Rational Environment Reference Manual

Session and Job Management (SJM)

Copyright © 1987 by Rational

Document Control Number: 8001A-24

Rev. 1.0, July 1987 (Delta)

This document subject to change without notice.

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

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

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

Rational
1501 Salado Drive
Mountain View, California 94043

Contents

RATIONAL

How to Use This Book

The Session and Job Management (SJM) book of the *Rational Environment Reference Manual* contains reference information describing some of the commands and tools for managing work sessions and jobs. It is intended for users who are familiar with the Rational Environment™ and with Ada® programming.

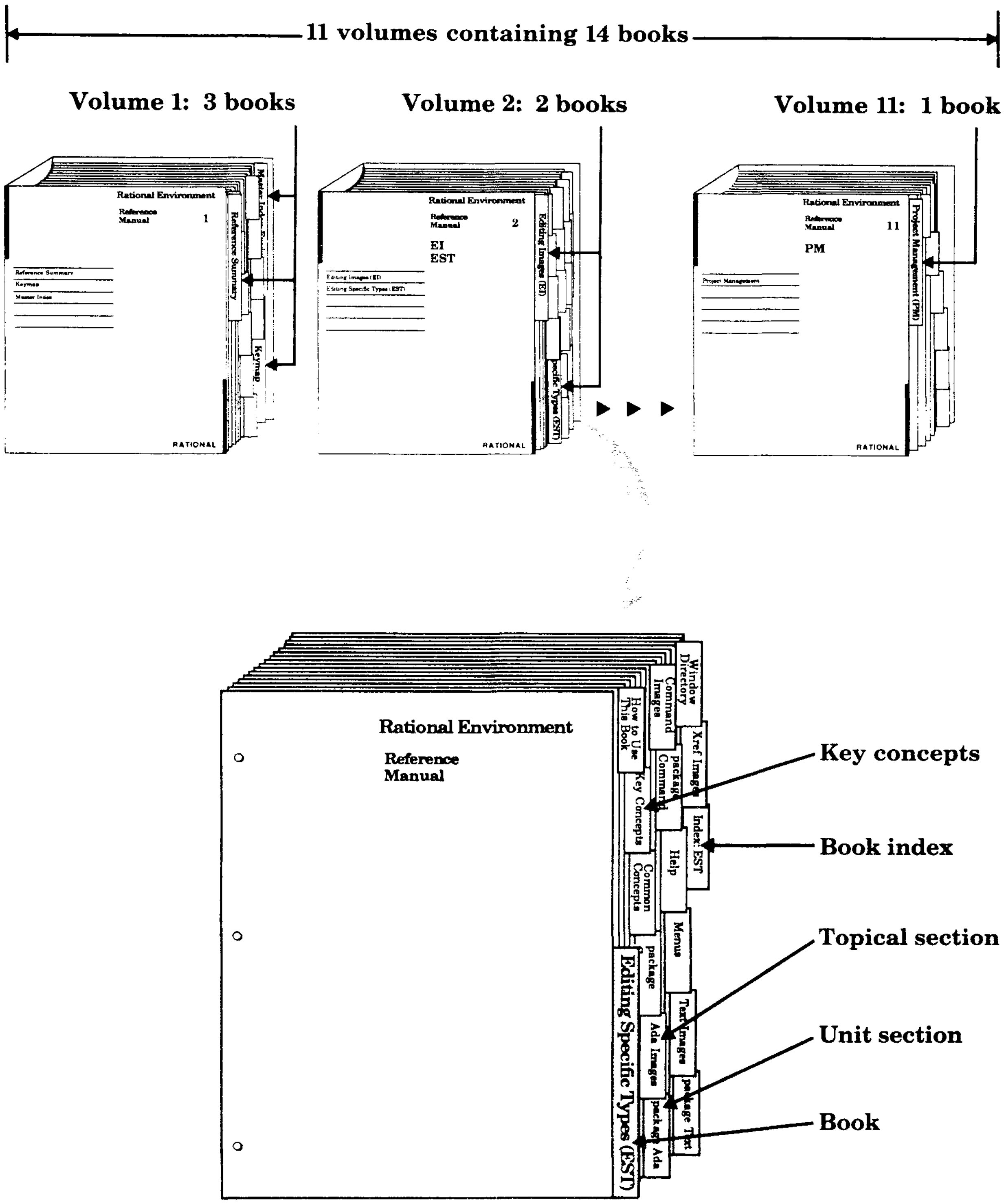
Organization of the Reference Manual

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.

Key Concepts

Managing your session and jobs suggests several things: tailoring your work sessions, initiating and running jobs, and utilizing Environment resources. This section of the *Rational Environment Reference Manual* discusses several packages and facilities that aid in managing your sessions and your jobs. These packages are:

- Job: Defines a set of procedures that control *jobs*. A job is initiated when commands are executed or explicitly through the Program package.
- Log: Defines a set of operations that create, manipulate, and filter *logs* that are generated or added to by other operations in other packages.
- Operator: Defines a set of operations for system managers, such as creating users, overseeing user sessions, managing groups for access control, enabling and disabling physical lines, and performing operations independent of access control. Only the operations that users perform are documented in this section. The complete package Operator is documented in System Management Utilities (SMU).
- Profile: Defines a set of operations for tailoring your *profile*. A profile is a collection of error responses, logging filters, and activities that are specific to your session or to a job.
- Program: Defines a set of operations that constitute a means for compiling and executing programs from other programs.
- Queue: Defines a set of operations for printing and setting up print queues. Only the printing operations are described in this section. The complete package Queue is documented in System Management Utilities (SMU).
- Search_List: Defines a set of operations for creating, editing, and using *searchlists*. This package also describes the type-specific editing operations available on searchlist images. A searchlist is a set of directories in which names are searched for when a command is executed.
- Session Switches: Defines a set of operations for creating, editing, and manipulating *session switches*. This section also describes the type-specific editing operations available on switch images. Switches provide a means of tailoring specific attributes of the editor, compilation system, pretty-printer, or other Environment facility.
- What: Defines a set of operations that return information about the current session and the Environment. This package includes Help facilities.

These packages and their commands are described in detail later in this book of the *Rational Environment Reference Manual.* Some concepts that apply to several of these facilities are described in the remainder of this Key Concepts section.

Library System

The library system in the Environment is a hierarchy of directories and worlds. Both are referred to as *libraries*. Special attributes are attached to worlds to form points for controlling resources. These attributes differentiate directories from worlds.

Objects in the hierarchy can have multiple versions. Each version is assigned a number by the Environment when the version is created. Each object has, at most, one current version and possibly several deleted versions. Deleted versions are retained until expunged or until newer versions are created. Objects can also be deleted, but they are retained until they are expunged.

Worlds are closed scope for Ada naming and require that program units in the world that need facilities outside the world explicitly import those needed facilities. These imports are specified in the set of links that are associated with the enclosing world.

Worlds

Worlds are used primarily where the contents of the structure are other structural elements or programs.

The root of the library system is a world called !. The home library of all users is a world. Another common use of worlds is for project-specific libraries.

Worlds have the special attribute that they and their contents are built on a specific disk volume.

Worlds also contain the set of links that import facilities for program units in the world or in any directories in the world.

Directories

Directories are used to contain related Ada units, main programs, or other directories or worlds.

Directories behave just as worlds do except that directories contain no links and they always exist on the same disk volume as their parent world.

Tailoring Your Session

You can tailor Environment behavior in several ways.

Error Reactions

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

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

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

The Environment has *special values* (used as parameters to commands) for which *profile* it should use when responding to errors in a command. The three most commonly used are "<PROFILE>", "<SESSION_PROFILE>", and "<DEFAULT>", which refer, respectively, to the job response profile, the session response profile, and the system default response profile returned by the Profile.Default_Profile function. See package Profile for further information on profiles.

Switches

Each user's session also has a set of switches that control Environment behavior on the user's terminal. Called *session switches*, they control window size, scrolling style, and output window formats. Session switches are documented in this book.

Switches are maintained in files. There are commands in package Switches for creating, setting, and displaying switch values in these files. In addition to these commands, the type-specific editing operations available on switch images are documented with package Switches in Library Management (LM). Library switches are also documented in package Switches.

Log Files

Many commands in the Environment produce logs of their operations. These logs are displayed in a window or written into a file, as specified by Current_Output or Current_Error, as they are created. The Environment initializes this to be the default output window. Thus, by default, the logs are displayed in the output window.

Several kinds of messages are produced by these commands. Commentary messages are general notes. Warning, error, and exception messages describe a problem that has arisen in the command. Progress messages indicate whether the command is making positive, errorless progress or negative, erroneous progress.

All of these messages are marked with a character sequence that indicates the kind of message it is. This allows the user to scan a log file looking for a particular character sequence. Several procedures exist in this package to scan for these sequences. Other tools that automatically scan for these sequences can also be built.

There are also commands for inserting user-defined messages into the log file. The content of these messages can be specified to be of any kind.

Logs are generated under the control of the current profile. This profile is manipulated with procedures from package Profile. The logs can then be manipulated with procedures from package Log.

Searchlists

Environment resources are available for use in programs or for executing directly. Links, which are discussed in Library Management (LM), are the mechanism by which closed-scope worlds have access to other resources in other worlds. Resources to be used from a Command window must be accessible via a mechanism called ***searchlists.***

Resolving names in a Command window requires searching in specific directories or worlds. Thus, searchlists contain a list of directories and worlds.

Searchlists are a way to name objects that do not reside in the current context. A searchlist is a list of directories and worlds; the names that occur in a Command window are searched for in those directories and worlds.

The Environment has the following default searchlist for all users and sessions:

```
1. $`
2. !COMMANDS
3. !COMMANDS.ABBREVIATIONS`
4. !MACHINE.RELEASE.CURRENT.COMMANDS`
5. !IO
6. !TOOLS
```

This searchlist provides access to the commands and utilities in the Environment. The first component of the searchlist is the current context and then the set of links. The others provide access to specific directories in the Environment.

All special naming characters can be used in searchlists. The most common one is the grave (`). This often appears at the end of a searchlist component and indicates that the links in the named world should be used when searching in that world. In particular, the default searchlist component of !Commands.Abbreviations` uses that feature to gain access to the links in that world. See "Naming Objects," below, for more information about special naming characters.

Naming Objects

Many commands in the Environment require a way of *naming* objects in the Environment to move those objects or to perform operations on those objects. The Environment uses two forms of naming: Ada names and string names. Ada names are used in program units or when executing a command. String names are typically used in the parameters to Environment commands.

Ada names are used to call an Environment command in a Command window or to reference an Ada unit in a program. Ada names are the extended Ada names as defined in the *Reference Manual for the Ada Programming Language.* Ada names are used to reference Ada units only. Files, worlds, directories, and other non-Ada units in the Environment cannot be referenced with an Ada name.

String names are used as arguments to commands. These strings are very similar to Ada names but can be used to reference any object in the Environment. Also, string names have five important additions: *special names*, *parameter placeholders*, *wildcards*, *special characters*, and *attributes*. The ability also exists to create a set of names using simple set notations and to substitute characters.

Special Names

Special names are used as parameter values for many Environment operations to specify text, objects, and regions. Special names allow you to specify selections and designations without providing a pathname. Anywhere that a string name can be used, special names can be used. They take the form *"<special name>"*, where *special name* specifies text, object, region, or activity, as described below:

`"<SELECTION>"`	References the highlighted object, if the cursor is located in a highlighted area.
`"<REGION>"`	References the highlighted object.
`"<CURSOR>"`	References the object on which the cursor is located, whether or not there is a highlighted area in the window.
`"<IMAGE>"`	References the highlighted object, if the cursor is in a highlighted area. If the cursor is not located in the highlighted area, this special name references the image on which the cursor is located.
`"<TEXT>"`	References the object named in the highlighted text in the image in the window.
`"<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.

Special names are used as default parameter values to many operations. The user can replace them with another special name or other form of string name, as accepted by that operation.

Special Values

Many operations in the Environment have a Response parameter that specifies how the command should respond to errors. The Response parameter takes special values, as described in "Error Reactions," above.

Parameter Placeholders

Many Environment commands use parameter placeholders as default parameter values. They take the form "*>>parameter placeholder<<*". This naming convention is used, as its name suggests, as a placeholder in parameters to indicate the type of string name that must be entered to replace it. Executing a command containing a parameter placeholder will result in an error. Parameter placeholders include:

```
">>FILE NAME<<"
">>SOURCE NAMES<<"
">>SWITCH<<"
">>SWITCH FILE<<"
">>SWITCHES<<"
">>WORLD NAMES<<"
```

For example, an operation that has the `">>FILE NAME<<"` parameter placeholder requires a filename, such as "!Users.John.File_1".

Wildcards

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

The Wildcard

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

Any wildcard can be used to represent a set of named objects. For example, if there are objects in the directory !Users.Stooges called Larry, Curly, and Moe, then a single string, such as !Users.Stooges.####y, can be created to refer to the first two of them.

The Wildcard @

The *at* sign (@) represents zero or more identifier characters in a name, including the underscore (_). It does not match any subunits of Ada units. It can be used several times within a single name. For example, the name !Users.Fred.Food can be written !U@.@.Food.

This wildcard can be used to represent a set of named objects. For example, if there are objects in the directory !Users.Stooges called Larry, Curly, and Moe, then a single string, such as !Users.Stooges.@, can be created to refer to all three of them.

This wildcard can be combined with special characters (see "Special Characters," below) to create very short names that represent sets of objects in the current context. As before, if there are three Ada units in the current context called Larry, Curly, and Moe, then the string @ can be used to represent all three Ada units, but it would not include their subunits.

The Wildcard ?

The question mark (?) represents zero or more components in a name, which are not worlds or objects contained by those worlds. For example, the name !Users.Stooges? represents the Ada units called Larry, Curly, Moe, and any of their subunits.

Also note that the periods before and after the wildcard are optional. For example, the name A.?.B is equivalent to the name A?B.

The Wildcard ??

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

Also note that the periods before and after the wildcard are optional. For example, the name A.??.B is equivalent to the name A??B.

Substitution Characters

Similar to the way in which wildcard characters can be used to specify a source group of objects, substitution characters can be used to create target names from source names.

The substitution characters and their definitions are described below. Note that if a substitution character is encountered after all segments/wildcards have been exhausted, the characters are replaced by the null string. If # or ? is replaced by the null string, an immediately following period (.) is also elided from the resulting string.

The Substitution Character

The pound sign (#) is replaced by the next complete (right to left) segment in a name. For example, if there are Ada units in the world !Users.Stooges called Larry, Curly, and Moe, and the user wants to copy them into a world !Users-.Stooges.New_World, the user could build the target name parameter (from the source name parameter !Users.Stooges) using substitution characters as follows: !#.#.New_World.#.

The Substitution Character @

The *at* sign (@) is replaced by the portion of the current segment that is matched by a wildcard in the source name. If there is more than one wildcard in the segment, a separate @ is needed in the target to match each one. Matching is performed from right to left. (For the purpose of this matching, @, #, ?, and ?? are considered wildcards.)

For example, there is a world called !Users.Gzc containing files File_1 through File_50. The user wants to rename these objects as My_File_1 through My_File_50. The source name parameter would be !Users.Gzc.File_@. The target name parameter, using substitution parameters, would be !#.#.My_File_@.

The Substitution Character ?

The question mark (?) is replaced by successive full segments, working from right to left, until the segment for a world is encountered. For example, to copy everything in the world up through the next-level world, !Users.Mary to !Users.John, the source string would be !Users.Mary?? and the target string would be !Users.John?.

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. This climbs the hierarchy of objects and eventually reaches the root of the library system. This 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 library system is itself.

A special use of this character occurs in combination with a bracketed name. A name component of the form ^[some_unit] resolves to the closest containing object whose simple name is Some_Unit. Brackets are normally used for creating sets of objects.

The caret can also be used as a shorthand method for referring to objects in a parent unit. For example, if the current context is !Users.Pete, another user named Joe can be referred to as !Users.Joe or simply ^Joe.

The Special Character $

The dollar sign ($) specifies that the context should be set to the immediately enclosing library. A library is either a directory or a world. 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 (%) is used only in the Rational Debugger and 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 !Tools.Debug_Tools.Set_Task_Name command or task numbers assigned by the Environment. The !Commands.Debug.Task_Display command 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), then 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 some Environment commands. If the first character after the underscore is an alphabetic character, then 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. It must also be used when specifying an activity file as an indirect 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, then the frame on the run-time stack that contains an activation of that subprogram must be named. Renaming is done using the notation *_frame number*. Stack frames are numbered for each task starting at the top with 1. For example, _4 refers to frame number 4 (fourth frame from the top). Frames are alternately numbered from the bottom using negative numbers.

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. For example, in the name Commands.Ada, the period separates the two components of the 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. For example, a name such as Larry would be evaluated in the current context. However, a name such as \Larry would be evaluated in each of the contexts of the searchlist in turn until all occurrences of the name Larry are found in those contexts. If more than one occurrence is found, a menu showing all occurrences is displayed.

More information about searchlists can be found in "Searchlists," above.

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. If the current context is an installed or coded Ada unit, the grave is resolved according to LRM rules.

This kind of naming does not allow for renamed packages or instances of generic packages or subprograms to be used. This kind of naming does not "look through" renaming declarations.

More information about links can be found in "Searchlists," above.

The Special Characters []

Brackets ([]) define a set notation. Sets are created by enclosing a series of name components, separated by commas, in brackets. For example, the name [Larry, Curly, Moe] represents only those three objects in the current context.

The semicolon character 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. For example, Foo?['C(Lib), 'Spec] matches any component of Foo that is either a library or an Ada spec. Foo [A;B] must match A and B in Foo.

Names can also be excluded from a set with the tilde (˜). For example, the name [@, ˜Curly] represents all names in the current context except the name Curly.

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 as well as objects that have not been deleted. For example, if the object Curly is deleted but not expunged, the name @ refers only to Larry and Moe, but the name {@} refers to Larry, Curly, and Moe.

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.

Visible Parts and Bodies

Names normally are searched for in both the visible part and the body of the current context. These attributes can restrict the resolution to either the visible part or the body.

Two Ada unit attributes are defined:

- 'Body: Any remaining name components specify an object in the body of the named unit.
- 'Spec: Any remaining name components specify an object in the visible part of the named unit.

If no attributes are used for a particular name component, the entire unit, visible part and body, is used to resolve any additional name components. This allows names to be created that specify objects not visible through Ada visibility rules.

Version Attributes

Objects in the directory system can have more than one version. It is necessary, therefore, to distinguish which version of the object is desired. By default, the most recent version is used; if some other version is desired, an attribute can be appended to the name to specify a specific version.

Examples of version attributes are Larry'V(2), Curly'V(ALL), and Moe'V(-1). The value in the parentheses can be any of the following:

ALL	Matches *all* versions of the object.
ANY	Matches the *default* version of the object.
MAX	Matches the *newest* version of the object.
MIN	Matches the *oldest* version of the object.
n	Matches the version with that *version number*.
−*n*	Matches the *n*th version preceding the current version. For example, −1 matches the version created just before the current version.

Class Attributes

Objects in the directory system consist of different classes and subclasses. A class or subclass attribute can be used to distinguish which class or subclass of objects is being named. By default, a name assumes any class of object.

Examples of class attributes are L@'C(LIBRARY) and Moe?'C(FILE). The value in the parentheses includes the following classes:

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

There are many subclasses associated with each class. These are described in Tables 1-1, 1-2, and 1-3.

Table 1-1. Library Class

Subclass Name	*Description*
Comb_Ss	Subsystem containing combined view that cannot contain spec or load views (see PM book)
Comb_View	Combined view of a subsystem (see PM book)
Directory	Directory
Load_View	Load view of a subsystem (see PM book)
Mailbox	Library containing Mail and Mail_Db files for the Rational Mail Utility
Spec_Load	Subsystem that cannot contain combined views (see PM book)
Spec_View	Spec view of subsystem (see PM book)
Subsystem	Subsystem (see PM book)
World	World

Link Attributes

The 'L attribute can be used for matching the link name in the set of links associated with a world. For example, My_World'L(My_Link) matches the link named My_Link in the set of links associated with My_World. The link attribute can take an argument of Any, External, Internal, or any prefix of these. Any specifies either external or internal links, External specifies external links (that is, links referencing units outside of the enclosing world), and Internal specifies internal links (that is, links referencing objects within the same enclosing world). For example, Your_World'L(Any)A@ matches all links beginning with the letter A in the set of links for Your_World.

State Attribute

The 'S attribute can be used for matching Ada units in a particular state. The state attribute can take an argument of Archived, Source, Installed, Coded, or the first letter of any of these. Archived specifies units in the archived state, Source specifies units in the source state, and so on. For example, !Users.John?'S(Coded) specifies all units in the coded state in John's home library.

Nickname Attributes

Names of subprograms in an Ada unit can be overloaded. A subprogram can be given a unique nickname via the Nickname pragma, which follows the declaration of the subprogram.

An example of nickname attributes is Larry'N(first). The value in the parentheses can be any alphanumeric identifier that corresponds to a nickname that has been defined via the Nickname pragma.

Table 1-2. Ada Class

Subclass	*Description*
Alt_List	Alternative list insertion point
Comp_Unit	A compilation unit that has not been semanticized
Context	Context clause insertion point
Decl_List	Declaration list insertion point
Func_Body	Function body
Func_Inst	Generic function instantiation
Func_Ren	Function rename
Func_Spec	Function specification
Gen_Func	Generic function
Gen_Pack	Generic package
Gen_Param	Generic parameter insertion point
Gen_Proc	Generic procedure
Insertion	Insertion point
Load_Func	Code-only function
Load_Proc	Code-only procedure
Main_Body	Main function body
Main_Body	Main procedure body
Main_Func	Main function specification
Main_Proc	Main procedure specification
Pack_Body	Package body
Pack_Inst	Generic package instantiation
Pack_Ren	Package rename
Pack_Spec	Package specification
Pragma	Pragma insertion point
Proc_Body	Procedure body
Proc_Inst	Generic procedure instantiation
Proc_Ren	Procedure rename
Proc_Spec	Procedure spec
Statement	Statement insertion point
Subp_Body	Subprogram body
Subp_Inst	Generic subprogram instantiation
Subp_Ren	Subprogram rename
Subp_Spec	Subprogram specification
Task_Body	Task body

Table 1-3. File Class

Subclass	*Description*
Activity	Activity file (see PM book)
Binary	Binary file
Cmvc_Db	CMVC database (see PM book)
Code_Db	Code saved for a subsystem load view (see LM, package Archive, and PM book)
Compat_Db	Compatibility database for a subsystem
Config	Configuration pointer for CMVC (see PM book)
Dictionary	For future development
Documents	Document database (part of Rational Design Facility)
File_Map	File map
Log	Log file
Mail	Collections of messages (part of Rational Mail Utility)
Mail_Db	User's mailbox (part of Rational Mail Utility)
Msg_In	For future development
Msg_Out	For future development
Objects	Object set
PS	PostScript file (part of Rational Design Facility)
Search	Searchlist file
Switch	Switch file
Swtch_Def	Switch definition file
Text	Text file
Venture	A collection of work orders for CMVC (see PM book)
Work	Work order for CMVC (see PM book)
Work_List	Work order list for CMVC (see PM book)

The Options Parameter

Many of the commands in the Environment have an optional *options specification* in the form of a parameter called Options. This options specification accepts different strings, depending on the command specified.

