Window
            when X3 =>
                case Key_2 is
                    when '!' | '1' =>
                        Editor.Window.Expand;
                    when '.' =>
                        Editor.Window.Expand (-4);
                    when '?' | '/' | F4 | '+' =>
                        Editor.Window.Directory;
                    when 'c' | 'C' =>
                        Editor.Window.Copy;
                    when 'd' | 'D' | 'k' | 'K' =>
                        Editor.Window.Delete;
                    when 'j' | 'J' =>
                        Editor.Window.Join (1);
                    when 't' | 'T' =>
                        Editor.Window.Transpose;
                    when 'x' | 'X' =>
                        Editor.Window.Delete;
                    when Delete =>
                        Editor.Window.Join (-1);
                    when End_Of | 'e' | 'E' =>
                        Editor.Window.End_Of;
                    when Down =>
                        Editor.Window.Next;
                    when X6 =>
                        Editor.Window.Focus;
                    when X4 =>
                        Editor.Window.Promote;
                    when S_F7 =>
                        Editor.Window.Demote;
                    when F7 =>
                        Editor.Window.Demote;
                    when Begin_Of | 'b' | 'B' =>
                        Editor.Window.Beginning_Of;
                    when Left =>
                        Editor.Window.Parent;
                    when Right =>
                        Editor.Window.Child;
                    when Up =>
```

```
                        Editor.Window.Previous;
                    when others =>
                        null;
                end case;

--  -- PF1 => Image
            when Pf1 =>
                case Key_2 is
                    when End_Of | 'e' | 'E' | Tab =>
                        Editor.Image.End_Of;
                    when Down =>
                        Editor.Image.Down;
                    when Begin_Of | 'b' | 'B' =>
                        Editor.Image.Beginning_Of;
                    when Left =>
                        Editor.Image.Left;
                    when Right =>
                        Editor.Image.Right;
                    when Up =>
                        Editor.Image.Up;
                    when 'I' | 'i' =>
                        Editor.Set.Insert_Mode (True);
                    when 'O' | 'o' =>
                        Editor.Set.Insert_Mode (False);
                    when 'F' | 'f' =>
                        Editor.Set.Fill_Mode (True);
                    when 'X' | 'x' =>
                        Editor.Set.Fill_Mode (False);
                    when others =>
                        null;
                end case;

--  -- PF4 => Mark
            when Pf4 =>
                case Key_2 is
                    when Down =>
                        Editor.Mark.Push;
                    when Right =>
                        Editor.Mark.Next;
                    when Left =>
                        Editor.Mark.Previous;
                    when Up =>
                        Editor.Mark.Top;
                    when Begin_Of | 'b' | 'B' | '[' | '{' =>
                        Editor.Macro.Start;
                    when End_Of | 'e' | 'E' | ']' | '}' =>
                        Editor.Macro.Finish;
                    when X4 | Enter =>
                        Editor.Macro.Execute;
                    when F4 =>
                        Editor.Macro.Bind;
                    when others =>
                        null;
                end case;

            when Numeric_0 =>
                Editor.Set.Argument_Digit (0);
            when Numeric_1 =>
                Editor.Set.Argument_Digit (1);
            when Numeric_2 =>
                Editor.Set.Argument_Digit (2);
            when Numeric_3 =>
                Editor.Set.Argument_Digit (3);
            when Numeric_4 =>
                Editor.Set.Argument_Digit (4);
            when Numeric_5 =>
                Editor.Set.Argument_Digit (5);
            when Numeric_6 =>
                Editor.Set.Argument_Digit (6);
            when Numeric_7 =>
                Editor.Set.Argument_Digit (7);
            when Numeric_8 =>
                Editor.Set.Argument_Digit (8);
            when Numeric_9 =>
                Editor.Set.Argument_Digit (9);
            when Numeric_Comma =>
                Editor.Set.Argument_Prefix;
            when Dash =>
                Editor.Set.Argument_Minus;
            when Dot =>
                Text.End_Of_Input;

--  -- X1 => Object
            when X1 =>
                case Key_2 is

                    when '1' | '!' =>
                        Common.Expand;
                    when '.' =>
                        Common.Elide;
                    when '/' | '?' | '+' =>
                        Common.Explain;
                    when 'c' | 'C' =>
                        Common.Object.Copy;
                    when 'd' | 'D' | 'k' | 'K' =>
                        Common.Object.Delete;
                    when 'g' | 'G' =>
                        Common.Abandon;
                    when 'i' | 'I' =>
                        Common.Object.Insert;
                    when 'l' | 'L' =>
                        Common.Revert;
                    when 'm' | 'M' =>
                        Common.Object.Move;
                    when 'x' | 'X' =>
                        Common.Release;
                    when 'r' | 'R' =>
                        Common.Redo;
                    when 'u' | 'U' =>
                        Common.Undo;
                    when Down =>
                        Common.Object.Next;
                    when Left =>
                        Common.Object.Parent;
                    when Right =>
                        Common.Object.Child;
                    when Up =>
                        Common.Object.Previous;
                    when Begin_Of | 'b' | 'B' =>
                        Common.Object.First_Child;
```