Syntax Rules

The general form of the Options parameter is *option=>value*, where *option* is the name of an option that modifies the way in which an operation behaves and the => symbol, called a *value delimiter*, separates the *option* from the value in *value*. Other permissible value delimiters are the symbols colon equals (:=) and equals (=). For example, in the !Commands.Archive.Restore procedure, all of the following specifications of the same option are permissible:

```
"AFTER=>12/25/86"
"AFTER:=12/25/86"
"AFTER=12/25/86"
```

If more than one Options parameter is to be specified, the Options parameter must be separated by commas (,) or semicolons (;). For example, in the !Commands.Archive.Restore procedure, the following two options might be used:

```
"AFTER=12/25/86,FORMAT=R1000"
```

Options that take string values containing comma or semicolon characters must have the string enclosed in parentheses. For example:

```
"LABEL=(MONDAY, JANUARY 26, 1987)"
```

Two or more options that will be assigned the same value can be combined by separating them with the vertical bar (|), with the value delimiter and value following the last option. For example, two access control options from the !Commands.Archive.Restore procedure that might take the same value could be specified as:

```
"OBJECT_ACL|DEFAULT_ACL=>(JOHN=>COD)"
```

Sequentially enumerated options that will be assigned the same value can be specified by listing only the first and last options, separated by the double dots (..). For example, in package !Tools.Profile, all log messages can be turned off by using the option:

```
"Auxiliary_Msg..Dollar_Msg=>False"
```

Boolean Options: A Special Case

For Boolean options, the value delimiter and value are optional. When they are not specified, the value of the Boolean option is true. To make it false without using the value delimiter and value, it can be preceded with the tilde (~). For example, specification of the REPLACE Boolean option for the !Commands.Archive.Restore procedure can be done by specifying any of the following:

```
"REPLACE"
"REPLACE=>TRUE"
"REPLACE:=TRUE"
"REPLACE=TRUE"
```

It can be set to false by using any of the following:

```
"~REPLACE"
"REPLACE=>FALSE"
"REPLACE:=FALSE"
"REPLACE=FALSE"
```

When Boolean options are specified without the value delimiter and value, the options can be separated by spaces only—for example, "REPLACE PROMOTE" from the Archive.Restore procedure.

Boolean sequential enumerations can also be specified without the value delimiter and value. Using the example from package Profile above, the option could be specified:

```
"~Auxiliary_Msg..Dollar_Msg"
```

Literals in Options: A Special Case

For literals of the form *literal=value*, the *literal* and *value delimiter* are optional.

RATIONAL

package Job

Package Job includes procedures for managing system resources and for controlling jobs associated with user sessions. These procedures are available to all users in the Environment for their own jobs. Users can disconnect or reconnect to their own jobs and can kill their own jobs.

Execution of several of the operations in this package requires that the username executing the commands have operator capability. See SMU, introduction to package Operator, for further information on operator capability.

Procedures for stopping, restarting, and terminating a job require job numbers or job identifiers. The Environment assigns job identification numbers uniquely for each session. The range is from 0 through 255.

Interactive display and control of jobs can be accomplished with the display produced by the What.Jobs procedure. See the reference entry for this procedure for further information.

procedure Connect

```
procedure Connect (The_Job : Id := 0);
```

Description

Connects the terminal to the specified job.

This effectively suspends terminal operation until the job is finished. After this procedure is executed, the job can read keyboard input with facilities in package Window_Io (DIO). A job can be disconnected by using the Interrupt procedure (the [Control][G] combination).

Parameters

```
The_Job : Id := 0;
```

Specifies the job number for a job associated with the current session. The default, 0, connects you to the most recently interrupted job for the current session.

Example

Given the following What.Users display:

```
    User Status on June 30, 1987 at 04:22:52 PM

      User     Line  Job    S       Time                  Job Name
    ========   ====  ====  ====  =========  ========================
     WET        21         RUN   53:04.96  session'(S_1)
                      232  IDLE  00:02.45  !USERS.WET % COMPILATION.MAKE
                      218  IDLE  02:37:01  !USERS.WET % WHAT.LOCKS
```

user WET has two jobs associated with session S_1. The following command:

```
   job.connect (232)
```

reconnects to job 232 in the current session.

References

procedure Interrupt

procedure Disable

```
procedure Disable (The_Job     : Id;
                   The_Session : String := "");
```

Description

Stops the specified job temporarily.

The job is not terminated but is prevented from executing by the Environment. A job identifier value of 0 stops all jobs for the current session. You cannot use the 0 value to stop jobs for a session other than your own.

The Enable procedure can be used to allow the job to execute again. You can disable only the jobs that belong to the username that you are working under, unless the username has operator capability, is running in privileged mode, or has read access to the session object. (Session objects do not themselves have access lists (ACLs). Read access to the enclosing library is required to have read access to the session object.) To stop jobs created by sessions other than your current session, you must supply the pathname of the session creating the job.

Parameters

`The_Job : Id;`

Specifies the job number for the job to be disabled. The job must be associated with the current username. A value of 0 stops all jobs for the current session. You cannot use the 0 value to stop jobs for a session other than your own.

`The_Session : String := "";`

Specifies the pathname of the session to which the job belongs. The name is the session object that the Environment creates in the user's home world. An example is !Users.Rjb.S_1. This name must resolve to a session object. The default null string value of the parameter represents the current session.

Example

Given the following What.Users display:

```
User Status on June 30, 1987 at 04:22:52 PM

  User     Line  Job    S       Time                    Job Name
 ========  ====  ===   ====  =========  ========================
 WET         21        RUN   53:04.96   session'(S_1)
                 232   IDLE  00:02.45   !USERS.WET % COMPILATION.MAKE
                 218   IDLE  02:37:01   !USERS.WET % WHAT.LOCKS
```

there are two jobs associated with the current session S_1 of user WET. The following command:

```
job.disable (218)
```

suspends job 218 in the current session.

References

procedure Enable

procedure Disconnect

```
procedure Disconnect (The_Job : Id := 0);
```

Description

Disconnects from a job.

This procedure is essentially equivalent to the Interrupt procedure, but it is not meant to be bound to a key. It should be used within an Ada procedure. This procedure forces the job calling it to run in the background rather than in the foreground.

Parameters

```
The_Job :  Id := 0;
```

Specifies the number of the job from which to disconnect. The default, 0, disconnects from the current job.

Restrictions

This procedure should not be bound to a key. Use the Interrupt procedure instead.

Example 1

The procedure can be used to force any program to run in the background, even if it started to run in the foreground. A program of the form:

```
procedure Main is
begin
    Job.Disconnect;
    ...
end Main;
```

always runs in the background even if it is executed from a Command window. This program performs basically the same as if a program had been executed from a Command window and then was interrupted by the Interrupt procedure.

Example 2

The procedure can also be used to force a program to run in the background after running in the foreground. A program of the form:

```
procedure Main is
begin
    ...  -- gather editor state: selection, current image,
    ...  -- cursor position, etc.
    Job.Disconnect;
    ...  -- process information
end Main;
```

runs in the background after gathering information about the session's current state, such as selections. The use of this procedure runs the program in the foreground for some amount of time and then returns control of the terminal to the user.

References

procedure Interrupt

procedure Enable

```
procedure Enable (The_Job     : Id;
                  The_Session : String := "");
```

Description

Restarts the specified disabled job.

A job identifier value of 0 restarts all jobs for the current session. You can only enable jobs that belong to your username, unless the username has operator capability, is running in privileged mode, or has read access to the session object. (Session objects do not themselves have access lists (ACLs). Read access to the enclosing library is required to have read access to the session object.) To stop jobs created by sessions other than your current session, you must supply the pathname of the session creating the job.

Parameters

`The_Job : Id;`

Specifies the job number of the job to be restarted. The job must be associated with your current username. A job identifier value of 0 restarts all jobs for this session.

`The_Session : String := "";`

Specifies the pathname of the session to which the job belongs. The name is the session object that the Environment creates in the user's home world. An example is !Users.Rjb.S_1. This name must resolve to a session object. The default value of the parameter represents the current session.

Example

Given the following What.Users display:

```
User Status on June 3Ø, 1987 at Ø4:22:52 PM

 User     Line  Job   S      Time                 Job Name
=======   ====  ===  ====  =========  ============================
 WET        21       RUN    53:Ø4.96  session'(S_1)
                232  IDLE   ØØ:Ø2.45  !USERS.WET % COMPILATION.MAKE
                213  IDLE   Ø2:37:Ø1  !USERS.WET % WHAT.LOCKS
```

there are two jobs associated with the current session S_1 of user WET. The following command:

```
    job.enable (218)
```

restarts job 218 in the current session.

subtype Id

```
subtype Id is Machine.Job_Id;
```

Description

Defines a subtype that identifies jobs.

procedure Interrupt

```
procedure Interrupt;
```

Description

Places the current job in background mode and frees the user's session to initiate other commands.

This procedure places the current job in background mode and returns control of the current session to the user. The job continues to execute in the background. This procedure is intended to be entered from the keyboard with the [Control][G] combination. To interrupt from an Ada procedure or program, use the Disconnect procedure.

Once a job has been interrupted, a message similar to `User Interrupt (Job 213 queued)` appears in the Message window.

After the job has been disconnected, the cursor position is the same as it was at the beginning of the job. Thus, a job that requires knowledge of cursor position to execute can be interrupted, leaving the user free to perform other commands. This includes resolution of any special name as it was at the beginning of the job.

Restrictions

This procedure should only be bound to a key. Use the Disconnect procedure from programs or Command windows.

References

procedure Disconnect

---

procedure Kill

```
procedure Kill (The_Job     : Id;
                The_Session : String := "");
```

Description

Kills the specified job.

The procedure kills the specified job that belongs to your current username. All files declared as Current_Output, Current_Input, Current_Error, or files on the current output, current input, or current error stacks are all closed (see package Log for more information about these stacks). All other files are abandoned, and temporary files are deleted.

You cannot kill a job for another username unless the username you are using has operator capability, is running in privileged mode, or has read access to the session object. (Session objects do not themselves have access lists (ACLs). Read access to the enclosing library is required to have read access to the session object.) To stop jobs created by sessions other than your current session, you must supply the pathname of the session creating the job.

Parameters

```
The_Job :  Id;
```

Specifies the job identification number.

```
The_Session :  String := "";
```

Specifies the pathname of the session to which the job belongs. The name is the session object that the Environment creates in the user's home world, such as !Users.Rjb.S_1. This name must resolve to a session object. The default value of the parameter represents the current session.

Example

Given the following What.Users display:

```
User Status on June 30, 1987 at 04:22:52 PM

  User    Line  Job  S      Time                  Job Name
========  ====  ===  ====  =========  =============================
WET        21        RUN    53:04.96  session'(S_1)
                232  IDLE   00:02.45  !USERS.WET % COMPILATION.MAKE
                218  IDLE   02:37:01  !USERS.WET % WHAT.JOBS
```

there are two jobs associated with the current session S_1 of user WET. The following command:

```
job.kill (218)
```

kills job 218 in the current session.

References

package Log

procedure Set_Termination_Message

```
procedure Set_Termination_Message (S : String := "");
```

Description

Defines the message that is displayed when the job terminates.

Parameters

```
S : String := "";
```

Specifies the message to be displayed. The default is no termination message.

Example

The command:

```
    job.set_termination_message("Terminated  with no errors");
```

when placed inside a program called Check_Sum, produces the following termination message in the Message window:

```
    !USERS.RJB % Check_Sum terminated: Terminated with no errors
```

end Job;

RATIONAL

package Log

This package provides facilities for manipulating the log files that are generated by many procedures in the Environment. For example, logs are generated by procedures in packages Access_List, Archive, Compilation, Library, Links, and Tape (all of which are in !Commands). These logs can be created, redirected, appended to, copied, or filtered with the procedures in this package. User-written commands can use these same facilities by writing output with the Put_Line procedure defined in this package. The log file can be defined as any of the following: Current_Output, Current_Error, Standard_Output, or Standard_Error, all defined in package !Io.Io. The default is Current_Output, but it may be changed using commands in package Profile. The Current_Output and Current_Error files can be assigned to any file, using package Io. The default Current_Output is the text output window. The default Current_Error is the Message window.

Log files have a form that is specified by a *profile*. In addition, profiles determine the response to errors and which activities to use during the execution of various commands in the Environment. The job profile, denoted by "<PROFILE>", is the default profile used by all Environment commands. The job profile is set to the session profile at the start of each job and can be changed by using commands in package Profile.

Each message in a log typically contains a three-character sequence that specifies the kind of message it is. This three-character sequence is used by many of the procedures in this package to filter the logs. A filter can be defined, either explicitly with procedures in this package or implicitly as part of a profile, and used to filter out unwanted messages.

Package Profile contains a variety of facilities for defining and manipulating profiles. Any explicit profile can be constructed and used in these commands with procedures in package Profile.

Stacks of log files can also be created. For further information, see TIO, package Io.

Special Names

Many of the commands in this package have *special names* as default values to parameters requiring names. Anywhere that a string name can be used, a special name can be used. Special names allow you to designate without supplying a pathname. They take the form "<*special name*>", where *special name* specifies the text, object, region, or activity, as described below:

`"<SELECTION>"`	References the highlighted object, if the cursor is located in a highlighted area.
`"<REGION>"`	References the highlighted object.
`"<CURSOR">`	References the object on which the cursor is located, whether or not there is a highlighted area in the window.
`"<IMAGE>"`	References the highlighted object, if the cursor is in a highlighted area. If the cursor is not located in the highlighted area, this special name references the image on which the cursor is located.
`"<TEXT>"`	References the object named in the highlighted text in the image in the window.
`"<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.

You can replace special names with other types of naming expressions allowable in that parameter.

Parameter Placeholders

Many of the commands in this package have a parameter placeholder of the form ">>*name*<<", where *name* is the type of object that should replace ">>*name*<<".

Parameter placeholders must be replaced by the name of an object, as specified by their type. Executing a command containing a parameter placeholder results in an error.

procedure Copy

```
procedure Copy (Log_File    : Name               := "<IMAGE>";
                Destination : Name               := "";
                Filter      : Profile.Log_Filter := Profile.Filter);
```

Description

Filters the specified log file using the filter supplied and then displays or writes the specified destination file.

The procedure both copies the log from one file into another and filters the log in the process. The filter can define any subset of messages in the log. By default, the procedure filters the specified log file and writes it into the current log file using the filter that is part of the job profile.

Parameters

`Log_File : Name := "<IMAGE>";`

Specifies the input log file. The default is the current image.

`Destination : Name := "";`

Specifies the destination for the log file. The default specifies that the file or window defined as the current log file is to be used.

`Filter : Profile.Log_Filter := Profile.Filter;`

Specifies the filter to use in filtering the log file as it is copied. The default is to use the default filter that is part of the job profile.

References

package Profile

procedure Filter

```
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);
```

Description

Filters the specified log file based on the specified parameters and then displays or writes the result into the specified destination file.

The specified kinds of messages from the log file are written into the specified destination. If the destination is explicitly named, the file is created, if necessary, and the contents are placed in the file. If the destination is the default value, the contents are placed in the file or window defined as the current log file.

The messages are filtered according to the three-character sequences that are part of the messages in logs. These character sequences are explained in the package Profile. If a message does not have one of these three-character sequences, the Filter procedure assumes that the kind of message is a note.

Parameters

```
Log_File : Name := "<IMAGE>";
```

Specifies the log file that contains the messages to be displayed or written. The default is the current image. If the name is the null string (""), the designated file is the log file that is generated by the rest of the current job.

```
Destination : Name := "";
```

Specifies the destination file into which the filtered log file is to be written. If the specified file does not exist, the procedure creates it. The default specifies that the file or window defined as the current log file is to be used.

```
Auxiliaries : Boolean := True;
```

Specifies whether or not to display or write the auxiliary messages in the destination file. The default is to display or write them.

```
Diagnostics : Boolean := True;
```

Specifies whether or not to display or write the diagnostic messages in the destination file. The default is to display or write them.

```
Notes : Boolean := True;
```

Specifies whether or not to display or write the note messages in the destination file. The default is to display or write them.

```
Positives : Boolean := True;
```

Specifies whether or not to display or write the positive messages in the destination file. The default is to display or write them.

```
Negatives : Boolean := True;
```

Specifies whether or not to display or write the negative messages in the destination file. The default is to display or write them.

```
Positions : Boolean := True;
```

Specifies whether or not to display or write the position messages in the destination file. The default is to display or write them.

```
Warnings : Boolean := True;
```

Specifies whether or not to display or write the warning messages in the destination file. The default is to display or write them.

```
Errors : Boolean := True;
```

Specifies whether or not to display or write the error messages in the destination file. The default is to display or write them.

```
Exceptions : Boolean := True;
```

Specifies whether or not to display or write the exception messages in the destination file. The default is to display or write them.

```
Sharps : Boolean := True;
```

Specifies whether or not to display or write the sharp (#) messages in the destination file. The messages denoted with the sharp sign have no implicit meaning and are generated only by user-supplied messages. The default is to display or write them.

```
Dollars : Boolean := True;
```

Specifies whether or not to display or write the dollar ($) messages in the destination file. The messages denoted with the dollar sign have no implicit meaning and are generated only by user-supplied messages. The default is to display or write them.

```
Ats : Boolean := True;
```

Specifies whether or not to display or write the *at* (@) messages in the destination file. The messages denoted with the *at* sign have no implicit meaning and are generated only by user-supplied messages. The default is to display or write them.

References

package Profile

---

renamed procedure Filter_Errors

```
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;
```

Description

Filters the specified log file based on the specified parameters and then displays or writes the result into the specified destination file.

The specified kinds of messages from the log file are written to the specified destination. If the destination is explicitly named, the file is created, if necessary, and the contents are placed in the file. If the destination is the default value, the contents are placed in the file or window defined as the current log file.

By default, the procedure filters out only the messages that generally indicate an error condition.

The messages are filtered according to the three-character sequences that are part of the messages in logs. These character sequences are explained in package Profile. If a message does not have one of these three-character sequences, the Filter_Errors procedure assumes that the kind of message is a note.

Parameters

```
Log_File : Name := "<IMAGE>";
```

Specifies the log file that contains the messages to be displayed or written. The default is the current image. If the name is the null string (""), the designated file is the log file that is generated by the rest of the current job.

```
Destination : Name := "";
```

Specifies the destination file into which the filtered log file is to be written. If the specified file does not exist, the procedure creates it. The default specifies that the file or window defined as the current log file is to be used.

```
Auxiliaries : Boolean := True;
```

Specifies whether or not to display or write the auxiliary messages in the destination file. The default is to display or write them.

```
Diagnostics : Boolean := True;
```

Specifies whether or not to display or write the diagnostic messages in the destination file. The default is to display or write them.

```
Notes : Boolean := False;
```

Specifies whether or not to display or write the note messages in the destination file. The default is not to display or write them.

```
Positives : Boolean := False;
```

Specifies whether or not to display or write the positive messages in the destination file. The default is not to display or write them.

```
Negatives : Boolean := True;
```

Specifies whether or not to display or write the negative messages in the destination file. The default is to display or write them.

```
Positions : Boolean := False;
```

Specifies whether or not to display or write the position messages in the destination file. The default is not to display or write them.

```
Warnings : Boolean := True;
```

Specifies whether or not to display or write the warning messages in the destination file. The default is to display or write them.

```
Errors : Boolean := True;
```

Specifies whether or not to display or write the error messages in the destination file. The default is to display or write them.

```
Exceptions : Boolean := False;
```

Specifies whether or not to display or write the exception messages in the destination file. The default is not to display or write them.

```
Sharps : Boolean := False;
```

Specifies whether or not to display or write the sharp (#) messages in the destination file. The messages denoted with the sharp sign have no implicit meaning and are generated only by user-supplied messages. The default is not to display or write them.

```
Dollars : Boolean := False;
```

Specifies whether or not to display or write the dollar ($) messages in the destination file. The messages denoted with the dollar sign have no implicit meaning and are generated only by user-supplied messages. The default is not to display or write them.

```
Ats : Boolean := False;
```

Specifies whether or not to display or write the *at* (@) messages in the destination file. The messages denoted with the *at* sign have no implicit meaning and are generated only by user-supplied messages. The default is not to display or write them.

References

procedure Filter

package Profile

---

procedure Flush

```
procedure Flush (Response : Profile.Response_Profile  := Profile.Get);
```

Description

Forces any buffered output into the log file.

For example, this procedure can be used to cause all buffered output to be saved permanently. This procedure is useful when synchronizing interactive activities to make certain that all buffered output has been displayed.

Parameters

```
Response :  Profile.Response_Profile  := Profile.Get;
```

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the job response profile.

References

package Profile

subtype Name

```
subtype Name is String;
```

Description

Defines a name for a log file.

Strings of this type can use all of the special names, wildcards, context prefixes, and attributes that are defined for naming objects in the Environment. Strings of this type, however, must be unambiguous; each can resolve only to a single object. For more information, see LM, Key Concepts.

References

procedure Copy

procedure Filter

procedure Filter_Errors

procedure Set_Error

procedure Set_Input

procedure Set_Log

procedure Set_Output

procedure Summarize

renamed procedure Pop_Error

```
procedure Pop_Error renames Io.Pop_Error;
```

Description

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

The procedure sets the error file to be a file value previously saved by a Set_Error procedure. The current value of the error file is not closed, so it can be used again with another Set_Error procedure.

References

procedure Reset_Error

procedure Set_Error

TIO, procedure Io.Pop_Error

TIO, procedure Io.Set_Error

renamed procedure Pop_Input

```
procedure Pop_Input renames Io.Pop_Input;
```

Description

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

The procedure sets the input file to be a file value previously saved by a Set_Input procedure. The current value of the input file is not closed, so it can be used again with another Set_Input procedure.

References

procedure Reset_Input

procedure Set_Input

TIO, procedure Io.Pop_Input

TIO, procedure Io.Set_Input

renamed procedure Pop_Output

```
procedure Pop_Output renames Io.Pop_Output;
```

Description

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

The procedure sets the output file to be a file value previously saved by a Set_Output procedure. The current value of the output file is not closed, so it can be used again via another Set_Output procedure.

References

procedure Reset_Output

procedure Set_Output

TIO, procedure Io.Pop_Output

TIO, procedure Io.Set_Output

procedure Put_Condition

```
procedure Put_Condition (Status   : Simple_Status.Condition;
                         Response : Profile.Response_Profile  :=
                                                    Profile.Get);
```

Description

Converts the value of the Status parameter to a textual representation and writes it into the log file.

Many commands in the Environment return a Simple_Status.Condition (PT) as an indication of whether the operation succeeded or failed. This procedure enables the user to write the results returned by such a status parameter in a readable format into the current log file.

Parameters

`Status :  Simple_Status.Condition;`

Specifies the status value that is into be written to the log file.

`Response :  Profile.Response_Profile  := Profile.Get;`

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the job response profile.

References

package Profile

PT, package Simple_Status

procedure Put_Job_Messages

```
procedure Put_Job_Messages (For_Job  : Machine.Job_Id;
                            Response : Profile.Response_Profile :=
                                                    Profile.Get);
```

Description

Formats and writes the contents of the system message log for the specified job into the current log file.

Parameters

`For_Job : Machine.Job_Id;`

Specifies the job whose messages are to be displayed.

`Response : Profile.Response_Profile := Profile.Get;`

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the job response profile.

References

package Profile

procedure Put_Line

```
procedure Put_Line (Message  : String;
                    Kind     : Profile.Msg_Kind          := Profile.Note_Msg;
                    Response : Profile.Response_Profile  := Profile.Get);
```

Description

Formats the specified message according to the setting of the prefixes and line width portions of the specified profile and, if the filter in the specified profile allows messages of the specified kind, writes the formatted message into the current log file.

This procedure allows additional messages to be placed in the log file beyond any supplied by the Environment. These additional messages can indicate progress within a sequence of commands or status information about the job.

This procedure produces different results based on the values of several other operations in this package.

If Profile.Includes (Kind) is false, the Put_Line call returns immediately. If Profile.Includes (Kind) is true, the message is generated as described below:

- The time, date, and symbols prefixes are printed in the order and format specified by the Profile.Prefixes (Response) array.
- If the symbols prefix is requested, a unique three-character string is generated for each possible value of Profile.Includes (Kind).
- 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 in place of the symbols described above.

Parameters

```
Message :  String;
```

Specifies the message to be placed in the log file. This procedure places the appropriate prefixes for the message kind in the log file before this string.

```
Kind :  Profile.Msg_Kind := Profile.Note_Msg;
```

Specifies the kind of message to be put into the log file. The default is the message kind for general commentary.

```
Response :  Profile.Response_Profile := Profile.Get;
```

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the job response profile.

References

package Profile

---

procedure Put_System_Messages

```
procedure Put_System_Messages (Response : Profile.Response_Profile :=
                                              Profile.Get);
```

Description

Puts the contents of the system message log for the current job into the current log file.

Parameters

```
Response : Profile.Response_Profile := Profile.Get;
```

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the job response profile.

References

package Profile

renamed procedure Reset_Error

```
procedure Reset_Error renames Io.Reset_Error;
```

Description

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

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

References

procedure Pop_Error

procedure Set_Error

TIO, procedure Io.Reset_Error

renamed procedure Reset_Input

```
procedure Reset_Input renames Io.Reset_Input;
```

Description

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

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

References

procedure Pop_Input

procedure Set_Input

TIO, procedure Io.Reset_Input

procedure Reset_Log

```
procedure Reset_Log (Filter : Profile.Log_Filter := Profile.Filter);
```

Description

Closes the current output file, pops it off the stack of output files, and sets the default filter to the specified value.

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

Parameters

```
Filter : Profile.Log_Filter := Profile.Filter;
```

Specifies the new value of the default filter. The default is the filter defined as part of the current profile.

References

procedure Set_Log

package Profile

renamed procedure Reset_Output

```
procedure Reset_Output renames Io.Reset_Output;
```

Description

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

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

References

procedure Pop_Output

procedure Set_Output

TIO, procedure Io.Reset_Output

procedure Save

```
procedure Save (Response: Profile.Response_Profile := Profile.Get);
```

Description

Forces any buffered output into the log file and then makes the current contents of the log file permanent.

A log file does not become permanent until it is closed. This procedure first flushes all buffered output into the file and then effectively performs the same function without actually closing it.

This procedure should not be used to save the contents of an I/O window.

Parameters

```
Response : Profile.Response_Profile := Profile.Get;
```

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the job response profile.

References

package Profile

renamed procedure Set_Error

```
procedure Set_Error (To_Be : Name := ">>FILE NAME<<") renames Io.Set_Error;
```

Description

Sets the current error file to the specified file and pushes any previous values of error files onto the stack of error files.

Parameters

`To_Be : Name := ">>FILE NAME<<";`

Specifies the filename to be set to the current error. The default parameter placeholder `">>FILE NAME<<"` must be replaced or an error will result.

References

procedure Pop_Error

procedure Reset_Error

TIO, procedure Io.Set_Error

renamed procedure Set_Input

```
procedure Set_Input (To_Be : Name := "<REGION>") renames Io.Set_Input;
```

Description

Sets the current input file to the specified file and pushes any previous values of input files onto the stack of input files.

Parameters

```
To_Be : Name := "<REGION>";
```

Specifies the filename to be set to the current input. The default is the currently highlighted object.

References

procedure Pop_Input

procedure Reset_Input

TIO, procedure Io.Set_Input

procedure Set_Log

```
procedure Set_Log (To_Be  : Name                := ">>FILE NAME<<";
                   Filter : Profile.Log_Filter  := Profile.Filter);
```

Description

Sets the current output file to the specified file, pushes any previous values of output files onto the stack of output files, and sets the filter to the specified value.

Parameters

`To_Be : Name := ">>FILE NAME<<";`

Specifies the filename to be set to the current error. The default parameter placeholder ">>FILE NAME<<" must be replaced or an error will result.

`Filter : Profile.Log_Filter := Profile.Filter;`

Specifies the new value of the default filter. The default is the filter defined as part of the current profile.

References

package Profile

renamed procedure Set_Output

```
procedure Set_Output (To_Be : Name := ">>FILE NAME<<")
                                                renames Io.Set_Output;
```

Description

Sets the current output file to the specified file and pushes any previous values of output files onto the stack of output files.

Parameters

`To_Be : Name := ">>FILE NAME<<";`

Specifies the filename to be set to the current output. The default parameter placeholder `">>FILE NAME<<"` must be replaced or an error will result.

References

procedure Pop_Output

procedure Reset_Output

TIO, procedure Io.Set_Output

renamed procedure Summarize

```
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;
```

Description

Filters the specified log file based on the specified parameters and then displays or writes the result in the specified destination file.

The specified kinds of messages from the log file are written into the specified destination. If the destination is explicitly named, the file is created, if necessary, and the contents are placed in the file. If the destination is the default value, the contents are placed in the file or window defined by the current log file.

By default, the procedure filters out only the messages that generally summarize progress through major steps of the job.

The messages are filtered according to the three-character sequences that are part of the messages in logs. These character sequences are explained in package Profile. If a message does not have one of these three-character sequences, the Summarize procedure assumes that the kind of message is a note.

Parameters

`Log_File : Name := "<IMAGE>";`

Specifies the log file that contains the messages to be displayed or written. The default is the current image. If the name is the null string (""), the designated file is the log file that is generated by the rest of the current job.

```
Destination : Name := "";
```

Specifies the destination file into which to write the messages. If the specified file does not exist, the procedure creates it. The default specifies that the file or window defined by the current log file is to be used.

```
Auxiliaries : Boolean := True;
```

Specifies whether or not to display or write the auxiliary messages in the destination file. The default is to display or write them.

```
Diagnostics : Boolean := True;
```

Specifies whether or not to display or write the diagnostic messages in the destination file. The default is to display or write them.

```
Notes : Boolean := False;
```

Specifies whether or not to display or write the note messages in the destination file. The default is not to display or write them.

```
Positives : Boolean := True;
```

Specifies whether or not to display or write the positive messages in the destination file. The default is to display or write them.

```
Negatives : Boolean := True;
```

Specifies whether or not to display or write the negative messages in the destination file. The default is to display or write them.

```
Positions : Boolean := False;
```

Specifies whether or not to display or write the position messages in the destination file. The default is not to display or write them.

```
Warnings : Boolean := False;
```

Specifies whether or not to display or write the warning messages in the destination file. The default is not to display or write them.

```
Errors : Boolean := False;
```

Specifies whether or not to display or write the error messages in the destination file. The default is not to display or write them.

```
Exceptions : Boolean := False;
```

Specifies whether or not to display or write the exception messages in the destination file. The default is not to display or write them.

```
Sharps : Boolean := False;
```

Specifies whether or not to display or write the sharp (#) messages in the destination file. The messages denoted with the sharp sign have no implicit meaning and are generated only by user-supplied messages. The default is not to display or write them.

```
Dollars : Boolean := False;
```

Specifies whether or not to display or write the dollar ($) messages in the destination file. The messages denoted with the dollar sign have no implicit meaning and are generated only by user-supplied messages. The default is not to display or write them.

```
Ats : Boolean := False;
```

Specifies whether or not to display or write the *at* (@) messages in the destination file. The messages denoted with the *at* sign have no implicit meaning and are only generated by user supplied messages. The default is not to display or write them.

References

procedure Filter

---

end Log;

RATIONAL

package Operator

Package Operator contains operations that enable system managers to create users, manage groups for access control, and enable terminals for login. Documented here are procedures useful to all users, including:

- Changing your password
- Determining how much disk space is available
- Displaying access control group membership
- Logging off sessions

For information on the other commands in package Operator, see the SMU book of the *Rational Environment Reference Manual.*

procedure Change_Password

```
procedure Change_Password (User         : String := ">>USER NAME<<";
                           Old_Password : String := "";
                           New_Password : String := "";
                           Response     : String := "<PROFILE>");
```

Description

Replaces the Old_Password parameter with the New_Password parameter for the specified username.

Parameters

`User : String := ">>USER NAME<<";`

Specifies the username. The default parameter placeholder ">>USER NAME<<" must be replaced or an error will result.

`Old_Password : String := "";`

Specifies the old password. If the old password is not known, the operator's password can be used—that is, the password for the username Operator. The default is the null string—that is, no password.

`New_Password : String := "";`

Specifies the new password. The default is the null string—that is, no password.

`Response : String := "<PROFILE>";`

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the job response profile.

Example

The command:

```
    operator.change_password ("anderson","hello","greetings");
```

changes the password for user Anderson from `hello` to `greetings`.

procedure Disk_Space

```
procedure Disk_Space;
```

Description

Displays disk data.

Example

The command:

```
operator.disk_space
```

returns a display such as the following:

```
Volume  Capacity  Available   Used   % Free
======  ========  =========  ======  ======
1        369120    284196     84924   76
2        391680    229239    162441   58
3        391680    274570    117110   70
4        391680    282619    118661   70

Total   1553760   1070624    483136   68
```

Volume indicates the disk drive, Capacity and Available describe disk space in terms of pages of 1 Kb each, Used describes the amount of disk space used in terms of pages of 1 Kb each, and % Free specifies the amount of disk space that is still unused in terms of percentages.

The bottom row describes totals for all volumes.

procedure Display_Group

```
procedure Display_Group (Group    : String := ">>GROUP NAME<<";
                         Response : String := "<PROFILE>");
```

Description

Displays the usernames that are current members of the specified group on Current_Output.

If there is a user with the name, the procedure displays the groups of which the user is a member.

Parameters

`Group : String := ">>GROUP NAME<<";`

Specifies the name of the group. The default parameter placeholder ">>GROUP NAME<<" must be replaced or an error will result.

`Response : String := "<PROFILE>";`

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the job response profile.

Example

The command:

```
    operator.display_group (group=>"public");
```

results in the following display:

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

!USERS.GZC % OPERATOR.DISPLAY_GROUP                     STARTED 6:20:29 PM
--------------------------------------------------------------------------

Contents of group "public"
==========================
    user BILL
    user BLB
    user GZC
    user JIM
    user JMK
    user PUBLIC
    user SMP

User "public" is a member of
============================
   group PUBLIC
```

procedure Force_Logoff

```
procedure Force_Logoff (Physical_Line  : Terminal.Port;
                        Commit_Buffers : Boolean        := True;
                        Response       : String         := "<PROFILE>");
```

Description

Terminates any session active on the specified line.

Uncommitted changes to images are saved if the Commit_Buffers parameter is true. The user's background jobs (if any) continue to run, and any foreground jobs that do not require interactive input are put in the background. Foreground jobs that attempt interactive input are killed.

Execution of this procedure requires that the executing job have operator capability for sessions other than those belonging to your current username.

Parameters

```
Physical_Line : Terminal.Port;
```

Specifies that Terminal.Port is a number from 0 through 255. You can determine which port by executing the What.Users procedure.

```
Commit_Buffers : Boolean := True;
```

Specifies whether uncommitted changes the user has made to any images will be committed. The changes are saved only when true.

```
Response : String := "<PROFILE>";
```

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the job response profile.

Example

The command:

```
    operator.force_logoff (18);
```

logs off the user currently on port 18. Any uncommitted changes to images are saved.

end Operator;

RATIONAL

package Profile

This package provides facilities for tailoring *attributes* that affect the behavior of many commands in the Environment. These attributes affect:

- Log generation
- Error reactions
- Activities
- Remote sessions
- Remote passwords

Each profile has eight attributes, each of which can be defined explicitly or use default values:

- Error_Reaction specifies which of four options a command is to follow in reacting to error situations. More specifically, the attribute specifies how a command is to respond to errors along two dimensions: perseverance and exception propagation. Each of the four options is defined to react in different combinations of reactions along these two dimensions.
- Log_Filter specifies, in some detail, the desired content of the log that is generated by the command.
- Log_Prefixes specifies the format of messages entered into the log.
- Width specifies the number of columns in the log display.
- Log_File specifies the predefined file variable to be used by package Log to generate output.
- Activity specifies the activity file used for loading subsystems.
- Remote_Sessions specifies the file used for sessions used in networking operations.
- Remote_Passwords specifies the file used for passwords used in networking operations.

This package provides operations for setting and retrieving each of these eight attributes on a job or session basis, for those users who want to tailor these attributes to meet their own specific requirements.

For further information on the prefixes used in generating log output, see the Msg_Kind type.

Overview of Profiles

Profiles tell the Environment how to respond to errors, how to generate logs, which activity to use, which remote passwords file to use, and which remote session file to use. Profiles can be converted from their internal representation to a string representation. The string representation of a profile looks similar to:

```
PERSEVERE, :::, ~???, ---..$$$, Prefix => YR_MN_DY HR_MN_SC SYMBOLS,
Width => 77, USE_OUTPUT, Activity => <NIL>, Remote_Passwords => <NIL>,
Remote_Sessions => <NIL>
```

The first field, PERSEVERE, is the Error_Reaction. This tells the Environment what to do when it encounters an error. There are four possible error reactions: QUIT, PROPAGATE, PERSEVERE, and RAISE_ERROR. For further information on error reactions, see the Error_Reaction type.

The second field, consisting of the symbols :::, ~???, ___ .. $$$, tells the Environment which kind of messages it should display in the log. The tilde (~) in front of a message type indicates that it should not be displayed. For further information on types of messages, see the Msg_Kind type.

The third field, Prefix => YR_MN_DY HR_MN_SC SYMBOLS, specifies which prefixes the Environment should use for messages that are added to log files. For further information on prefixes, see the Log_Prefix type.

The fourth field, Width => 77, specifies how many characters wide log messages can be before they are wrapped. For further information, see the Default_Width constant.

The fifth field, USE_OUTPUT, specifies where log output should be placed. For further information, see the Log_Output_File type.

The sixth field, Activity => <NIL>, specifies which activity should be used. Activities are a component of subsystems. For further information on subsystems, see the Project Management (PM) book of the *Rational Environment Reference Manual*.

The seventh field, Remote_Passwords => <NIL>, specifies which remote passwords file is to be used. Remote password files are described in more detail below.

The eighth field, Remote_Sessions => <NIL>, specifies which remote sessions file is to be used. Remote sessions files are described in more detail below.

The Environment provides for three types of response profiles: a session profile, a per-job response profile, and the system default profile. These profiles and their uses are described below.

System Default Profile

The system default profile is specified by the *special value* "<DEFAULT>". This profile is used in situations requiring a profile where the user has not set up a session profile. This is the profile returned by the Default_Profile function. The default values for the default profile are:

- Error reaction: The command should persevere. For further information on error reactions, see the Error_Reaction type.
- Type of messages to be put in the log file or filtered out: Auxiliary, note, positive, position, negative, warning, error, exception, sharp, at, and dollar. For further information on types of messages, see the Msg_Kind type.
- Prefixes to be put in the log file: Yr_Mn_Dy, Hr_Mn_Sc, and Symbols indicating the type of prefixes put in the log, as described above. For further information, see the Log_Prefix and Msg_Kind types.
- Width of log messages before they are wrapped: The width is 77 characters. Messages that have been wrapped are prefixed with the ellipsis (...).
- Where log output should be placed: Specifies that log output should be placed in the current output window.
- Which activity to use: Specifies that none should be used.
- Which remote passwords file to use: Specifies that none should be used.
- Which remote sessions file to use: Specifies that none should be used.

Session Profile

The Environment session has its own permanent profile, which users can modify to suit their own needs. This session profile is specified in Environment commands using the special value "<SESSION_PROFILE>".

Job Response Profile

Each job has its own profile, which is initially the same as the session profile, but can be modified by the job. The per-job response profile is specified by the special value "<PROFILE>".

Setting the Profile to a Predefined Response Profile

The following ten special values can be used to set the profile to a predefined response profile:

"<IGNORE>"	Specifies a response profile that ignores all log messages. The error reaction is to persevere.
"<ERRORS>"	Specifies that only error messages are logged (that is, negative messages, error messages, and exception messages). The error reaction is to persevere.
"<NIL>"	Specifies a response profile that ignores all log messages. The error reaction is to quit.

"<PROGRESS>"	Specifies that positive messages, error messages, negative messages, and exception messages are logged. The error reaction is to persevere.
"<QUIET>"	Specifies that no messages are to be logged. The error reaction is to quit.
"<RAISE_EXCEPTION>"	Specifies a response profile that forces commands to raise an exception when an error occurs. No error messages are logged. The error reaction is to quit.
"<SESSION>"	Specifies the response profile for the current session (same as "<SESSION_PROFILE>").
"<VERBOSE>"	Specifies a response profile that puts all messages except debug messages in the log file. The error reaction is to persevere.
"<WARN>"	Specifies a response profile that puts negative messages, warning messages, error messages, and exception message in the log. The error reaction is to persevere.
"<WARNINGS>"	Same as <WARN>.

Networking and Package Profile

Package Profile provides two facilities for facilitating networking between one R1000 and another, as well as between an R1000 and another machine. These facilities, called ***remote passwords*** and ***remote sessions***, are described below.

Remote Passwords

Package Profile can also be used to facilitate networking operations. Networking operations, for access control purposes, require a username and password. Package Profile allows the user to specify, in a session or job response profile, a ***remote passwords file*** that contains the information required by networking operations to give the user access to remote machines. This enables the user to perform networking operations without the necessity of supplying usernames and passwords.

The remote passwords file must be a text file, using the following format:

machine name   *username*   *password or* <PROMPT>

The first column in the file consists of the machine names to which a user may have access. The word "Others" in this column is reserved to mean all other machines to which a user may have access, in the event that the user has identical usernames and passwords on two or more machines and does not want to specify each of them individually. If used, "Others" should be on the last line in the file. The second column consists of the username by which the user is known on the machine on the same line in the first column. Finally, the third column consists of the password for the username and machine specified on the same line in the first two columns. If the user does not want to put a list of passwords to different machines in a file, the user can use the <PROMPT> symbol in the third column. This causes the networking operation to prompt the user for the password. The <PROMPT> symbol can also be used in the username column. Note that the word PROMPT must be in uppercase.

The following is a sample situation illustrating the use of a remote passwords file. User John has accounts on five machines (called Machine_1 through Machine_5) that are connected to the same network. The usernames on Machine_3 and Machine_4 have the same password, which is My_Password!. The security on Machine_5 is such that John does not want his password in a file, preferring to be prompted for it. To use the remote passwords facility, John would build a text file called !Users.John.Secret_Stuff that appears like this:

```
Machine_1    John    Snarf_!#%
Machine_2    John    Splork#now@$
Machine_5    John    <PROMPT>
Others       John    My_Password!
```

He could then update the Remote_Passwords switch in his session switch file to point to this file. In the future, when performing networking operations that use his session switch file, the contents of this remote passwords file would be used, and he would no longer have to supply usernames and passwords (except on Machine_5).

Similarly, John could set up other remote password files using other identities on other machines or other sessions for jobs he runs with their own profiles.

This package provides a subtype and four operations that can be used to manipulate the remote passwords file, as follows:

- Default_Remote_Passwords function
- Remote_Passwords function
- Remote_Passwords_Type subtype
- Set_Default_Remote_Passwords procedure
- Set_Remote_Passwords procedure

Hints for Facilitating Use of Remote Passwords Files

The following list illustrates how remote password files can be used, starting from open shops through highly secure situations:

- In an open shop, servers can be initialized with an identity that allows them to access any object on their machine. Users therefore do not need remote password files, and no usernames or passwords are used in networking. In this situation, any user can access any object.
- If a user uses the same username and password on all machines, the file could consist of a single line with the machine name of "Other" and his username and password.
- If a group of users on a given machine have the same remote privileges, they can share the same password file.
- If a user frequently logs into several machines, the remote password files on those machines can be copies of each other. The remote password file can contain an entry for the machine on which it resides.

- If usernames and passwords are to be kept secret, then the remote password files are protected by access control. For maximum security, each user has his own remote password file, protected so that only he can read it. If still further access control is desired, the user can use the <PROMPT> symbol in either or both the username and password fields of the file.

Remote Sessions

As well as having a remote passwords file, the user can have a remote sessions file that specifies to the Environment which session should be used when logging onto another machine. For the purpose of this discussion, sessions and accounts on remote machines are considered to be the same thing.

The remote sessions file must be a text file, using the following format:

machine name *session name*

The first column in the file consists of the machine names to which a user may have access. The word "Others" in this column is reserved to mean all other machines to which a user may have access, in the event that the user may have identical session names on two or more machines and does not want to specify each of them individually. If used, "Others" should be on the last line of the file. The second column consists of the session name the user wants to use when logging into another machine. The user can use the <PROMPT> symbol in the second column to be prompted for the session name.

The remote sessions file is used in conjunction with the remote passwords file, so that username John mentioned in the example above can specify which session he wants to use when using remote passwords to log into another machine.

Once the file is set up, and the user performs an FTP operation, the system makes the following checks:

1. It checks your session switch file for the value of Session_Ftp.Account. This switch can be used to specify an account name. If a value is specified, the system uses that account when logging onto the other machine.
2. If the value of Session_Ftp.Account is null, the system then looks at the Session_Ftp.Prompt_For_Account switch. If it is set to true, the system prompts for an account name.
3. If the value of Session_Ftp.Account is set to false, the system then looks at the filename specified in the Profile.Remote_Sessions switch. If a filename is specified, the system uses the session name specified in the Remote_Sessions file for that machine as the account name.

To show the use of a remote sessions file, consider the previous example of John, who has the following remote passwords file:

```
Machine_1    John   Snarf_!#%
Machine_2    John   Splork#now@$
Machine_5    John   <PROMPT>
Others       John   My_Password!
```

John's remote sessions file looks like this:

```
    Machine_1      S_1
    Machine_2      System_Manager
    Machine_5      Engineering
    Others         S_2
```

Assuming that John has set his session switches so that he will be using a remote sessions file, when John performs an FTP operation on Machine_1, he will be logged in under session S_1. On Machine_2, his session name will be System_Manager. On Machine_5, his session name will be Engineering, and on all other machines, it will be session S_2.

function Activity

```
function Activity (Response : Response_Profile := Profile.Get)
                                         return Activity_Type;
```

Description

Returns the activity that is part of the specified profile.

Activities are used in subsystems. For further information, see Project Management (PM).

Parameters

```
Response : Response_Profile := Profile.Get;
```

Specifies the profile from which the activity is desired. The default is the job response profile.