```
                        when End_Of | 'e' | 'E' =>
                            Common.Object.Last_Child;
                        when F4 =>
                            Common.Definition;
                        when others =>
                            null;
                    end case;

                when others =>
                    null;
            end case;

    end case;
end Facit_Commands;
```

RATIONAL

READER'S COMMENTS

Note: This form is for documentation comments only. You can also submit problem reports and comments electronically by using the SIMS problem-reporting system. If you use SIMS to submit documentation comments, please indicate the manual name, book name, and page number.

Did you find this book understandable, usable, and well organized? Please comment and list any suggestions for improvement.

_____
_____
_____
_____
_____
_____

If you found errors in this book, please specify the error and the page number. If you prefer, attach a photocopy with the error marked.

_____
_____
_____
_____

Indicate any additions or changes you would like to see in the index.

_____
_____
_____
_____

How much experience have you had with the Rational Environment?

6 months or less _____ 1 year _____ 3 years or more _____

How much experience have you had with the Ada programming language?

6 months or less _____ 1 year _____ 3 years or more _____

Name (optional)_____ Date_____
Company _____
Address _____
City _____ State _____ ZIP Code _____

Please return this form to:

Publications Department
Rational
1501 Salado Drive
Mountain View, CA 94043

Rational Environment Reference Manual, Keymap: Facit Terminal, 8001A-51

Rational Environment Reference Manual

Master Index: Environment

Document Control Number: 8001A-21 (803-002304)

Rev. 1.0, July 1987
Rev. 2.0, September 1988 (Software Rev. D_10_20_0)

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.

Rational
3320 Scott Boulevard
Santa Clara, California 95054-3197

Master Index

The Master Index combines the indexes for Volumes 2–11 in the *Rational Environment Reference Manual*. Thus, the prefix to the page number is the abbreviation for the book in which the item can be found. This index contains information on key concepts as well as entries for each unit and its declarations, exceptions, errors, enumerations, pragmas, switches, and so on. The entries for each unit are arranged alphabetically by simple name. The first indented item under the simple name is the pathname; second-level indentions, if any, list units in which cross-references appear. An italic page number indicates the primary reference for an entry. Units that are not documented separately as reference entries, but are cross-referenced, appear under their full pathnames (see the ! section at the beginning of the index).

A

C

D

E

F

G

H

I

J

K

L

M

O

P

Q

R

S

T

U

V

W

X

Y

RATIONAL

READER'S COMMENTS

Note: This form is for documentation comments only. You can also submit problem reports and comments electronically by using the SIMS problem-reporting system. If you use SIMS to submit documentation comments, please indicate the manual name, book name, and page number.

Did you find this book understandable, usable, and well organized? Please comment and list any suggestions for improvement.

_____

_____

_____

_____

_____

_____

If you found errors in this book, please specify the error and the page number. If you prefer, attach a photocopy with the error marked.

_____

_____

_____

_____

Indicate any additions or changes you would like to see in the index.

_____

_____

_____

_____

How much experience have you had with the Rational Environment?

6 months or less _____ 1 year _____ 3 years or more _____

How much experience have you had with the Ada programming language?

6 months or less _____ 1 year _____ 3 years or more _____

Name (optional)_____ Date_____

Company _____

Address _____

City _____ State _____ ZIP Code _____

Please return this form to:

Publications Department
Rational
3320 Scott Boulevard
Santa Clara, CA 95054-3197

JANC

COMMENTS

n is for instruction comments only. You can also submit problem reports and
etronically by using the EMS problem-reporting system. If you use EMS to
e the comments, please reference the manual name, book name, and page number.

e book understandable, usable, and well organized? Please comment and list any
ion improvement.

ere in this book that is simply the error and the page number. If you prefer, citation a
ter(s) is...

tion changes you would like to see in this book.

ere have you read with the Palatka Environment?

ss _____ 1 year _____ 3 years or more _____

ere have you read with this same programming language?

s than 1 year _____ 1 year _____ 3 years or more _____

_____ Date _____

_____

_____ State _____ Zip Code _____

ns to:

Publications Department
Rational
3320 Scott Boulevard
Santa Clara, CA 95054-3197

RATIONAL

READER'S COMMENTS

Note: This form is for documentation comments only. You can also submit problem reports and comments electronically by using the SIMS problem-reporting system. If you use SIMS to submit documentation comments, please indicate the manual name, book name, and page number.

Did you find this book understandable, usable, and well organized? Please comment and list any suggestions for improvement.

_____
_____
_____
_____
_____
_____

If you found errors in this book, please specify the error and the page number. If you prefer, attach a photocopy with the error marked.

_____
_____
_____
_____

Indicate any additions or changes you would like to see in the index.

_____
_____
_____
_____

How much experience have you had with the Rational Environment?

6 months or less _____ 1 year _____ 3 years or more _____

How much experience have you had with the Ada programming language?

6 months or less _____ 1 year _____ 3 years or more _____

Name (optional)_____ Date_____
Company _____
Address _____
City _____ State _____ ZIP Code _____

Please return this form to:

Publications Department
Rational
3320 Scott Boulevard
Santa Clara, CA 95054-3197