```
return Activity_Type;
```

Returns the activity from the specified profile. For further information, see Project Management (PM).

References

Project Management (PM)

subtype Activity_Type

```
subtype Activity_Type is Directory.Object;
```

Description

Defines a representation for an activity.

For further information on activities, see Project Management (PM).

function Cached_Selected_Text

```
function Cached_Selected_Text return String;
```

Description

Returns a string representing the selected text as cached at job initialization.

The cursor need not be in the selection.

Parameters

```
return String;
```

Returns a string representing the selected text.

procedure Convert

```
procedure Convert (Image    :        String;
                   Response :    out Response_Profile;
                   Status   : in out Simple_Status.Condition);
```

Description

Converts between the string representation of a profile (as used in most commands) and the internal representation.

The functional version of this operation is called Value.

Parameters

`Image : String;`

Specifies the string representation of a profile.

`Response : out Response_Profile;`

Specifies the internal representation of the profile specified by the Image parameter.

`Status : in out Simple_Status.Condition;`

Returns the status indicating that the procedure has executed correctly, or, if there is an error, a message indicating the type of error.

References

function Value

procedure Log.Put_Condition

PT, package Simple_Status

function Default_Activity

```
function Default_Activity return Activity_Type;
```

Description

Returns the default activity that is part of the session response profile.

Parameters

```
return Activity_Type;
```

Returns the activity from the session response profile. For further information on activities, see Project Management (PM).

References

Project Management (PM)

constant Default_Filter

```
Default_Filter : constant Log_Filter := Full;
```

Description

Defines a constant that represents the default filter for generating log files.

If the user does not specify another profile, but uses the system default profile, this is the filter that will be used.

The default filter specifies that the following message types should be put in the log file: auxiliary, note, positive, position, negative, warning, error, exception, sharp, at, and dollar. For further information on types of messages, see the Msg_Kind type.

References

function Default_Profile

type Msg_Kind

constant Default_Log_File

```
Default_Log_File : constant Log_Output_File := Use_Output;
```

Description

Defines a constant that specifies that output should be placed in Current_Output.

This is the log file that is used if a different log file is not specified in the session or job response profile.

References

function Default_Profile

type Log_Prefix

type Msg_Kind

constant Default_Prefixes

```
Default_Prefixes : constant Log_Prefixes := (Yr_Mn_Dy, Hr_Mn_Sc, Symbols);
```

Description

Defines a constant that represents the default set of log message prefixes used when messages are inserted into log files.

These prefixes are used if different prefixes are not specified in the session or job response profile.

The default prefixes are Yr_Mn_Dy, Hr_Mn_Sc, and Symbols. For further information, see the Log_Prefix and Msg_Kind types.

References

function Default_Profile

type Log_Prefix

type Msg_Kind

function Default_Profile

```
function Default_Profile return Response_Profile;
```

Description

Returns the system's default response profile in the internal representation of a response profile.

The default response profile, used if a user has not established his or her own response profile, is the aggregate of all of the default constants defined in this package.

This is the profile that is used in situations requiring a profile where the user has not set up a session or job response profile.

The default values for the default response profile are as follows:

- Error reaction: The command should persevere. For further information on error reactions, see the Error_Reaction type.
- Type of messages to be put in the log file or filtered out: Auxiliary, note, positive, position, negative, warning, error, exception, sharp, at, and dollar. For further information on types of messages, see the Msg_Kind type.
- Prefixes to be put in the log file: Yr_Mn_Dy, Hr_Mn_Sc, and symbols indicating the type of message put in the log. For further information, see the Log_Prefix and Msg_Kind types.
- Width of log messages before they are wrapped: The width is 77 characters. Messages that have been wrapped are prefixed with the ellipsis (...) symbol.
- Where log output should be placed: Specifies that log output should be placed in the current output window.
- Which activity to use: Specifies that none should be used.
- Which remote passwords file to use: Specifies that none should be used.
- Which remote sessions file to use: Specifies that none should be used.

References

constant Default_Width

type Error_Reaction

type Log_Prefix

type Msg_Kind

introduction to this package

constant Default_Reaction

```
Default_Reaction : constant Error_Reaction := Persevere;
```

Description

Defines a constant that represents the default reaction to errors.

This is the reaction to errors that is used if a different reaction is not specified in the session or job response profile.

The reaction is to persevere, which means that, if an error is encountered, the command will continue to execute.

References

function Default_Profile

function Default_Remote_Passwords

```
function Default_Remote_Passwords  return Remote_Passwords_Type;
```

Description

Returns the remote passwords file for the current session.

For further information, see the introduction to this package.

Parameters

```
return Remote_Passwords_Type;
```

Returns the remote passwords file for the current session. The default is none.

References

function Default_Profile

function Default_Remote_Sessions

```
function Default_Remote_Sessions return Remote_Sessions_Type;
```

Description

Returns the remote sessions file for the current session.

For further information, see the introduction to this package.

Parameters

```
return Remote_Sessions_Type;
```

Returns the remote sessions file for the current session. The default is none.

References

function Default_Profile

constant Default_Width

```
Default_Width : constant Natural := 77;
```

Description

Defines a constant that represents the default width of the log file.

The width is used when messages are inserted into the log files. If the message is longer than the defined width in the profile, the message is wrapped.

If the user has not specified a different width in the session profile or the job response profile, a width of 77 is used.

References

function Default_Profile

function Width

exception Error

```
Error : exception;
```

Description

Defines the exception raised when an error condition occurs in a command and the error reaction is defined to be Raise_Error.

constant Errors

```
Errors : constant Log_Filter := Log_Filter'(Negative_Msg ..
                                   Exception_Msg => True, others => False);
```

Description

Defines a constant filter that allows only error messages to be logged.

The constant can be used in procedures, such as the Set_Response procedure, to specify that only error messages should appear in the log file.

These messages include negative, warning, error, and exception messages.

References

type Msg_Kind

type Error_Reaction

```
type Error_Reaction is (Quit, Propagate, Persevere, Raise_Error);
```

Description

Defines the four different reactions a command can have to an error.

Commands respond, based on the specified error reaction, along two dimensions:

- Perseverance: The first dimension determines whether to continue processing or to 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 propagation: The second dimension determines whether to propagate an exception to its caller when it terminates a run in which errors have occurred. If processing has persevered, the Error exception is raised. Otherwise, an exception related to the error that caused processing to stop is raised.

The enumeration defines a set of reactions to errors based on these two dimensions. The four reactions can allow the command that has an error to continue executing or to terminate, and to raise or not raise exceptions. The Error exception is raised if no other exception is defined for the command that has the error.

Enumerations

`Persevere`

Specifies that the command continue executing as best it can after errors are discovered. No exception is raised. Some error messages may have been accumulated for the job, which can be displayed by the Log.Put_System_Messages procedure.

`Propagate`

Specifies that the command terminate upon its first error. An exception is raised. If no other exception is defined for the command, the Error exception is raised. Some error messages may have been accumulated for the job, which can be displayed by the Log.Put_System_Messages procedure.

`Quit`

Specifies that the command terminate upon its first error. No exception is raised.

Raise_Error

Specifies that the command continue executing as best it can after errors are discovered. At some point, an exception is raised. If no other exception is defined for the command, the Error exception is raised. Some error messages may have been accumulated for the job, which can be displayed by the Log.Put_System_Messages procedure.

---

function Filter

```
function Filter (Response : Response_Profile := Profile.Get)
                                                  return Log_Filter;
```

Description

Returns the filter that is part of the specified profile.

Parameters

```
Response : Response_Profile := Profile.Get;
```

Specifies the profile from which the filter is desired. The default is the job response profile.

```
return Log_Filter;
```

Returns the filter from the specified profile.

constant Full

```
Full : constant Log_Filter := Log_Filter'(Debug_Msg => False,
                                          others => True);
```

Description

Defines a constant that represents a filter that allows all messages except debugging messages to be logged.

The constant can be used in procedures, such as the Set_Response procedure, to specify that all messages except debugging messages should appear in the log file.

function Get

```
function Get return String;
```

Description

Returns the string representation of the current response profile for the current job. If a profile for the job has not been defined, the session response profile is returned.

Many Environment commands require the string representation of a profile. This function provides a method for retrieving this string representation.

Parameters

```
return String;
```

Returns the string representation of the response profile. If no job response profile has been explicitly defined, the session response profile is returned.

References

procedure Set

procedure Get_Cached_Resolution

```
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);
```

Description

Retrieves the resolution of Name as cached at job initialization.

The resolution of the special names "<IMAGE>", "<CURSOR>", "<REGION>", and "<SELECTION>" is cached.

Parameters

`Name : String;`

Specifies the name of the object.

`The_Declaration : out Directory.Declaration;`

Returns the directory declaration.

`The_Object : out Directory.Object;`

Returns the directory object.

`The_Version : out Directory.Version;`

Returns the directory version.

`Status : out Directory.Naming.Name_Status;`

Returns the name status.

function Get_Default

```
function Get_Default return Response_Profile;
```

Description

Returns the string representation of the session response profile.

The function returns the response profile for the session. If no session response profile has been defined with the Set_Default procedure or through the switch editor, this function returns the system default profile, the Default_Profile function.

Parameters

```
return Response_Profile;
```

Returns the session response profile. If no session response profile is defined, the Default_Profile function is returned.

References

function Default_Profile

procedure Set_Default

function Get_Default

```
function Get_Default return String;
```

Description

Returns the string representation of the session response profile.

If no default profile has been defined with the Set_Default procedure, this function returns the system default profile, the Default_Profile function.

Parameters

```
return String;
```

Returns the string representation of session response profile. If no session response profile is defined, the Default_Profile function is returned.

References

function Default_Profile

procedure Set_Default

renamed function Ignore

```
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;
```

Description

Returns a response profile that ignores all log messages.

The function can be used in other commands to use, or to set as the default, a profile that ignores all log messages. This profile uses the session default profile for all but the log filter. The filter used filters out all log messages so that none are added to the log file.

Parameters

`Reaction : Error_Reaction := Profile.Persevere;`

Specifies the error reaction for the profile. The default is the persevere response.

`Filter : Log_Filter := Profile.Quiet;`

Specifies the log filter for the profile. The default is the quiet filter that allows no messages to be added to the log file.

`Prefixes : Log_Prefixes := Profile.Prefixes;`

Specifies the set of log prefixes for the profile. The default is the prefixes in the current job response profile.

`Width : Natural := Profile.Width;`

Specifies the width of the log file. The default is the width in the current job response profile.

`Activity : Activity_Type := Profile.Activity;`

Specifies the activity for the profile. The default is the activity in the current job response profile.

```
Log_File :  Log_Output_File := Profile.Log_File;
```

Specifies the log file for the profile. The default is the log file specified in the current job response profile.

```
return Response_Profile;
```

Returns the internal representation of the response profile.

---

function Image

```
function Image (Profile : Response_Profile := Standard.Profile.Get)
                                                       return String;
```

Description

Returns the string representation of the response profile.

Parameters

```
Profile : Response_Profile := Standard.Profile.Get;
```

Specifies the response profile. The default is the job response profile.

```
return String;
```

Returns the string representation of the response profile.

procedure Include

```
procedure Include (Kind  : Msg_Kind;
                   Value : Boolean    := True);
```

Description

Changes the log filter in the current job response profile to include the specified kind of message.

The procedure alters the log filter in the current job response profile. The procedure can specify either to include or not to include the specified kind of message. The default is to include the specified message.

Parameters

`Kind :  Msg_Kind;`

Specifies the kind of message to include or not to include in the filter.

`Value :  Boolean := True;`

Specifies whether the specified kind of message should be included. The default is to include the message.

function Includes

```
function Includes (Kind     : Msg_Kind;
                   Response : Response_Profile  := Profile.Get)
                                                   return Boolean;
```

Description

Returns whether the specified kind of message is included in the log files generated with the specified response profile.

The function returns the value of the filter in the specified response profile for the specified kind of message—that is, whether those messages are included in any log files generated with the profile.

Parameters

`Kind : Msg_Kind;`

Specifies the kind of message to be checked.

`Response : Response_Profile := Profile.Get;`

Specifies the profile in which to check. The default is the job response profile.

`return Boolean;`

Returns true if the specified message kind is included in the log files generated with the specified profile; otherwise, the function returns false.

procedure Include_In_Default

```
procedure Include_In_Default (Kind  : Msg_Kind;
                              Value : Boolean   := True);
```

Description

Changes the log filter in the session response profile to include the specified kind of message.

The procedure alters the log filter in the session response profile. The procedure can specify either to include or not to include the specified kind of message. The default is to include the specified message.

Parameters

```
Kind :  Msg_Kind;
```

Specifies the kind of message to include or not to include in the filter.

```
Value :  Boolean := True;
```

Specifies whether the specified kind of message should be included. The default, true, is to include the message.

function Log_File

```
function Log_File (Response : Response_Profile := Profile.Get)
                                                   return Log_Output_File;
```

Description

Returns the log output file, which is where log output is directed.

Parameters

```
Response :  Response.Profile := Profile.Get;
```

Defines the response profile in which to check. The default is the job response profile.

```
return Log_Output_File;
```

Returns the log output file.

References

type Log_Output_File

type Log_Filter

```
type Log_Filter is array (Msg_Kind) of Boolean;
```

Description

Defines the log file filters.

The type defines the filters that are used to generate/filter the log files. There is a Boolean value for each kind of message that can be produced for log files. This value determines whether that kind of message is included (true) in the log file or filtered (not included) from the log file (false).

type Log_Output_File

```
type Log_Output_File is (Use_Output, Use_Error, Use_Standard_Output,
                                                 Use_Standard_Error);
```

Description

Defines the four different places where log output can be directed.

Enumerations

`Use_Error`
Specifies that log output should be placed in Current_Error.

`Use_Output`
Specifies that log output should be placed in Current_Output.

`Use_Standard_Error`
Specifies that log output should be placed in Standard_Error.

`Use_Standard_Output`
Specifies that log output should be placed in Standard_Output.

type Log_Prefix

```
type Log_Prefix is (Nil, Time, Hr_Mn_Sc, Hr_Mn, Date, Mn_Dy_Yr,
                                            Dy_Mon_Yr, Yr_Mn_Dy, Symbols);
```

Description

Defines the kinds of prefixes that can be used as prefixes to messages that are added to log files.

Enumerations

`Date`

Specifies a verbose format for the date: the spelled-out month, the day of the month, and the year, such as June 30, 1987.

`Dy_Mon_Yr`

Specifies a short format for the date: the day of the month, an abbreviation of the month, and the year, such as 30-JUN-87.

`Hr_Mn`

Specifies a military format for time: hours and minutes, such as 21:08.

`Hr_Mn_Sc`

Specifies a military format for time: hours, minutes, and seconds, such as 19:54:04.

`Mn_Dy_Yr`

Specifies a short format for the date: the numeric month, the day of the month, and the year, such as 06/30/87.

`Nil`

Specifies an empty prefix. If any of the three prefixes are not used, the enumeration should be set to this prefix.

`Symbols`

Specifies the symbols associated with the kind of message. These symbols denote the kind of message being added to the log file, such as *** or !!!. See the Msg_Kind type for the set of all message kinds and their symbols.

`Time`

Specifies the normal A.M./P.M. format for time: hours, minutes, and seconds, such as 11:34:32 AM or 8:19:09 PM.

`Yr_Mn_Dy`

Specifies a short format for the date: the year, the numeric month, and the day of the month, such as 87/06/30.

References

type Msg_Kind

---

type Log_Prefixes

```
type Log_Prefixes is array (1 .. 3) of Log_Prefix;
```

Description

Defines the set of prefixes for each log message.

The type defines the set of three message prefixes. Each prefix is inserted in order at the beginning of each message, which is added to the log file. Up to three prefixes are allowed. Less than three prefixes can be used by setting the extra prefix or prefixes to the Nil enumeration.

References

type Msg_Kind

type Msg_Kind

```
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);
```

Description

Defines the various kinds of messages that can be added to the log files.

These kinds of messages can be denoted with symbols. Each symbol, a three-character sequence, can be added to the message, along with date and time information via prefixes.

Enumerations

`At_Msg`

Specifies a message that has no predefined meaning. This kind of message is not used by the Environment but can be used by users to insert messages into the log files for user-defined purposes. This message is denoted by @@@.

`Auxiliary_Msg`

Specifies a general message. This kind of message is provided by the Environment as commentary on the execution of some commands. This message is denoted by :::.

`Debug_Msg`

Specifies a debugging message. This kind of message is provided by the Environment as an aid in debugging some commands. This message is denoted by ???.

`Dollar_Msg`

Specifies a message that has no predefined meaning. This kind of message is not used by the Environment but can be used by users to insert messages into the log files for user-defined purposes. This message is denoted by **$$$**.

Error_Msg

Specifies a message indicating a serious error. This kind of message indicates that the operation has found some condition that is not correct and prevents the operation from completing successfully. This causes the error reaction to be taken by the command. A negative message may also be provided. This message is denoted by ***.

Exception_Msg

Specifies a message indicating that an exception was raised during the execution of the command. This kind of message indicates that the operation has found some condition that is not correct and prevents the operation from continuing. This causes the error reaction to be taken by the command. A negative message may also be provided. This message is denoted by %%%.

Negative_Msg

Specifies a message indicating negative progress. This kind of message is provided by the Environment when operations do not complete successfully. This causes the error reaction to be taken by the command. This message is denoted by ++*.

Note_Msg

Specifies a general message. This kind of message is provided by the Environment as commentary on the execution of some commands. This message is denoted by ---.

Position_Msg

Specifies a message indicating the position in some object where an event, explained by another message, has occurred. This kind of message is provided by the Environment as an aid in locating errors and other events in large objects. This message is denoted by >>>.

Positive_Msg

Specifies a message indicating positive progress. This kind of message is provided by the Environment when operations complete successfully. This message is denoted by +++.

Sharp_Msg

Specifies a message that has no predefined meaning. This kind of message is not used by the Environment but can be used by users to insert messages into the log files for user-defined purposes. This message is denoted by ###.

`Warning_Msg`

Specifies a message that warns of some minor error. This kind of message indicates that the operation has found some condition that is not quite correct but is not serious enough to prevent the operation from continuing. This message is denoted by !!!.

References

type Log_Prefix

---

subtype Name

```
subtype Name is String;
```

Description

Defines a name.

This type allows all of the special names, wildcards, context prefixes, and attributes that are defined for naming objects in the Environment. This type, however, must be unambiguous; it can resolve only to a single object. For more information, see LM, Key Concepts.

renamed function 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.Activity;
              Log_File : Log_Output_File := Profile.Log_File)
                                  return Response_Profile renames Response;
```

Description

Returns a response profile that ignores all log messages.

The function can be used in other commands to use, or to set as the default, a profile that ignores all log messages. This profile uses the session default profile for all but the log filter. The filter used filters out all log messages so that none are added to the log file.

Parameters

`Reaction : Error_Reaction := Profile.Quit;`

Specifies the error reaction for the profile. The default is the quit response.

`Filter : Log_Filter := Profile.Quiet;`

Specifies the log filter for the profile. The default is the quiet filter, which allows no messages to be added to the log file.

`Prefixes : Log_Prefixes := Profile.No_Prefixes;`

Specifies the set of log prefixes for the profile. The default is that no prefixes be used.

`Width : Natural := Profile.Default_Width;`

Specifies the width of the log file. The default is Default_Width.

`Activity : Activity_Type := Profile.Activity;`

Specifies the activity for the profile. The default is the activity for the current job response profile.

```
Log_File : Log_Output_File := Profile.Log_File;
```

Specifies the log file for the profile. The default is log file for the current job response profile.

```
return Response_Profile;
```

Returns the job response profile.

constant No_Prefixes

```
No_Prefixes : constant Log_Prefixes := (Nil, Nil, Nil);
```

Description

Defines a constant that specifies that no prefixes be used.

function Persevere

```
function Persevere (Response : Response_Profile := Profile.Get)
                                                  return Boolean;
```

Description

Returns whether the specified profile's error reaction allows any command that uses that profile to continue executing after errors are discovered.

The function returns a Boolean indicating whether commands that use the profile will continue executing—that is, whether the error reaction, defined as part of the profile, is set either to the Persevere enumeration or to the Raise_Error enumeration.

Parameters

```
Response : Response_Profile := Profile.Get;
```

Specifies the profile to be checked. The default is the job response profile.

```
return Boolean;
```

Returns true if the error reaction that is part of the response profile is set to the Persevere enumeration or the Raise_Error enumeration.

function Prefixes

```
function Prefixes (Response : Response_Profile := Profile.Get)
                                                    return Log_Prefixes;
```

Description

Returns the log prefixes defined as part of the specified profile.

Parameters

```
Response : Response_Profile := Profile.Get;
```

Specifies the profile from which to get the log prefixes. The default is the job response profile.

```
return Log_Prefixes;
```

Returns the log prefixes from the response profile.

function Propagate

```
function Propagate (Response : Response_Profile := Profile.Get)
                                                     return Boolean;
```

Description

Returns true if the specified response profile's error reaction requires any commands using this profile to raise an exception after errors are discovered.

The function returns a Boolean indicating whether commands that use the profile will raise an exception—that is, whether the error reaction, defined as part of the profile, is set to either the Propagate enumeration or the Raise_Error enumeration.

Parameters

```
Response : Response_Profile := Profile.Get;
```

Specifies the profile to be checked. The default is the job response profile.

```
return Boolean;
```

Returns true if the error reaction that is part of the response profile is set to the Propagate enumeration or the Raise_Error enumeration.

constant Quiet

```
Quiet : constant Log_Filter := Log_Filter'(others => False);
```

Description

Defines a constant that represents a filter that allows no messages to be logged.

The constant can be used in procedures, such as the Set_Response procedure, to specify that no messages should appear in the log file.

renamed function Raise_Exception

```
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;
```

Description

Returns a response profile that forces commands to raise an exception when an error occurs.

The function can be used in other commands to use, or set as the default, a response profile that forces commands to raise an exception when an error occurs. This response profile uses the session response profile for all but the error response. The error response is set to the Propagate enumeration, which allows the command to continue executing as best it can but requires it to raise an exception if an error occurs.

Parameters

`Reaction :  Error_Reaction := Profile.Propagate;`

Specifies the error reaction for the response profile. The default is the propagate response.

`Filter :  Log_Filter := Profile.Filter;`

Specifies the log filter for the profile. The default is the defined filter in the current job response profile.

`Prefixes :  Log_Prefixes := Profile.Prefixes;`

Specifies the set of log prefixes for the profile. The default is the defined prefixes in the current job response profile.

`Width :  Natural := Profile.Width;`

Specifies the width of the log file. The default is the defined width in the current job response profile.

```
Activity :  Activity_Type := Profile.Activity;
```

Specifies the activity for the profile. The default is the activity for the current job response profile.

```
Log_File :  Log_Output_File := Profile.Log_File;
```

Specifies the log file for the profile. The default is the log file for the current job response profile.

```
return Response_Profile;
```

Returns the response profile.

---

function Reaction

```
function Reaction (Response : Response_Profile := Profile.Get)
                                                return Error_Reaction;
```

Description

Returns the default error reaction for the job.

Many commands in the Environment have a parameter that specifies what action to take if an error occurs during execution of that command. This function is used to return the default error reaction as the default value of that parameter.

Parameters

```
Response : Response_Profile := Profile.Get;
```

Specifies the profile from which to return the error reaction. The default is the job response profile.

```
return Error_Reaction;
```

Returns the default error reaction for the job.

function Remote_Passwords

```
function Remote_Passwords (Response : Response_Profile := Profile.Get)
                                          return Remote_Passwords_Type;
```

Description

Returns the remote passwords file for the given response profile.

For further information, see the introduction to this package.

Parameters

```
Response :  Response_Profile := Profile.Get;
```

Specifies the profile from which to return the remote passwords file. The default is the current job response profile.

```
return Remote_Passwords_Type;
```

Returns the remote passwords file for the current job.

subtype Remote_Passwords_Type

```
subtype Remote_Passwords_Type  is Directory.Object;
```

Description

Defines the type that represents a remote passwords file.

For further information on remote passwords, see the introduction to this package.

function Remote_Sessions

```
function Remote_Sessions (Response : Response_Profile := Profile.Get)
                                             return Remote_Sessions_Type;
```

Description

Returns the remote sessions file for the given response profile.

For further information on remote sessions, see the introduction to this package.

Parameters

```
Response : Response_Profile := Profile.Get;
```

Specifies the profile from which to return the remote sessions file. The default is the current job response profile.

```
return Remote_Sessions_Type;
```

Returns the remote sessions file for the current job.

subtype Remote_Sessions_Type

```
subtype Remote_Sessions_Type is Directory.Object;
```

Description

Defines the type that represents a remote sessions file.

For further information, see the introduction to this package.

function Response

```
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;
```

Description

Returns a response profile that can be tailored with a specific reaction, filter, prefixes, width, activity, and log file.

The function can be used in other commands to use, or to set as the default, a specific profile. This function uses the current job response profile by default, but any of the parameters can be changed to specify a different element of the profile to tailor the profile.

Parameters

```
Reaction : Error_Reaction := Profile.Reaction;
```

Specifies the error reaction for the response profile. The default is the defined reaction in the current job response profile.

```
Filter : Log_Filter := Profile.Filter;
```

Specifies the log filter for the response profile. The default is the defined filter in the current job response profile.

```
Prefixes : Log_Prefixes := Profile.Prefixes;
```

Specifies the set of log prefixes for the profile. The default is the defined prefixes in the current job response profile.

```
Width : Natural := Profile.Width;
```

Specifies the width of the log file. The default is the defined width in the current job response profile.

```
Activity : Activity_Type := Profile.Activity;
```

Specifies the activity for the profile. The default is the activity for the current job response profile.

```
Log_File : Log_Output_File := Profile.Log_File;
```

Specifies the log file for the profile. The default is the log file for the current job response profile.

```
return Response_Profile;
```

Returns the response profile.

type Response_Profile

```
type Response_Profile is private;
```

Description

Defines the response profile.

The type defines the set of elements that is collectively called a response profile. A profile consists of an error reaction, a log filter, a set of log prefixes, a log width, remote passwords, remote sessions, and an activity. All of these are described further in the introduction to this package.

procedure Set

```
procedure Set (Profile : Response_Profile);
```

Description

Sets the profile for the rest of the current job.

This procedure can be used to change the response profile for the rest of a job.

Parameters

`Profile : Response_Profile;`
Specifies the string representation of the profile.

procedure Set

```
procedure Set (Profile :        String;
               Status  : in out Simple_Status.Condition);
```

Description

Sets the profile for the rest of the current job.

This procedure can be used in user-written procedures to change the profile for the rest of a job.

Parameters

`Profile : String;`

Specifies the string representation of the profile.

`Status : in out Simple_Status.Condition;`

Returns the status indicating that the procedure has executed correctly, or, if there is an error, a message indicating the type of error.

References

PT, package Simple_Status

procedure Set_Activity

```
procedure Set_Activity (Activity : Activity_Type);
```

Description

Sets the activity in the job response profile to be the specified activity.

Parameters

`Activity : Activity_Type;`

Specifies the activity to which to set the activity in the job response profile. For further information on activities, see Project Management (PM).

References

Project Management (PM)

procedure Set_Default

```
procedure Set_Default (Profile : Response_Profile);

procedure Set_Default (Profile :        String;
                       Status  : in out Simple_Status.Condition);
```

Description

Sets the session response profile to be the specified response profile.

Parameters

`Profile : Response_Profile;`

Specifies the response profile to which to set the session response profile.

`Profile : String;`

Specifies the string representation of the profile.

`Status : in out Simple_Status.Condition;`

Returns the status indicating that the procedure has executed correctly, or, if there is an error, a message indicating the type of error.

References

procedure Log.Put_Condition

PT, package Simple_Status

procedure Set_Default_Activity

```
procedure Set_Default_Activity (Activity : Activity_Type);
```

Description

Sets the activity in the session response profile to be the specified activity.

Parameters

```
Activity : Activity_Type;
```

Specifies the activity to which to set the activity in the session response profile. For further information, see Project Management (PM).

References

Project Management (PM)

procedure Set_Default_Filter

```
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);
```

Description

Sets the filter in the session response profile to be the specified filter.

Parameters

`Auxiliaries : Boolean := True;`

Specifies whether to filter out the auxiliary messages. The default is to leave the auxiliary messages in the log.

`Diagnostics : Boolean := True;`

Specifies whether to filter out the diagnostic messages. The default is to leave the diagnostic messages in the log.

`Notes : Boolean := True;`

Specifies whether to filter out the note messages. The default is to leave the note messages in the log.

`Positives : Boolean := True;`

Specifies whether to filter out the positive messages. The default is to leave the positive messages in the log.

`Negatives : Boolean := True;`

Specifies whether to filter out the negative messages. The default is to leave the negative messages in the log.

```
Positions : Boolean := True;
```

Specifies whether to filter out the position messages. The default is to leave the position messages in the log.

```
Warnings : Boolean := True;
```

Specifies whether to filter out the warning messages. The default is to leave the warning messages in the log.

```
Errors : Boolean := True;
```

Specifies whether to filter out the error messages. The default is to leave the error messages in the log.

```
Exceptions : Boolean := True;
```

Specifies whether to filter out the exception messages. The default is to leave the exception messages in the log.

```
Sharps : Boolean := True;
```

Specifies whether to filter out the messages denoted with the sharp or pound sign (#). The default is to leave the sharp messages in the log.

```
Dollars : Boolean := True;
```

Specifies whether or not to filter out the messages denoted with the dollar sign ($). The default is to leave the dollar messages in the log.

```
Ats : Boolean := True;
```

Specifies whether or not to filter out the messages denoted with the *at* sign (@). The default is to leave the *at* messages in the log.

References

type Log_Output_File, enumerations

---

procedure Set_Default_Filter

```
procedure Set_Default_Filter (Filter : Log_Filter);
```

Description

Sets the filter in the session response profile to be the specified filter.

Parameters

`Filter : Log_Filter;`

Specifies the filter to which to set the filter in the session response profile.

procedure Set_Default_Log_File

```
procedure Set_Default_Log_File (Log_File : Log_Output_File);
```

Description

Sets the default log file to the specified log output file.

This can be one of the following enumerations of the Log_Output_File type: Use-_Output, Use_Error, Use_Standard_Output, Use_Standard_Error.

Parameters

```
Log_File : Log_Output_File;
```

Specifies the log prefixes to which to set the log prefixes in the job response profile.

procedure Set_Default_Prefixes

```
procedure Set_Default_Prefixes (Prefixes : Log_Prefixes);

procedure Set_Default_Prefixes (Prefix1, Prefix2, Prefix3 : Log_Prefix :=
                                                                      Nil);
```

Description

Sets the log prefixes in the session response profile to the specified prefixes.

Parameters

```
Prefixes : Log_Prefixes;
```

Specifies the log prefixes for the session response profile.

```
Prefix1 : Log_Prefix := Nil;
```

Specifies the log prefix to which to set the first log prefix in the session response profile.

```
Prefix2 : Log_Prefix := Nil;
```

Specifies the log prefix to which to set the second log prefix in the session response profile.

```
Prefix3 : Log_Prefix := Nil;
```

Specifies the log prefix to which to set the third log prefix in the session response profile.

procedure Set_Default_Reaction

```
procedure Set_Default_Reaction (Reaction : Error_Reaction);
```

Description

Sets the error reaction in the session response profile to the specified error reaction.

Parameters

```
Reaction :  Error_Reaction;
```

Specifies the error reaction to which to set the error reaction in the session response profile.

procedure Set_Default_Remote_Passwords

```
procedure Set_Default_Remote_Passwords (Passwords : Remote_Passwords_Type);
```

Description

Sets the remote passwords file for the current session.

For further information, see the introduction to this package.

Parameters

```
Passwords : Remote_Passwords_Type;
```

Specifies the file containing the remote passwords. The default is `<Nil>`, meaning that there is no remote passwords file. This is set in the session switches.

procedure Set_Default_Remote_Sessions

```
procedure Set_Default_Remote_Sessions (Sessions : Remote_Sessions_Type);
```

Description

Sets the remote sessions file for the current session.

For further information, see the introduction to this package.

Parameters

```
Sessions :  Remote_Sessions_Type;
```

Specifies the file containing the remote sessions. The default is `<Nil>`, meaning that there is no remote passwords file. This is set in the session switch file.

procedure Set_Default_Response

```
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));
```

Description

Sets the session response profile to the specified reaction, filter, prefixes, width, activity, and log file.

The default parameters specify the current session default values, which have the effect of not changing the session default profile. However, any of the parameters can be changed to affect only some of the default values.

Parameters

```
Reaction : Error_Reaction := Profile.Reaction (Profile.Get_Default);
```

Specifies the error reaction to which to set the error reaction in the session response profile. The default is the current value of the error reaction in the session response profile.

```
Filter : Log_Filter := Profile.Filter (Profile.Get_Default);
```

Specifies the filter to which to set the filter in the session default profile. The default is the current value of the filter in the session response profile.

```
Prefixes : Log_Prefixes := Profile.Prefixes (Profile.Get_Default);
```

Specifies the log prefixes to which to set the log prefixes in the session response profile. The default is the current value of the log prefixes in the session response profile.

```
Width : Natural := Profile.Width (Profile.Get_Default);
```

Specifies the log width to which to set the log width in the session response profile. The default is the current value of the log width in the session response profile.

`Activity : Activity_Type := Profile.Activity (Profile.Get_Default);`

Specifies the activity for the profile. The default is the activity for the session response profile.

`Log_File : Log_Output_File := Profile.Log_File (Profile.Get_Default);`

Specifies the log file for the profile. The default is the log file for the session response profile.

---

procedure Set_Default_Width

```
procedure Set_Default_Width (Width : Natural);
```

Description

Sets the log width to the specified value in the session response profile.

Parameters

```
Width :  Natural;
```

Specifies the log width to which to set the log width in the session response profile.

procedure Set_Filter

```
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);
```

Description

Sets the filter in the job response profile to be the specified filter.

Parameters

`Auxiliaries : Boolean := True;`

Specifies whether to filter out the auxiliary messages. The default is to leave the auxiliary messages in the log.

`Diagnostics : Boolean := True;`

Specifies whether to filter out the diagnostic messages. The default is to leave the diagnostic messages in the log.

`Notes : Boolean := True;`

Specifies whether to filter out the note messages. The default is to leave the note messages in the log.

`Positives : Boolean := True;`

Specifies whether to filter out the positive messages. The default is to leave the positive messages in the log.

`Negatives : Boolean := True;`

Specifies whether to filter out the negative messages. The default is to leave the negative messages in the log.

```
Positions : Boolean := True;
```

Specifies whether to filter out the position messages. The default is to leave the position messages in the log.

```
Warnings : Boolean := True;
```

Specifies whether to filter out the warning messages. The default is to leave the warning messages in the log.

```
Errors : Boolean := True;
```

Specifies whether to filter out the error messages. The default is to leave the error messages in the log.

```
Exceptions : Boolean := True;
```

Specifies whether to filter out the exception messages. The default is to leave the exception messages in the log.

```
Sharps : Boolean := True;
```

Specifies whether to filter out the messages denoted with the sharp or pound sign (#). The default is to leave the sharp messages in the log.

```
Dollars : Boolean := True;
```

Specifies whether to filter out the messages denoted with the dollar sign ($). The default is to leave the dollar messages in the log.

```
Ats : Boolean := True;
```

Specifies whether to filter out the messages denoted with the *at* sign (@). The default is to leave the *at* messages in the log.

procedure Set_Filter

```
procedure Set_Filter (Filter : Log_Filter);
```

Description

Sets the filter in the job response profile to be the specified filter.

Parameters

```
Filter : Log_Filter;
```

Specifies the filter to which to set the filter in the job response profile.

procedure Set_Log_File

```
procedure Set_Log_File (Log_File : Log_Output_File);
```

Description

Sets the log file for the current job.

Parameters

```
Log_File :  Log_Output_File;
```

Specifies the log file to which to set the log file for the current job.

procedure Set_Prefixes

```
procedure Set_Prefixes (Prefixes : Log_Prefixes);

procedure Set_Prefixes (Prefix1, Prefix2, Prefix3 : Log_Prefix := Nil);
```

Description

Sets the log prefixes to the specified log prefixes in the job response profile.

Parameters

`Prefixes : Log_Prefixes;`
Specifies the log prefixes to which to set the log prefixes in the job response profile.

`Prefix1 : Log_Prefix := Nil;`
Specifies the prefix to which to set the first log prefix in the job response profile.

`Prefix2 : Log_Prefix := Nil;`
Specifies the prefix to which to set the second log prefix in the job response profile.

`Prefix3 : Log_Prefix := Nil;`
Specifies the prefix to which to set the third log prefix in the job response profile.

procedure Set_Reaction

```
procedure Set_Reaction (Reaction : Error_Reaction);
```

Description

Sets the error reaction in the job response profile to the specified error reaction.

Parameters

```
Reaction :  Error_Reaction;
```

Specifies the error reaction to which to set the error reaction in the job response profile.

procedure Set_Remote_Passwords

```
procedure Set_Remote_Passwords (Passwords : Remote_Passwords_Type);
```

Description

Sets the remote passwords file for the current job.

For further information, see the introduction to this package.

Parameters

```
Passwords :  Remote_Passwords_Type;
```
Specifies the file containing the remote passwords.

procedure Set_Remote_Sessions

```
procedure Set_Remote_Sessions (Sessions : Remote_Sessions_Type);
```

Description

Sets the remote sessions file for the current job.

For further information, see the introduction to this package.

Parameters

```
Sessions :  Remote_Sessions_Type;
```

Specifies the file containing the remote sessions.

procedure Set_Response

```
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);
```

Description

Sets the job response profile to the specified reaction, filter, prefixes, width, activity, and log file.

The default parameters specify the current job's values, which have the effect of not changing the job response profile. However, any of the parameters can be changed to affect only some of the default values.

Parameters

`Reaction : Error_Reaction := Profile.Reaction;`

Specifies the error reaction to which to set the error reaction in the job response profile. The default is the current value of the error reaction in the job response profile.

`Filter : Log_Filter := Profile.Filter;`

Specifies the filter to which to set the filter in the job response profile. The default is the current value of the filter in the job response profile.

`Prefixes : Log_Prefixes := Profile.Prefixes;`

Specifies the log prefixes to which to set the log prefixes in the job response profile. The default is the current value of the log prefixes in the job response profile.

`Width : Natural := Profile.Width;`

Specifies the log width to which to set the log width in the job response profile. The default is the current value of the log width in the job response profile.

`Activity : Activity_Type := Profile.Activity;`

Specifies the activity for the profile. The default is the activity for the job response profile.

```
Log_File : Log_Output_File := Profile.Log_File;
```

Specifies the log file for the response profile. The default is the log file for the job response profile.

---

procedure Set_Width

```
procedure Set_Width (Width : Natural);
```

Description

Sets the log width in the job response profile to the specified log width.

Parameters

```
Width : Natural;
```

Specifies the log width to which to set the log width in the job response profile. The maximum width is 1,024.

constant Summary

```
Summary : constant Log_Filter := Log_Filter'(Positive_Msg  |
                                       Negative_Msg => True, others => False);
```

Description

Defines a constant that represents a filter that allows only summary messages to be logged.

The constant can be used in procedures, such as the Set_Response procedure, to specify that only summary messages should appear in the log file.

constant Terse

```
Terse : constant Log_Filter := Log_Filter'(False, False, False, others =>
                                                                    True);
```

Description

Defines a constant that represents a filter that allows only a terse set of messages to be logged.

Terse messages include the following message types: positive, position, negative, warning, error, exception, sharp, at, and dollar.

The constant can be used in procedures, such as the Set_Response procedure, to specify that only this terse set of messages should appear in the log file.

References

type Msg_Kind

function Value

```
function Value (Image : String := Profile.Get) return Response_Profile;
```

Description

Returns the internal representation of the response profile.

Given a string representation of a response profile, this function returns the internal representation of that response profile.

The procedural form of this command is Convert.

Parameters

```
Image : String := Profile.Get;
```

Specifies the string representation of the response profile to be converted to the internal representation. The default profile to be converted is the job response profile.

```
return Response_Profile;
```

Returns the internal representation of the response profile.

References

procedure Convert

renamed function Verbose

```
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;
```

Description

Returns a response profile that puts all messages except debug messages in the log file.

The function can be used in other commands to use, or to set as the default, a response profile that puts all messages except debug messages in the log file. This response profile uses the job response profile for all but the log filter. The log filter is set to the Full constant, which puts all but debug messages in the log file.

Parameters

```
Reaction : Error_Reaction := Profile.Reaction;
```

Specifies the error reaction for the profile. The default is the defined error reaction in the job response profile.

```
Filter : Log_Filter := Profile.Full;
```

Specifies the log filter for the response profile. The default is the full filter.

```
Prefixes : Log_Prefixes := Profile.Prefixes;
```

Specifies the set of log prefixes for the profile. The default is the defined prefixes in the job response profile.

```
Width : Natural := Profile.Width;
```

Specifies the width of the log file. The default is the defined width in the job response profile.

```
Activity : Activity_Type := Profile.Activity;
```

Specifies the activity for the profile. The default is the activity for the job response profile.

```
Log_File : Log_Output_File := Profile.Log_File;
```

Specifies the log file for the response profile. The default is the log file for the job response profile.

```
return Response_Profile;
```

Returns the response profile.

---

renamed function Warn

```
function Warn (Reaction : Error_Reaction  := Profile.Persevere;
               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;
```

Description

Returns a response profile that puts all messages except debug messages in the log file and requests the command to persevere without raising exceptions.

The function can be used in other commands to use, or to set as the default, a profile that puts all messages except debug messages in the log file and requests the command to persevere without raising exceptions. This profile uses the job response profile for all but the error response and log filter. The error response is set to the Persevere enumeration and the log filter is set to Profile.Filter.

Parameters

`Reaction : Error_Reaction := Profile.Persevere;`

Specifies the error reaction for the response profile. The default is the persevere response.

`Filter : Log_Filter := Profile.Filter;`

Specifies the log filter for the response profile. The default is Profile.Filter.

`Prefixes : Log_Prefixes := Profile.Prefixes;`

Specifies the set of log prefixes for the profile. The default is the defined prefixes in the current job response profile.

`Width : Natural := Profile.Width;`

Specifies the width of the log file. The default is the defined width in the job response profile.

`Activity : Activity_Type := Profile.Activity;`

Specifies the activity for the response profile. The default is the activity for the job response profile.

```
Log_File : Log_Output_File := Profile.Log_File;
```

Specifies the log file for the profile. The default is the log file for the job response profile.

```
return Response_Profile;
```

Returns the response profile.

function Width

```
function Width (Response : Response_Profile := Profile.Get) return Natural;
```

Description

Returns the log width from the specified response profile.

Parameters

```
Response : Response_Profile := Profile.Get;
```

Specifies the profile from which to return the log width. The default is the job response profile.

```
return Natural;
```

Returns the log width. If no log width has been specified in the response profile, the system default, the Default_Width constant, is returned.

end Profile;

package Program

Package Program provides a means of compiling and executing programs from other programs. The procedures in this package execute the specified list of Ada statements, presented as a string, in a specified context. The package also provides various utilities for writing applications that must create and manage jobs programmatically. These utilities include a routing for changing the access control identity of executing jobs.

For more information on access control, see LM, packages Access_List and Access_List_Tools.

procedure Change_Identity

```
procedure Change_Identity (To_User  :        String     := "";
                           Password :        String     := "";
                           Options  :        String     := "";
                           Status   : in out Condition);
```

Description

Changes the access control identity of the calling job to the specified user and changes other characteristics of the job as specified by the Options parameter.

A password must be supplied and correct unless the caller is privileged. The Options parameter specifies additional changes in the characteristics of the calling job.

Note that only the access control identity is changed. The actual username and session of the job are not changed. The identity can always be changed back to the original job identity.

Parameters

`To_User : String := "";`

Specifies the user to whom to change the access control identity. If the default value of the null string is supplied, the username is not changed.

`Password : String := "";`

Specifies the password for the user to whom access control identity is being changed. A password is not required if the calling job is privileged or the username is not being changed.

`Options : String := "";`

Specifies additional changes to the characteristics of the calling job. For information on the syntax of options parameters, see the Key Concepts in this book. If the To_User parameter is null, the Options parameter is still processed. Note that option names and values are literals and do not have to be quoted. For example, if a username or password is to be specified, the username or password is provided as a literal in the option string, not a string embedded in the options string.

The syntax for specifying an option and its value is the option name followed by a value delimiter (=, :=, or =>). Options and option/value pairs should be separated by spaces, commas, or semicolons.

For example, to specify that the calling job should become unprivileged and should return to its original identity, the string "Privileged=>False; Restore_Identity" should be supplied as the Options parameter.

The legal options are:

`Privileged`

Enables privileged mode for the calling job. The specified user, or the current user if none is specified, must be a member of group Privileged.

`Privileged` Boolean

Enables or disables privileged mode based on the value supplied. Disabling has no effect if the caller is not already privileged.

`Restore_Identity`

Changes the access control identity of the calling job back to the original identity of the job. A password is not required to do this.

`Status : in out Condition;`

Returns the status of the attempt to change the identity of the calling job. If !Tools.Simple_Status.Error (Status)=False, the operation is successful. If it is not successful, the Log.Put_Condition (Status) command displays an error message describing the cause of the problem to the Current_Output window or file.

References

procedure Log.Put_Condition

LM, package Access_List

LM, package Access_List_Tools

PT, package Simple_Status

PT, function Simple_Status.Error

SMU, procedure Operator.Enable_Privileges

subtype Condition

```
subtype Condition is Simple_Status.Condition;
```

Description

Defines the status resulting from an attempted operation.

Typically, a condition is returned from an operation such as a call of the Create_Job or the Change_Identity procedure. To determine whether the operation has completed successfully, the condition returned can be interrogated with the !Tools.Simple_Status.Error function; for conditions returned by calls to the Create_Job procedure, the Started_Successfully function can also be used. If there are errors, the !Tools.Log.Put_Condition procedure can be used to display the cause of the error to the current output window or file.

References

procedure Change_Identity

procedure Create_Job

function Started_Successfully

procedure Log.Put_Condition

PT, package Simple_Status

PT, function Simple_Status.Error

procedure Create_Job

```
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>");
```

Description

Creates a job to run the statements specified by the S parameter in the specified context and returns the identity of that job.

This procedure formats, compiles, links, and executes the Ada statements of the first parameter. The context in which the fragment executes is named in the Context parameter. A new job is created to execute the statements once they are successfully formatted, compiled, and linked, and the identity of this job is returned in the Job parameter. The call to Create_Job returns once this job has been started. Note that, if the identity of the job is not needed, the Run_Job procedure is more convenient to use.

The Ada statements can be any series of statements that can be placed between a begin/end pair. The begin/end pair is implicit.

If the default value of the S parameter is used, the procedure designated by the selection is executed. The selection must be in an Ada unit or a library. If the selection is in a library, the procedure that is selected is executed. If the selection is in an Ada unit, the procedure to be executed must be visible (that is, it must be a library unit or it must be visible in a package specification). If an entire procedure is selected in an Ada unit, it is the procedure executed. Otherwise, the procedure that contains the selection is executed.

The job begins execution of the statements after the specified duration. If the Debug parameter is true, the new job is started with the Rational Debugger.

The Status parameter returns an indication of the success or failure of the attempt to start the job. This condition can be interrogated by the Started_Successfully function.

The Options parameter can be used to specify various options for running the job, including its input, output, and error files as well as its username and password (which determine its access control identity) and session.

If no username is specified, the new job runs with the same access control identity as the initiating job. If the initiating job has changed its identity using procedure

Change_Identity and no username is specified, the new job runs under the changed identity. If a username and password are specified, these must be valid and the new job will run with that access control identity. If the initiating job is running in privileged mode, the password need not be supplied.

The session associated with the new job affects how it interacts with the terminal and other windows of that session. Note that activities and session switches are always inherited from the initiating job, never the new job's session. If the Session and User options are absent, the session of the initiating job is used for the new job. If the User option is present but no Session option is supplied, the default session of the specified user (S_1) is used. If a user and a session are specified, the new job is associated with that session. If a session is specified but a user is not, the Session option is ignored.

By default, the normal editor windows will be used for Current_Output, Current_Error, and Current_Input, respectively, if the user's session is still active. If the session is not active and the defaults are used, files named Output, Error, and Input will be used instead (note that these filenames will be resolved using the context of the caller of Create_Job, not the context supplied by the Context parameter). Filenames other than these can be specified using the appropriate options for starting the job.

Parameters

```
S : String := "<SELECTION>";
```

Specifies the Ada statements to be formatted, compiled, linked, and executed. Note that partially complete statements are formatted before they are compiled. Line breaks do not need to be inserted between statements unless the length of a given line exceeds the maximum line length for Ada programs. The default value of the parameter is the selected Ada statements.

```
Job :  out Job_Id;
```

Returns the identity of the job that is started. If the job is not started successfully, the value returned is undefined.

```
Status :  in out Condition;
```

Returns an indication of the success or failure of the attempt to create the job. To determine whether a job has started successfully, the Started_Successfully function can be used to interrogate the condition returned by the call to Create_Job. If the job has not started successfully, the Log.Put_Condition procedure can be used to display the cause of the error to the Current_Output window or file.

```
Debug :  Boolean := False;
```

Specifies whether the job should be started with the debugger. By default, the debugger will not be started.

```
Context : String := "$";
```

Specifies the context in which the Ada statements are compiled. The string provided can be any legal library name. The default value of the parameter is the library containing current context or the current context if it is a library.

```
After : Duration := 0.0;
```

Specifies how long the job should wait before beginning to execute the statements.

```
Options : String := "";
```

Specifies options to be used for running the job, which can include its input, output, and error files as well as its username and password (which determine its access control identity) and session. Note that option names and values are literals and do not have to be quoted. For example, if a username or password is to be specified, the username or password is provided as a literal in the option string, not a string embedded in the options string.

If no username is specified, the new job runs with the same access control identity as the initiating job. If the initiating job has changed its identity using the Change_Identity procedure and no username is specified, the new job runs under the changed identity. If a username and a password are specified, these must be valid and the new job will run with that access control identity. If the initiating job is running in privileged mode, the password need not be supplied.

The session associated with the new job affects how it interacts with the terminal and other windows of that session. Note that activities and session switches are always inherited from the initiating job, never the new job's session. If the Session and User options are absent, the session of the initiating job is used for the new job. If the User option is present, but no Session option is supplied, the default session of the specified user (S_1) is used. If a user and a session are specified, the new job is associated with that session. If a session is specified but a user is not, the Session option is ignored.

If no options are specified, the normal editor windows will be used for Current_Output, Current_Error, and Current_Input, respectively, if the user's session is still active. If the session is not active, files named Output, Error, and Input will be used instead (note that these filenames will be resolved using the context of the caller of Create_Job, not the context supplied by the Context parameter).

The syntax for specifying an option and its value is the option name followed by a value delimiter (=, :=, or =>). Option/value pairs should be separated by spaces, commas, or semicolons.

For example, to specify that the new job should have the files called Temp_Input and Temp_Output for its input and output files, respectively, the string "Input=>Temp-_Input; Output=>Temp_Output" could be supplied.

The legal options are described below:

> Input=*filename*
>
> Specifies that this file be used for the new job's Current_Input file. Note that the name of this file is resolved using the context of the caller of Create_Job, not the context supplied by the Context parameter.
>
> Output=*filename*
>
> Specifies that this file be used for the new job's Current_Output file. Note that the name of this file is resolved using the context of the caller of Create_Job, not the context supplied by the Context parameter.
>
> Error=*filename*
>
> Specifies that this file be used for the new job's Current_Error file. Note that the name of this file is resolved using the context of the caller of Create_Job, not the context supplied by the Context parameter.
>
> User=*username*
>
> Specifies that the new job be run with the identity of this user. A Password option also must be supplied unless the job of the caller to Create_Job is privileged. If a User option is not specified, the new job runs with the same identity as the parent, and the Session and Password options are ignored.
>
> Password=*password value*
>
> Specifies that this password be used. This option is used in conjunction with the User option to supply the password for the user. If the User option is not supplied, this option is ignored.
>
> Session=*session name*
>
> Specifies the session for the newly created job. This option is used in conjunction with the User option. If a Session option is not supplied, the default session S_1 will be used. Note that the session must already exist. If the User option is not supplied, this option is ignored.

```
Response : String := "<PROFILE>";
```

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the current response profile for the initiating job. Note that the response specified by this parameter is used by the job making the call to Run_Job; it is not used by the job that is started as a result of the call to Run_Job.

Restrictions

The Create_Job, Run, and Run_Job procedures should not be called concurrently from a single job—that is, within a job only one task should be executing a call to any of these procedures at any given time. If calls are made concurrently from a single job, unpredictable errors will result.

Errors

Errors can occur when the Ada statement is syntactically or semantically incorrect, when code generation errors exist, or when improper activities are provided. The executed Ada statements also can produce errors. Errors other than those occurring in the execution of the Ada statements will be reported as part of the current job, based on the response specified in the Response parameter.

Example

This example is a program that creates a scheduling server as a separate running job and then waits until the server completes:

```
with Program, Simple_Status;
procedure Start_Server is
    Job : Program.Job_Id;
    Status : Simple_Status.Condition;
begin
    Program.Create_Job ("""!Projects.Schedule_Board"".Scheduler;",
                        Context => "!Projects.Schedule_Board",
                        Job => Job, Status => Status);
    if Program.Started_Successfully (Status) then
        Program.Wait_For(Job);
    else
        Log.Put_Condition (Status);
    end if;
end Start_Server;
```

References

procedure Run

procedure Run_Job

function Started_Successfully

procedure Wait_For

procedure Log.Put_Condition

PT, package Simple_Status

function Current

```
function Current (Subsystem  : String := ">>SUBSYSTEM NAME<<";
                  Unit       : String := ">>PROCEDURE NAME<<";
                  Parameters : String := "";
                  Activity   : String := "<ACTIVITY>") return String;
```

Description

Returns a string containing a procedure call to the current view of a subsystem suitable for passing as the S parameter to the Create_Job, Run, and Run_Job procedures.

This function is useful, for example, for constructing general-purpose routines for starting servers and so forth that do not need to be modified each time a new version of the application is released.

The Subsystem parameter defines the subsystem in which the procedure to be called is contained. The Activity parameter points to the subsystem load view in which to look for the procedure. The Unit parameter defines the name of a procedure that will be located anywhere in the load view or its links, even if it does not reside in the view. The Parameters parameter provides the parameters for the procedure call.

Parameters

`Subsystem : String := ">>SUBSYSTEM NAME<<";`

Specifies the subsystem in which the procedure to be called is contained. This subsystem will be located in the activity specified by the Activity parameter, and the load view entry for this subsystem will be the view to be searched. The default parameter placeholder must be replaced with the name of a subsystem or an error will result.

`Unit : String := ">>PROCEDURE NAME<<";`

Specifies the name of a procedure to look for in the view of the subsystem or its links. Note that the unit does not have to reside in the view. The default parameter placeholder must be replaced with the name of a procedure or an error will result.

`Parameters : String := "";`

Specifies the parameters to be supplied to the procedure. Note that parentheses, commas, quotes, and so forth must be supplied.

```
Activity : String := "<ACTIVITY>";
```

Specifies the activity to be used to find the subsystem and load view containing the procedure to be called. By default, the current activity for the job will be used.

```
return String;
```

Returns a string containing a procedure call that is suitable for the Create_Job, Run, and Run_Job procedures.

Errors

The !Io.Io_Exceptions.Name_Error exception will be raised if the Unit specified cannot be located in the view or in the links of the view.

Example

Assume that the current activity for the calling job contains a load view entry for subsystem !Project_Tools of Rev1_0_0 and that the view contains a procedure in the Units directory with the following specification:

```
procedure Gather_Metrics (Units : String; Verbose : Boolean);
```

Then the call:

```
    Current (Subsystem => "!Project_Tools", Unit => "Gather_Metrics",
             Parameters => " (""!Project_Library"", True)");
```

would return the string with value:

```
"!PROJECT_TOOLS.REV1_Ø_Ø.UNITS".Gather_Metrics ("!Project_Library", True);
```

which could be used in a call to Run_Job in the login procedure for the project administrator to collect project metrics every day each time she or he logged in.

References

procedure Create_Job

procedure Run

procedure Run_Job

Project Management (PM)

---

subtype Job_Id

```
subtype Job_Id is Machine.Job_Id;
```

Description

Defines the identity of a job.

References

procedure Create_Job

procedure Wait_For

procedure Run

```
procedure Run (S        : String := "<SELECTION>";
               Context  : String := "$";
               Response : String := "<PROFILE>");
```

Description

Runs the statements specified by the S parameter in the specified context as part of the current job.

This procedure formats, compiles, links, and executes the Ada statements of the first parameter. The context in which the fragment executes is named in the second parameter. The Ada statements must run to completion before this procedure completes; they are not run as a separate job.

The Ada statements can be any series of statements that can be placed between a begin/end pair. The begin/end pair is implicit.

If the default value of the S parameter is used, the procedure designated by the selection is executed. The selection must be in an Ada unit or a library. If the selection is in a library, the procedure that is selected is executed. If the selection is in an Ada unit, the procedure to be executed must be visible (that is, it must be a library unit or it must be visible in a package specification). If an entire procedure is selected in an Ada unit, it is the procedure executed. Otherwise, the procedure that contains the selection is executed.

Parameters

```
S : String := "<SELECTION>";
```

Specifies the Ada statements to be formatted, compiled, linked, and executed. Note that partially complete statements are formatted before they are compiled. Line breaks do not need to be inserted between statements unless the length of a given line exceeds the maximum line length for Ada programs. The default value of the S parameter indicates that the selected statements or subprogram should be used.

```
Context :  String := "$";
```

Specifies the context in which the Ada statements are compiled. The string provided can be any legal library name. The default value of the parameter is the current library.

```
Response : String := "<PROFILE>";
```

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the current job response profile.

Restrictions

The Run, Create_Job, and Run_Job procedures should not be called concurrently from a single job—that is, within a job only one task should be executing a call to any of these procedures at any given time. If calls are made concurrently from a single job, unpredictable errors will result.

Errors

Errors can occur when the Ada statement is syntactically or semantically incorrect, when code generation errors exist, or when improper activities are provided. The executed Ada statements can also produce errors.

Example

This example shows how to create a login procedure that contains a request for another procedure to be executed:

```
with Program;
procedure Login is
begin
    ...
    Program.Run ("""!Projects.Project_A"".Initialize;");
    ...
end Login;
```

The Run procedure requests that an initialization routine be run. It exists in a library for a specific project and can be changed regularly. The Login procedure in this example does not require demotion and repromotion as it would if the initialization procedure were executed directly from the Login procedure.

References

procedure Create_Job

procedure Run_Job

procedure Run_Job

```
procedure Run_Job (S        : String   := "<SELECTION>";
                   Debug    : Boolean  := False;
                   Context  : String   := "$";
                   After    : Duration := 0.0;
                   Options  : String   := "";
                   Response : String   := "<PROFILE>");
```

Description

Runs the statements specified by the S parameter in the specified context as a separate job.

This procedure formats, compiles, links, and executes the Ada statements of the first parameter. The context in which the fragment executes is named in the Context parameter. A new job is created to execute the statements once they are successfully formatted, compiled, and linked. The call to Run_Job completes once this job has been started. Note that the identity of the job is not returned. If this information is needed, the Create_Job procedure should be used.

The Ada statements can be any series of statements that can be placed between a begin/end pair. The begin/end pair is implicit.

If the default value of the S parameter is used, the procedure designated by the selection is executed. The selection must be in an Ada unit or a library. If the selection is in a library, the procedure that is selected is executed. If the selection is in an Ada unit, the procedure to be executed must be visible (that is, it must be a library unit or it must be visible in a package specification). If an entire procedure is selected in an Ada unit, it is the procedure executed. Otherwise, the procedure that contains the selection is executed.

The job begins execution of the statements after the specified duration.

If the Debug parameter is true, the new job is started with the debugger.

The Options parameter can be used to specify various options for running the job, including its input, output, and error files as well as its username and password (which determine its access control identity) and session.

If no username is specified, the new job runs with the same access control identity as the initiating job. If the initiating job has changed its identity using procedure Change_Identity and no username is specified, the new job runs under the changed identity. If a username and a password are specified, these must be valid and the new job will run with that access control identity. If the initiating job is running in privileged mode, the password need not be supplied.

The session associated with the new job affects how it interacts with the terminal and other windows of that session. Note that activities and session switches are always inherited from the initiating job, never the new job's session. If the Session and User options are absent, the session of the initiating job is used for the new job. If the User option is present, but no Session option is supplied, the default session of the specified user (S_1) is used. If a user and a session is specified, the new job is associated with that session. If a session is specified but a user is not, the Session option is ignored.

By default, the normal editor windows will be used for Current_Output, Current_Error, and Current_Input, respectively, if the user's session is still active. If the session is not active and the defaults are used, files named Output, Error, and Input will be used instead (note that these filenames will be resolved using the context of the caller of Run_Job, not the context supplied by the Context parameter). Filenames other than these can be specified using the appropriate options for starting the job.

Parameters

```
S : String := "<SELECTION>";
```

Specifies the Ada statements to be formatted, compiled, linked, and executed. Note that partially complete statements are formatted before they are compiled. Line breaks do not need to be inserted between statements unless the length of a given line exceeds the maximum line length for Ada programs. The default value of the parameter indicates that the selected statements should be used.

```
Debug : Boolean := False;
```

Specifies whether the job should be started with the debugger. By default, the debugger will not be started.

```
Context : String := "$";
```

Specifies the context in which the Ada statements are compiled. The string provided can be any legal library name. The default value of the parameter is the current library.

```
After : Duration := 0.0;
```

Specifies how long the job should wait before beginning to execute the statements.

```
Options : String := "";
```

Specifies options to be used for running the job, including its input, output, and error files as well as its username and password (which determine its access control identity) and session. Note that option names and values are literals and do not have to be quoted. For example, if a username or password is to be specified, the username or password is provided as a literal in the option string, not a string embedded in the options string.

If no username is specified, the new job runs with the same access control identity as the initiating job. If the initiating job has changed its identity using procedure

Change_Identity and no username is specified, the new job runs under the changed identity. If a username and a password are specified, these must be valid and the new job will run with that access control identity. If the initiating job is running in privileged mode, the password need not be supplied.

The session associated with the new job affects how it interacts with the terminal and other windows of that session. Note that activities and session switches are always inherited from the initiating job, never the new job's session. If the Session and User options are absent, the session of the initiating job is used for the new job. If the User option is present but no Session option is supplied, the default session of the specified user (S_1) is used. If a user and a session are specified, the new job is associated with that session. If a session is specified but a user is not, the Session option is ignored.

If no options are specified, the normal editor windows will be used for Current_Output, Current_Error, and Current_Input, respectively, if the user's session is still active. If the session is not active, files named Output, Error, and Input will be used instead (note that these filenames will be resolved using the context of the caller of Run_Job, not the context supplied by the Context parameter).

The syntax for specifying an option and its value is the option name followed by a value delimiter (=, :=, or =>). Option/value pairs should be separated by spaces, commas, or semicolons.

For example, to specify that the new job should have the files called Temp_Input and Temp_Output for its input and output files, respectively, the string "Input=>Temp-_Input; Output=>Temp_Output" could be supplied.

The legal options are:

Input=*filename*

Specifies that this file be used for the new job's Current_Input file. Note that the name of this file is resolved using the context of the caller of Run_Job, not the context supplied by the Context parameter.

Output=*filename*

Specifies that this file be used for the new job's Current_Output file. Note that the name of this file is resolved using the context of the caller of Run_Job, not the context supplied by the Context parameter.

Error=*filename*

Specifies that this file be used for the new job's Current_Error file. Note that the name of this file is resolved using the context of the caller of Run_Job, not the context supplied by the Context parameter.

User=*username*

Specifies that the new job run with the identity of this user. A Password option also must be supplied unless the job of the caller to Run_Job is privileged. If a User option is not specified, the new job runs with the same identity as the parent, and the Session and Password options are ignored.

`Password=`***password value***

Specifies the password for the user. This option is used in conjunction with the User option to supply the password for the user. If a User option is not supplied, this option is ignored.

`Session=`***session name***

Specifies the session for the newly created job. This option is used in conjunction with the User option. If a Session option is not supplied, the default session S_1 will be used. Note that the session must already exist. If a User option is not supplied, this option is ignored.

```
Response  :  String := "<PROFILE>";
```

Specifies how to respond to errors, how to generate logs, and what activities to use during the execution of this command. The default is the current response profile for the initiating job. Note that the response specified by this parameter is used by the job making the call to Run_Job; it is not used by the job that is started as a result of the call to Run_Job.

Restrictions

The Run_Job, Create_Job, and Run procedures should not be called concurrently from a single job—that is, within a job only one task should be executing a call to any of these procedures at any given time. If calls are made concurrently from a single job, unpredictable errors will result.

Errors

Errors can occur when the Ada statement is syntactically or semantically incorrect, when code generation errors exist, or when improper activities are provided. The executed Ada statements also can produce errors. Errors other than those occurring in the execution of the Ada statements will be reported as part of the current job, based on the response specified in the Response parameter.

Example

This example creates a scheduling server that is a separate running job as part of a login procedure:

```
with Program;
procedure Login is
begin
    ...
    Program.Run_Job ("""!Projects.Schedule_Board"".Scheduler;",
                     Context => "!Projects.Schedule_Board");
    ...
end Login;
```

References

procedure Create_Job

procedure Run

function Started_Successfully

```
function Started_Successfully (Status : Condition) return Boolean;
```

Description

Determines whether an attempt to create a job using the Create_Job procedure is successful.

If the attempt is not successful, the cause of the error can be displayed to the Current_Output window or file using the !Tools.Simple_Status.Display_Message procedure.

Parameters

```
Status : Condition;
```

Specifies the status returned from the attempt to create a job.

```
return Boolean;
```

Returns true if the job to which the condition pertains is started successfully; otherwise, the function returns false.

References

procedure Create_Job

PT, package Simple_Status

PT, procedure Simple_Status.Display_Message

procedure Wait_For

```
procedure Wait_For (Job : Job_Id);
```

Description

Waits at least until the specified job has terminated.

In some cases, the call does not return immediately but returns some time after the job completes.

Parameters

`Job : Job_Id;`
Specifies the identity of the job for which to wait.

References

procedure Create_Job

PT, package Simple_Status

PT, procedure Simple_Status.Display_Message

end Program;

package Queue

Package Queue allows systems managers to set up print queues and allows all users to perform printing operations. The user-performed printing operations are documented in this section. These operations include:

- Making, displaying, and canceling print requests
- Displaying information about devices
- Displaying information about print classes

For information on the other commands in package Queue, see System Management Utilities (SMU).

procedure Cancel

```
procedure Cancel (Request_Id : Positive);
```

Description

Cancels the specified print request.

This procedure cancels requests whether or not those requests have started to print. The Request_Id value can be obtained with the Display procedure.

Although the cancel request completes quickly, the actual canceling may take several minutes until the print spooler can remove/terminate the request.

Parameters

```
Request_Id : Positive;
```

Specifies the number assigned to the print request.

References

procedure Display

procedure Classes

```
procedure Classes (Which        : Class_Name := "all";
                   Show_Devices : Boolean    := True);
```

Description

Displays information about the specified classes.

Parameters

```
Which : Class_Name := "all";
```

Specifies the class for which information is requested. The default is all classes. Users on installations that use Rational Networking—TCP/IP to connect multiple R1000 systems can query other machines on the network. Thus, the name can specify a machine name of the form !!*machine name*, where *machine name* is the name of a machine—for example, !!M1.

```
Show_Devices : Boolean := True;
```

Specifies whether to display information on devices as well as on classes. The default is true.

Example

The command:

```
    queue.classes;
```

produces a display such as the following:

```
     Class    Device(s)
    =======  ============
     LP       TERMINAL_34
     PL       TERMINAL_250
```

This display shows that class LP is associated with the device TERMINAL_34. That is, print requests made to LP are routed to TERMINAL_34. It also shows that class PL is associated with device TERMINAL_250, which is, by convention, a Telnet port.

procedure Devices

```
procedure Devices (Which        : String  := "all";
                   Show_State   : Boolean := True;
                   Show_Classes : Boolean := True);
```

Description

Displays information about the specified devices.

Parameters

`Which : String := "all";`

Specifies the device for which information is requested. The default is to show information about all devices.

`Show_State : Boolean := True;`

Requests information on the current state of the devices, whether enabled or disabled. The default is true.

`Show_Classes : Boolean := True;`

Specifies whether to display information on classes associated with the displayed devices. The default is true.

Example

The command:

```
    queue.devices;
```

produces a listing such as the following:

```
   Device      Protocol     Characteristics    State      Classes
 =========== ==========   ===============   ========   =======
 TERMINAL_40  XON_XOFF      Laser_Comm       Disabled   (none)
 TERMINAL_32  RTS                            Disabled   (none)
 TERMINAL_255 TELNET (postscript Ø,23)       Enabled    LP
```

This display shows three devices. Two are disabled and have no associated classes. TERMINAL_255, however, is enabled and is associated with class LP.

procedure Display

```
procedure Display (Class : Class_Name := "all");
```

Description

Displays the print requests currently queued in the specified class.

The display appears in the Current_Output window. If there are no queued requests, a message to this effect appears in the Message window.

The display shows the identification number for each request. Use the appropriate number as the Request_Id parameter when using the Cancel procedure.

Parameters

```
Class : Class_Name := "all";
```

Specifies the class for which the contents are to be displayed. The default is to display the contents of all classes. Users on installations that use Rational Networking—TCP/IP to connect multiple R1000 systems can query other machines on the network. Thus, the name can specify a machine name of the form !!*machine name*, where *machine name* is the name of a machine—for example, !!M1.

Example

The command:

```
queue.display
```

produces a display like the following in the Current_Output window:

```
ID   Time   State  Class    User          Object
==  =====  ======  =====  ========  ==================
20  17:54  Queued   LP    OPERATOR  !USERS.OPERATOR.LOG
```

In this example, the state of the print request is Queued because the device associated with the class LP is disabled. When a print request is currently being processed by an enabled device, the state of the request is Active.

References

procedure Cancel

---

procedure Print

```
procedure Print (Name    : String := "<IMAGE>";
                 Options : String := "<DEFAULT>";
                 Banner  : String := "<DEFAULT>";
                 Header  : String := "<DEFAULT>";
                 Footer  : String := "<DEFAULT>");
```

Description

Queues the specified objects for printing.

One or more objects can be specified by naming, by selection, or by placing the cursor in a window containing the object's images.

In addition to text files, Ada units and library listings can be printed. To print images from output windows or images of other kinds of objects (for example, a switch file), the user must first copy the object's image into a text file, commit the file, and then print that text file.

The default is to queue the print request to the device associated with the default class and to notify when the jobs are complete. The Message window echoes all print requests, unless the switches are set differently in the user's switch file.

The Print procedure uses the Options parameter to allow the user to change the printout format or to request multiple copies of the print request.

Parameters

`Name : String := "<IMAGE>";`

Specifies the name of the object to be sent to the print queue. This parameter can use special names and wildcards to specify a set of objects. The default is the current selection or image.

A text file, an Ada unit, or a library (that is, a list of the library's contents) can be specified. If a file or an Ada unit is specified using the current image, the most recently committed version is printed. Therefore, the printout will differ from the actual image on the screen if that image contains uncommitted changes.

`Options : String := "<DEFAULT>";`

Specifies options to be used in formatting output. The following is a list of the options available for use in the Options parameter to format output. Note that the Options parameter uses the special name "<DEFAULT>". When this special name is used, the vsystem looks in the session switch file for the options set in the Queue.Options switch. If the switch file is not accessible, the system uses the options "FORMAT=> (Wrap System_Header)". These options are no longer true and must be respecified if any of the options are changed by substituting an option for the "<DEFAULT>" special name. One of the following three options must be specified: Original_Raw, Raw, or Format. Unless otherwise specified in the Options parameter, the Boolean options Original_Raw, Raw, and Spool_Each_Item are false. The other options take the defaults specified below.

Banner_Page_User_Text=*string*

Specifies a string (with a maximum of 60 characters) that will be printed on the banner page, beneath the banner and above the system-generated information.

Class=*string*

Specifies the name of the class, where *string* is the class to which the print request is queued. The class determines the device that will handle the print request. Note that the specified class must exist and must be associated with an enabled device. If this option is not specified, the class is the default class.

Copies=*positive integer*

Specifies the number of copies to be printed, where *positive integer* is the number of copies to be printed. Copies are generated one at a time, and other jobs may intervene between copies. The default is one copy.

Format option

The Format option is an options parameter within the Options parameter that can be used to specify the format options Wrap, Truncate, Numbering, System_Header, Width, Length, and Tab_Width. Options must appear in parenthesis—for example, "FORMAT=>(WRAP WIDTH=77)".

Length=*positive integer*

Specifies the total number of printed lines per page, including headers and footers.

The default length is 60 lines. Note that, by default, the Rational Printer is set to eject a page after 66 lines if a formfeed is not encountered earlier. To print pages longer than 66 lines, you must change the appropriate setting on the Rational Printer (see "Printer Operations and Maintenance" in the *Rational R1000 Development System: System Manager's Guide*) in addition to increasing the Length value.

The number of lines in the body text for each page is automatically adjusted to accommodate any combination of a one-line header, a one-line footer, or a system page header. However, if you specify a multiple-line header or footer, you must decrease the Length value for every additional line beyond the expected one.

Numbering Boolean

Specifies whether to provide line numbering. The default is false.

Original_Raw Boolean

Specifies whether a copy of the file will be made. The options can be used on machines low on space when large files need to be printed. The option prints without using space to make the print spooler copy. The file can be spooled only to a local device. Each file is spooled separately (that is, the Spool_Each_Item option is ignored). A message is sent when printing is complete (that is, the Notify option is ignored). A banner is printed (that is, the Banner_Page_User option is ignored). The default is false.

System_Header Boolean

Specifies whether to print the system page header on each page. The system header is the name of the object and a page number. If there is a user-specified header, the system page header appears above it. The default is false.

Tab Width=***positive integer***

Specifies the number of spaces with which to replace a Tab character (Ascii-.Ht). The default is 8. A value of 0 specifies no replacement.

Truncate Boolean

Specifies whether to truncate lines that are longer than the Width option. The default is false. If both the Truncate and the Wrap options are set to true, Wrap is assumed to be true.

Width=***positive integer***

Specifies the maximum number of printable characters per line, where ***positive integer*** is the number of characters.

The default width is 80 columns. Note that the Rational Printer itself can be set to wrap after 80 columns. To print wider pages, the user must change the appropriate setting on the Rational Printer (see "Printer Operations and Maintenance" in the ***Rational R1000 Development System: System Manager's Guide***) in addition to increasing the value of Width.

The Wrap and Truncate options specify what to do with lines that are longer than Width.

Wrap Boolean

Specifies whether to wrap lines that are longer than the Width option. The default is false. If both the Truncate and the Wrap options are set to true, Wrap is assumed to be true. The wrapped portions of wrapped lines do not receive a new line number.

Notify literal

Specifies the manner of notification after a print request is completed. By default, an informative message is sent to the Message window. The available types are None, Message (the default), and Mail (reserved for future development). Remote requests, under normal conditions, will also notify the user.

`Raw` Boolean

Specifies whether the printer should interpret the input. This option prints the file without interpreting characters (that is, without recognizing formfeeds or linefeeds), which is useful for preformatted text or binary data. Using this option turns off other options. It does not provide a system or user header. The default is false.

`Spool_Each_Item` Boolean

Specifies whether to spool each file indicated by the Name parameter as a separate job. When true, each file has its own banner page. When false, a single banner page is printed. The default is false.

```
Banner : String := "<DEFAULT>";
```

Specifies the string that appears on the single banner page that precedes the printout. The string supplied by the user is truncated at 11 characters. If the null string is specified, a banner page will not be generated.

The special name `"<DEFAULT>"` refers to the banner specified in the username's session switch file or the username if one is not specified.

```
Header : String := "<DEFAULT>";
```

Specifies a line of text that appears at the top of each page of the printout. Any nonnull string (including blank characters) constitutes a user-specified header. A blank line is automatically inserted below the user-specified header to separate the header text from the printout.

If the Options parameter requests a system page header in addition to the user-specified header, the system header appears first, followed by the user-specified header.

The user-specified header can be longer than the Width option; however, a lengthy header is not wrapped automatically. The user must include linefeeds in a header if it should wrap onto multiple lines.

The number of lines in the body text for each page is automatically adjusted to accommodate any combination of a one-line header, a one-line footer, or a system page header. However, if a multiple-line header is specified, the Length value must be decreased for every header line beyond the expected one.

The special name `"<DEFAULT>"` refers to the header specified in the username's session switch file.

```
Footer : String := "<DEFAULT>";
```

Specifies a line of text that appears at the bottom of each page of the printout. Any nonnull string (including blank characters) constitutes a user-specified footer. A blank line is automatically inserted above the user-specified footer to separate the footer text from the printout.

The user-specified footer can be longer than the Width option; however, a lengthy footer is not wrapped automatically. The user must include linefeeds in a footer if it should wrap onto multiple lines.

The number of lines in the body text for each page is automatically adjusted to accommodate any combination of a one-line header, a one-line footer, or a system page header. However, if a multiple-line footer is specified, the Length value must be decreased for every footer line beyond the expected one.

The special name "<DEFAULT>" refers to the footer specified in the username's session switch file.

Example

The command:

```
    queue.print (name=>"output_samples", options=>"copies=2,truncate,
                        system_header", banner=>"dept Ø4",
                        header=>"May 9, 1987");
```

prints two copies of the object Output_Samples, with DEPT 04 on the banner page and the date appearing under the system page header. Lines longer than 80 characters are truncated.

The request to print Output_Samples is queued to the default class, and a message such as the following appears in the Message window:

```
    Request number 58 has been queued
```

A further message in the Message window notifies the user when the job is complete. The user is also notified if the request is made on a remote machine.

References

Session Switches

end Queue;

RATIONAL

package Search_List

When a command is executed, it must be compiled and run. Searchlists are used to determine how Ada names in commands are resolved—in other words, which libraries are to be searched for the name. A searchlist is a sequence of library names that are used in command name resolution on a per-session basis.

Since a library object name can use wildcard characters and therefore can resolve to many libraries, the resolution of a command name depends on the resolution of the library. Therefore, the order of the libraries is also important.

Package Search_List provides operations for creating, manipulating, changing, and deleting searchlists. It contains commands for changing searchlists from the user interface and commands for changing searchlists in programs.

The default searchlist, shown below, is searched for name resolution in the order in which the names appear. Each of the items in the searchlist is referred to as a *component* of that searchlist.

```
$`
!COMMANDS
!COMMANDS.ABBREVIATIONS`
!MACHINE.RELEASE.CURRENT.COMMANDS`
!IO
!TOOLS
```

When a command is searched for, the links in the current context are searched first. If the command is not resolved, then !Commands is searched. If the command is still unresolved, !Commands.Abbreviations and its links are searched, and so on until the command is either resolved or the searchlist ends. To resolve a name on a searchlist, the user must have read access to the name (otherwise, the name will appear to be undefined).

Note that the dollar symbol ($), meaning enclosing library, will resolve to a different library depending on the current context. The grave symbol (`) appearing after the searchlist components $, !Commands.Abbreviations, and !Machine.Release.Current-.Commands indicates that the set of links for the indicated world also should be searched.

If the session has no searchlist, the system uses the system default searchlist found in !Machine.Search_Lists.Default.

Procedures from Package !Commands.Common

Many of the operations in package !Commands.Common apply to searchlists. Searchlists can be brought into a window with the Search_List.Show_List procedure and then edited with common editing operations that apply to searchlists. The following procedures from package !Commands.Common apply to searchlists. Other operations from package !Commands.Common that do not apply to searchlists produce a message to that effect in the Message window when used on searchlists.

Note that names requiring searchlist resolution cannot be added to the searchlist.

When executing commands, the user can use the backslash () to force name resolution using a searchlist—for example, `Definition("\Access_List");`. The system will use the searchlist to find the package in !Commands.

procedure Common.Abandon

Ends the editing of the searchlist and removes the window from the screen. Since all changes to searchlists are done immediately, this procedure does not abandon any of those changes.

procedure Common.Commit

Commits changes to the searchlist. Since all changes to searchlists are made immediately and permanently, this procedure has no effect. All other operations on searchlists implicitly commit any changes.

procedure Common.Create_Command

Creates a Command window below the current window. The *use* clause in the Command window includes package Search_List, so operations in package Search_List are visible in the Command window without qualification. The *use* clause for searchlist windows is:

```
    use Editor, Search_List, Common;
```

procedure Common.Definition

Finds the definition of the component in the searchlist to which the cursor points. This procedure creates a window containing that component. The In_Place parameter allows the user to specify whether the new window will replace the current window. The Visible parameter specifies whether the specification or the body should be preferred.

procedure Common.Edit

Creates a Command window below the searchlist and places in it the command:

The New_Component parameter allows the user to specify the new searchlist component. The Old_Component parameter specifies the searchlist component to be replaced. The User and Session parameters specify the User and Session whose searchlist should be modified. The user must have read access to another user's home world to modify that user's searchlist.

procedure Common.Release

Ends the editing of the searchlist and removes the window from the screen.

procedure Common.Revert

Redraws the searchlist in the current window. If the searchlist has been changed by another user or program, this procedure redraws the list to ensure that the image is up to date.

procedure Common.Object.Child

Selects the component in the searchlist on which the cursor is located. If all components are already selected, the procedure selects the component on which the cursor is located. If a single component is already selected, the procedure has no effect. If no component is selected, the procedure selects the component on which the cursor is located.

procedure Common.Object.Delete

Deletes from the searchlist the selected component or the component on which the cursor is located.

procedure Common.Object.First_Child

Selects the first component in the searchlist.

procedure Common.Object.Insert

Creates a Command window and places in it the command:

```
Add (Component => "[STRING-expression]",
     Position => 1,
     Session => "",
     User => "");
```

where the first parameter is a string that can specify one or more components (separated with commas) and the second parameter is the position within the searchlist. Providing a value for the first parameter and promoting the command adds the specified component to the searchlist. The Session and User parameters allow the user to specify another session or username's searchlist to which to add an entry.

procedure Common.Object.Last_Child

Selects the last component in the searchlist.

procedure Common.Object.Move

Moves the selected searchlist component to the current cursor position in the current searchlist.

procedure Common.Object.Next

Selects the next component in the searchlist. If no component is already selected, the procedure selects the component on which the cursor is located. If all components are selected, this procedure produces an error.

procedure Common.Object.Parent

Selects the component in the searchlist on which the cursor is located. If no component is already selected, the procedure selects the component on which the cursor is located. If a component is selected, the procedure selects all components in the set. Otherwise, the procedure has no effect.

procedure Common.Object.Previous

Selects the previous component in the searchlist. If no component is already selected, the procedure selects the component on which the cursor is located. If all components are selected, this procedure produces an error.

procedure Add

```
procedure Add (Component  : String  := ">>LIBRARY NAME<<";
               Position   : Integer := Integer'Last;
               Session    : String  := "";
               User       : String  := "");
```

Description

Adds a component to the searchlist in the specified position for the specified session and user.

The procedure adds a component or components to the searchlist in the specified position. Multiple components are separated by commas in the parameter value.

Parameters

`Component : String := ">>LIBRARY NAME<<";`

Specifies the components to add to the searchlist. Multiple components (that is, library names) can be separated with commas. The default parameter placeholder `">>LIBRARY NAME<<"` must be replaced or an error will result.

`Position : Integer := Integer'Last;`

Specifies the position within the searchlist for the new component. The default places the new component at the end of the searchlist.

`Session : String := "";`

Specifies the session whose searchlist is to be modified. The default is the current session. Another session can be specified—for example, "S_2".

`User : String := "";`

Specifies the user whose searchlist is to be modified. The default is the current user. Another user can be specified—for example, "joe".

Errors

If a nonexistent username or session is specified, an error will result. It is possible to add searchlist entries without the required access to the object.

If an attempt to add a searchlist component to a nonexistent library is made, an error will result.

---

procedure Delete

```
procedure Delete (Component : String := "<SELECTION>";
                  Session   : String := "";
                  User      : String := "");
```

Description

Deletes the specified component from the searchlist of the specified session and user.

If a component is not named, the current selection is used only if it is on a searchlist.

Parameters

`Component : String := "<SELECTION>";`

Specifies the component to be deleted. The default is the selected component.

`Session : String := "";`

Specifies the session whose searchlist is to be modified. The default is the current session. Another session can be specified—for example, "S_2".

`User : String := "";`

Specifies the user whose searchlist is to be modified. The default is the current user. Another user can be specified—for example, "joe".

Errors

If the searchlist is not displayed on the screen and a component name is not specified for the Component parameter, nothing will be deleted.

If a nonexistent username or session is specified, an error will result.

procedure Display

```
procedure Display (Session : String := "";
                   User    : String := "");
```

Description

Displays the searchlist for the specified session and user to the current log file.

The procedure displays the searchlist in the log file that is, by default, the current output window. By default, the searchlist of the current session and user is displayed.

Parameters

`Session : String := "";`

Specifies the session whose searchlist is to be displayed. The default specifies the current session. Selection can be used to select a session name. Another session can be specified—for example, "S_2".

`User : String := "";`

Specifies the user whose searchlist is to be displayed. The default specifies the current user. Another user can be specified—for example, "joe".

Errors

If a nonexistent user or session is specified, an error will result.

procedure Display_Libraries

```
procedure Display_Libraries;
```

Description

Displays to the current output window the list of libraries to which the current searchlist resolves.

The procedure displays the resolution of the components on the current searchlist. The resolution uses the current context. The display appears in the current output file or window and is for the current username and session.

A searchlist uses Environment string names. A searchlist name can contain wildcards or context characters. When the searchlist is used to search for Ada units, these names are resolved in the current context to give a specific set of libraries to be searched and the order in which they should be searched. This command displays the result of performing searchlist resolution for the specified username and session name's searchlist in the current context. This may be useful in determining which libraries are actually being searched if an Ada unit's name is not found in a command or is not in the expected unit.

For example, if the current context is !Users.Fred and the searchlist is:

```
    $`
    !TOOLS
    !COMMANDS
```

the resulting display would be:

```
    !USERS.FRED
    !TOOLS
    !COMMANDS
```

procedure Release

```
procedure Release;
```

Description

Ends the editing of the searchlist.

The window is removed from the screen. Any changes made to the searchlist are made permanent.

procedure Replace

```
procedure Replace (New_Component : String := ">>LIBRARY NAME<<";
                   Old_Component : String := "<SELECTION>";
                   Session       : String := "";
                   User          : String := "");
```

Description

Replaces the specified component with the new component in the searchlist of the specified session and user.

Parameters

`New_Component : String := ">>LIBRARY NAME<<";`

Specifies the new component to replace the old component. The default parameter placeholder `">>LIBRARY NAME<<"` must be replaced or an error will result.

`Old_Component : String := "<SELECTION>";`

Specifies the old component to be replaced. The default specifies the currently selected component.

`Session : String := "";`

Specifies the session whose searchlist is to be modified. The default is the current session. Another session can be specified—for example, "S_2".

`User : String := "";`

Specifies the user whose searchlist is to be modified. The default is the current user. Another user can be specified—for example, "joe".

Errors

If a nonexistent user or session is specified, an error will result.

procedure Reset_To_System_Default

```
procedure Reset_To_System_Default (Session : String := "";
                                   User    : String := "");
```

Description

Resets the searchlist of the specified session and user to the system default.

The procedure destroys the current contents of the searchlist of the specified session and user and replaces the contents with the system default searchlist, shown below:

```
$`
!COMMANDS
!COMMANDS.ABBREVIATIONS`
!MACHINE.RELEASE.CURRENT.COMMANDS`
!IO
!TOOLS
```

This default comes from file !Machine.Search_Lists.Default.

Parameters

```
Session : String := "";
```

Specifies the session whose searchlist is to be reset. The default is the current session. Another session can be specified—for example, "S_2".

```
User : String := "";
```

Specifies the user whose searchlist is to be reset. The default is the current user. Another user can be specified—for example, "joe".

Errors

If a nonexistent user or session is specified, an error will result.

procedure Revert

```
procedure Revert (File_Name : String := "";
                  Session   : String := "";
                  User      : String := "");
```

Description

Replaces the searchlist for the specified session and user with the contents of the specified file.

Searchlists can be saved in files with the Save procedure. The Revert procedure replaces the current searchlist with the searchlist in the specified file.

Parameters

`File_Name : String := "";`

Specifies the file from which to revert the searchlist. The default is the permanent searchlist that is part of the user's session.

`Session : String := "";`

Specifies the session whose searchlist is to be reverted. The default is the current session. Another session can be specified—for example, "S_2".

`User : String := "";`

Specifies the user whose searchlist is to be reverted. The default is the current user. Another user can be specified—for example, "joe".

Errors

If a nonexistent user or session is specified, an error will result.

References

procedure Save

procedure Save

```
procedure Save (File_Name : String := ">>FILE NAME<<";
                 Session   : String := "";
                 User      : String := "");
```

Description

Saves the searchlist for the specified session and user in a file.

The Revert procedure replaces the current searchlist with the searchlist in a specified file.

Parameters

`File_Name : String := ">>FILE NAME<<";`

Specifies the file in which to save the searchlist. The default parameter placeholder `">>FILE NAME<<"` must be replaced or an error will result.

`Session : String := "";`

Specifies the session whose searchlist is to be saved. The default is the current session. Another session can be specified—for example, "S_2".

`User : String := "";`

Specifies the user whose searchlist is to be saved. The default is the current user. Another session can be specified—for example, "joe".

Errors

If a nonexistent user or session is specified, an error will result.

References

procedure Revert

procedure Set_Up

```
procedure Set_Up (Component  : String := ">>SEARCH LIST<<";
                  Session    : String := "";
                  User       : String := "");
```

Description

Replaces the entire searchlist with the specified component(s).

The procedure sets up the searchlist with the specified component(s) supplied as a string. The previous searchlist is destroyed. Multiple components for the new searchlist are separated by commas.

Parameters

`Component :  String := ">>SEARCH LIST<<";`

Specifies the component(s) with which to set up the searchlist. Multiple components in the string are separated by commas. The default parameter placeholder `">>SEARCH LIST<<"` must be replaced or an error will result.

`Session :  String := "";`

Specifies the session whose searchlist is to be changed. The default is the current session. Another session can be specified—for example, "S_2".

`User :  String := "";`

Specifies the user whose searchlist is to be changed. The default is the current user. Another user can be specified—for example, "joe".

Errors

If a nonexistent user or session is specified, an error will result.

procedure Show_Item

```
procedure Show_Item (Component : String := "<CURSOR>");
```

Description

Finds the definition of the specified component from the searchlist.

The procedure creates a window and displays the specified component from the searchlist. The current context is used to resolve the context-dependent components. If no component is named or selected, the component on which the cursor is located is used.

Parameters

```
Component : String := "<CURSOR>";
```

Specifies the component for which to find the definition. The default is the component on which the cursor is located.

procedure Show_List

```
procedure Show_List (Session : String := "";
                     User    : String := "");
```

Description

Creates a window in which the searchlist can be edited for the specified session and user.

The procedure creates a window and displays the searchlist for the specified session and user in that window. From the window, the searchlist can be edited with many operations from package !Commands.Common that apply to the window as well as other operations in this package (see the introduction to this package for details).

Parameters

`Session :  String := "";`

Specifies the session whose searchlist is to be modified. The default is the current session. Selection can be used to specify another session. Another session can be specified—for example, "S_2".

`User :  String := "";`

Specifies the user whose searchlist is to be modified. The default is the current user. Another user can be specified—for example, "joe".

Errors

If a nonexistent user or session is specified, an error will result.

end Search_List;

RATIONAL

Session Switches

This section describes the functionality of session switches, which control the way the system behaves on the user's terminal. For information on commands for manipulating session switches, see LM, package Switches. Library switches are also documented in LM, package Switches.

Overview of Switches

The two kinds of switches are: *library switches* and *session switches.* Library switches are defined for a specific library (that is, a directory or world). A set of these switches is associated with a particular library. These switches affect how compilation is done, how links are managed, or how pretty-printing is done in that library. Library switches are described in LM, package Switches.

Session switches affect the way the system behaves on the user's terminal. The actual switches are documented in this package, but the mechanism for displaying switches is available through the operations in LM, package Switches.

Switches are stored in File class objects with subclass Switch. Each switch file contains all the switches for a given session or library. A session switch file is located in the user's home directory. The user can have several switch files, one for each session. Some session switches are read by the Environment only when the user logs in. Others are effective immediately. Still others are checked when some window is created.

Editing Session Switch Files

A session switch file can be edited using the !Commands.Switches.Edit_Session_Attributes command. After the Switches.Edit_Session_Attributes command is entered, the Environment opens a window showing the list of switches. To change a switch, the user places the cursor on the line displaying the switch, selects it by pressing [Object] - [←], and then presses [Edit]. If the switch has a Boolean value, the value will be toggled. If the switch takes a non-Boolean value, a Command window will be opened with prompts for inserting the new value.

To get help on a switch, the user enters the switch file as described previously, places the cursor on the line of the switch in question, and presses [Explain]. A Help file appears on the line below the switches.

The following list names all session (user) switches, grouped by functionality. Following the list are descriptions of the session switches, ordered alphabetically.

Session Switches Grouped by Function

Switches for Editing Images

Beep_On_Errors	Beep_On_Interrupt
Beep_On_Messages	Cursor_Bottom_Offset
Cursor_Left_Offset	Cursor_Right_Offset
Cursor_Top_Offset	Cursor_Transpose_Moves
Image_Fill_Column	Image_Fill_Extra_Space
Image_Fill_Indent	Image_Fill_Mode
Image_Fill_Prefix	Image_Insert_Mode
Image_Tab_Stops	Key_Directory
Prompt_Delimiters	Screen_Dump_File
Search_Ignore_Case	Search_Preserve_Case
Search_Regular_Expr	Window_Command_Size
Window_Frames	Window_Frames_Startup
Window_Have_Sides	Window_Is_Staggered
Window_Message_Life	Window_Message_Size
Window_Scroll_Overlap	Window_Shift_Overlap
Window_Word_Breaks	

Switches for Ada Units

Default_Job_Page_Limit	Notify_Warnings

Switches for Debugging

Debug_Addresses	Debug_Break_At_Creation
Debug_Declaration_Display	Debug_Delete_Temporary_Breaks
Debug_Display_Count	Debug_Display_Creation
Debug_Display_Level	Debug_Echo_Commands
Debug_Element_Count	Debug_First_Element
Debug_Freeze_Tasks	Debug_History_Count
Debug_History_Entries	Debug_History_Start
Debug_Include_Packages	Debug_Interpret_Control_Words
Debug_Kill_Old_Jobs	Debug_Machine_Level
Debug_Memory_Count	Debug_No_History_Timestamps
Debug_Optimize_Generic_History	Debug_Permanent_Breakpoints
Debug_Pointer_Level	Debug_Put_Locals
Debug_Qualify_Stack_Names	Debug_Require_Debug_Off
Debug_Save_Exceptions	Debug_Show_Location
Debug_Stack_Count	Debug_Stack_Start
Debug_Timestamps	

Switches for Text I/O

Text_Bottom_Stripe	Text_Convert_Tabs
Text_Header	Text_Print_Name
Text_Print_Number	Text_Print_Time
Text_Reuse_Window	Text_Scroll_Output
Text_Top_Stripe	

Switches for Networking

Account	Auto_Login
Escape	Escape_On_Break
Password	Prompt_For_Account
Prompt_For_Password	Remote_Directory
Remote_Machine	Remote_Passwords
Remote_Roof	Remote_Sessions
Remote_Type	Send_Port_Enabled
Transfer_Mode	Transfer_Structure
Transfer_Type	Username

Switches for Library Display

Library_Break_Long_Lines	Library_Capitalize
Library_Indentation	Library_Lazy_Realignment
Library_Line_Length	Library_Misc_Show_Edit_Info
Library_Misc_Show_Frozen	Library_Misc_Show_Retention
Library_Misc_Show_Size	Library_Misc_Show_Subclass
Library_Misc_Show_Unit_State	Library_Misc_Show_Volume
Library_Shorten_Names	Library_Shorten_Subclass
Library_Shorten_Unit_State	Library_Show_Deleted_Objects
Library_Show_Deleted_Versions	Library_Show_Miscellaneous
Library_Show_Standard	Library_Show_Subunits
Library_Show_Version_Number	Library_Std_Show_Class
Library_Std_Show_Subclass	Library_Std_Show_Unit_State
Library_Uppercase	

Switches for Log and Profile Operations

Activity_File	Job_Context_First
Job_Context_Length	Job_Name_Length
Job_Name_Separator	Log_At_Sign_Msgs
Log_Auxiliary_Msgs	Log_Diagnostic_Msgs
Log_Dollar_Msgs	Log_Error_Msgs
Log_Exception_Msgs	Log_File
Log_Line_Width	Log_Negative_Msgs
Log_Note_Msgs	Log_Position_Msgs
Log_Positive_Msgs	Log_Prefix_1
Log_Prefix_2	Log_Prefix_3
Log_Sharp_Msgs	Log_Warning_Msgs
Reaction	

Switches for Printing

Banner	Footer
Header	Options

Descriptions of Session Switches

Account

Specifies an account name to set up an FTP connection between a local and a remote computer. The account is on the remote computer. The default is the null string. The full switch name is Session_Ftp.Account.

Activity_File

Specifies a filename for a default activity file for the session. The default is Release.Current_Activity. Changes to this switch take effect immediately. The full switch name is Profile.Activity_File. See Project Management (PM) for further information.

Auto_Login

Specifies that, when an FTP connection is established, FTP should automatically log the user into the remote machine. The default is false. The full switch name is Session_Ftp.Auto_Login.

Banner

Controls what the print spooler prints on the banner page of a printout. The null string results in no banner page being printed. The default is <User_Id>. The full switch name is Queue.Banner.

Beep_On_Errors

Sets a Boolean value that specifies whether the Environment should make the terminal beep or flash (depending on terminal setup characteristics) when the Environment discovers an error. The default is true. Changes to this switch take effect immediately. The full switch name is Session.Beep_On_Errors.

Beep_On_Interrupt

Sets a Boolean value that specifies whether the Environment should make the terminal beep or flash (depending on terminal setup characteristics) when the current job is interrupted with the [Control][G] combination. If set to true, the value of Session.Beep_On_Errors must also be set to true to perform the beep. The default is true. Changes to this switch take effect immediately. The full switch name is Session.Beep_On_Interrupt.

Beep_On_Messages

Sets a Boolean value that specifies whether the Environment should make the terminal beep or flash (depending on terminal setup characteristics) when a new message appears in the Message window. If set to true, the value of Session.Beep_On_Errors must also be set to true to perform the beep. The default is false. Changes to this switch take effect immediately. The full switch name is Session.Beep_On_Messages.

Cursor_Bottom_Offset

Represents, as an integer value, the percentage of the cursor's position relative to the bottom of the window. When the cursor is moved off the bottom edge of a window, the image in the window is scrolled so that the cursor is this percentage from the bottom of the window. Values of this switch should be in the range of 1 through 99. The default is 33. Changes to this switch take effect immediately. The full switch name is Session.Cursor_Bottom_Offset.

Cursor_Left_Offset

Represents, as an integer value, the percentage of the cursor's position relative to the left edge of the window. When the cursor is moved off the left edge of a window, the image in the window is scrolled so that the cursor is this percentage from the left edge of the window. Values of this switch should be in the range of 1 through 99. The default is 33. Changes to this switch take effect immediately. The full switch name is Session.Cursor_Left_Offset.

Cursor_Right_Offset

Represents, as an integer value, the percentage of the cursor's position relative to the right edge of the window. When the cursor is moved off the right edge of a window, the image in the window is scrolled so that the cursor is this percentage from the right edge of the window. Values of this switch should be in the range of 1 through 99. The default is 33. Changes to this switch take effect immediately. The full switch name is Session.Cursor_Right_Offset.

Cursor_Top_Offset

Represents, as an integer value, the percentage of the cursor's position relative to the top of the window. When the cursor is moved off the top edge of a window, the image in the window is scrolled so that the cursor is this percentage from the top of the window. Values of this switch should be in the range of 1 through 99. The default is 33. Changes to this switch take effect immediately. The full switch name is Session.Cursor_Top_Offset.

Cursor_Transpose_Moves

Sets a Boolean value that specifies whether to advance the cursor on a transpose operation. This switch is examined by Character, Word, Line, or Window transpose operations. The default is false, which maintains the cursor position on a transpose operation. Changes to this switch take effect immediately. The full switch name is Session.Cursor_Transpose_Moves.

Debug_Addresses

Specifies whether machine information should be put in displays produced by the Stack, Task_Display, Information, and Trace procedures in package !Commands-.Debug. The default is false. The full switch name is Session.Debug_Addresses.

Debug_Break_At_Creation

Specifies whether the equivalent of a breakpoint should be placed at the point where new tasks begin elaboration. The default is false. The full switch name is Session.Debug_Break_At_Creation.

Debug_Declaration_Display

Specifies whether all declarations should be displayed when source code is listed by means of the !Commands.Debug.Display procedure. The default is true. The full switch name is Session.Debug_Declaration_Display.

Debug_Delete_Temporary_Breaks

Specifies whether each temporary breakpoint should be deleted once its conditions are met and execution has stopped. The default is false. The full switch name is Session.Debug_Delete_Temporary_Breaks.

Debug_Display_Count

Specifies the default value of the Count parameter in the !Commands.Debug.Display procedure. The default is 10. The full switch name is Session.Debug_Display_Count.

Debug_Display_Creation

Specifies whether a tracelike display of the creation of each task is performed. The default is false. The full switch name is Session.Debug_Display_Creation.

Debug_Display_Level

Specifies the number of levels to expand complex data structures in the !Commands.Debug.Put procedure. The default is 3. The full switch name is Session.Debug_Display_Level.

Debug_Echo_Commands

Specifies whether commands are echoed in the Debugger window. The default is true. The full switch name is Session.Debug_Echo_Commands.

Debug_Element_Count

Specifies the maximum number of elements in any array that are displayed by means of the !Commands.Debug.Put procedure. The default is 25. The full switch name is Session.Debug_Element_Count.

Debug_First_Element

Specifies the offset, relative to the array's first element, of the first element that is displayed by means of the !Commands.Debug.Put procedure. The default is 0. The full switch name is Session.Debug_First_Element.

Debug_Freeze_Tasks

Specifies whether all other tasks should attempt to stop when a task is stopped in the Rational Debugger. The default is false. The full switch name is Session.Debug_Freeze_Tasks.

Debug_History_Count

Specifies the default value for the Count parameter to the !Commands.Debug.History_Display procedure. The default is 10. The full switch name is Session.Debug_History_Count.

Debug_History_Entries

Specifies the maximum number of history entries that the Rational Debugger will keep. This switch is not currently implemented. The default is 1,000. The full switch name is Session.Debug_History_Entries.

Debug_History_Start

Specifies the oldest history entry to be displayed in the !Commands.Debug.History_Display procedure. The default is 10. The full switch name is Session.Debug_History_Start.

Debug_Include_Packages

Specifies whether packages that have completed elaboration should be included in the output of the !Commands.Debug.Task_Display procedure. The default is false. The full switch name is Session.Debug_Include_Packages.

Debug_Interpret_Control_Words

Specifies whether the !Commands.Debug.Memory_Dump procedure should interpret control stacks when they are displayed. The default is false. The full switch name is Session.Debug_Interpret_Control_Words.

Debug_Kill_Old_Jobs

Specifies whether the last program being debugged is killed when a new program is being debugged. The default is true. The full switch name is Session.Debug_Kill_Old_Jobs.

Debug_Machine_Level

Specifies whether certain machine-level operations are allowed. This switch is not currently implemented. The default is false. The full switch name is Session.Debug_Machine_Level.

Debug_Memory_Count

Specifies the default value of the Count parameter in the !Commands.Debug.Memory_Dump procedure. The default is 3. The full switch name is Session.Debug_Memory_Count.

Debug_No_History_Timestamps

Specifies whether timestamps are displayed with each history entry created by the !Commands.Debug.History_Display procedure. The default is true. The full switch name is Session.Debug_No_History_Timestamps.

Debug_Optimize_Generic_History

Specifies whether the generic's instance, or just the generic itself, is recorded when taking history. The default is true, meaning that the instance is not recorded, which causes history-taking to run considerably faster. The full switch name is Session.Debug_Optimize_Generic_History.

Debug_Permanent_Breakpoints

Specifies whether breakpoints are permanent (must be explicitly deactivated or deleted) or temporary (are deactivated or deleted when the breakpoint first causes execution to stop). The default is true, which specifies permanent breakpoints. The full switch name is Session.Debug_Permanent_Breakpoints.

Debug_Pointer_Level

Specifies the level of pointer values to be expanded in the display produced by the !Commands.Debug.Put procedure. The default is 3. The full switch name is Session.Debug_Pointer_Level.

Debug_Put_Locals

Specifies whether the !Commands.Debug.Put procedure on a subprogram or packages results in a display of local variables as well as parameters. The default is false. The full switch name is Session.Debug_Put_Locals.

Debug_Qualify_Stack_Names

Specifies, when true, that the names displayed by the !Commands.Debug.Stack procedure are fully qualified. When false, the names displayed are the simple names of the subprograms executing in each frame. The default is false. The full switch name is Session.Debug_Qualify_Stack_Names.

Debug_Require_Debug_Off

Specifies, when true, that the current job being debugged cannot be aborted simply by starting debugging on a new job. To start debugging a new job, debugging must be stopped on the current job by completing its execution or by explicitly executing the !Commands.Debug.Debug_Off procedure to release or abort the job. The default is false. The full switch name is Session.Debug_Require_Debug_Off.

Debug_Save_Exceptions

Specifies whether exception-handling information (from the Catch and Propagate procedures in the !Commands.Debug package) should be saved from run to run. The default is false. The full switch name is Session.Debug_Save_Exceptions.

Debug_Show_Location

Specifies whether the current location in the source should be displayed in an Ada window with the current location being executed highlighted. The default is true. The full switch name is Session.Debug_Show_Location.

Debug_Stack_Count

Specifies the default value of the Count parameter in the !Commands.Debug.Stack procedure. The default is 10. The full switch name is Session.Debug_Stack_Count.

Debug_Stack_Start

Specifies the default starting stack frame in the Stack procedure. The default is 1. The full switch name is Session.Debug_Stack_Start.

Debug_Timestamps

Specifies whether a time stamp is displayed with each command and task stop. The default is false. The full switch name is Session.Debug_Timestamps.

Default_Job_Page_Limit

Controls the page limit for a job when no other switches or pragmas apply to that job. The default is 8,000. The full switch name is Session.Default_Job_Page_Limit.

Escape

Sets the escape sequence for returning to the local environment from a Telnet session. The escape sequence can be any nonnull character string. This switch can be used in conjunction with the Escape_On_Break switch. The default is null. The full switch name is Telnet.Escape.

Escape_On_Break

Specifies, when true, that a break signal will cause a return to the local environment from a Telnet session. This switch can be used in conjunction with the Escape switch. The default is true. The full switch name is Telnet.Escape_On_Break.

Footer

Controls the user-supplied text to be printed on the bottom of each page of a printout. The default is the null string, which prints nothing. The full switch name is Queue.Footer.

Header

Controls the user-supplied text to be printed on the top of each page of a print-out. The default is the null string, which prints nothing. The full switch name is Queue.Header.

Image_Fill_Column

Specifies, as an integer value, the column the image will fill. The fill column for newly created images is specified by this value. The fill column for individual images can be specified with the !Commands.Editor.Set.Fill_Column procedure. The default is column 72. Changes to this switch take effect when a new image is created. The full switch name is Session.Image_Fill_Column.

Image_Fill_Extra_Space

Specifies a set of characters after which the Environment cannot compress extra space when justifying an image. Changes to this switch take effect when the !Commands.Editor.Region.Justify procedure is used. The default is these three characters: exclamation (!), period (.), and question mark (?). The full switch name is Session.Image_Fill_Extra_Space.

Image_Fill_Indent

Sets the number of spaces to indent from the left column when doing a region fill operation. The default is −1, which sets the indentation to be that of the first line. The full switch name is Session.Image_Fill_Indent.

Image_Fill_Mode

Sets a Boolean value that specifies whether the image fills automatically. The fill mode for newly created images is specified by this value. The fill mode for individual images can be specified with the !Commands.Editor.Set.Fill_Mode procedure. The default is false. Changes to this switch affect new images. The full switch name is Session.Image_Fill_Mode.

Image_Fill_Prefix

Specifies a text string value that prefixes each line created as the Environment fills or justifies an image. An example is setting this string to the comment delimiter (--), which makes each line that is justified a comment line. The value is used with the !Commands.Editor.Region.Fill procedure or the !Commands.Editor.Region.Justify procedure when the string parameter is null. The default is null. Changes to this switch affect subsequent uses of the Editor.Region.Fill procedure or the Editor.Region.Justify procedure. The full switch name is Session.Image_Fill_Prefix.

Image_Insert_Mode

Sets a Boolean value that specifies whether text entered in a newly created image is in insert mode or overwrite mode. The insert mode of individual images can be changed with the Editor.Set.Insert_Mode procedure. The default is true, which specifies insert mode. Changes to this switch take effect when a new image is created. The full switch name is Session.Image_Insert_Mode.

Image_Tab_Stops

Specifies a text string value that indicates the initial tab stops in an image. The string should contain only spaces and vertical bars to indicate the tab stops. Individual tab stops can be changed with the Editor.Set.Tab_On procedure or the Editor.Set.Tab_Off procedure. Tabs are unique in the first 160 columns; thereafter the tab stops sequence is repeated. The default is no tab stops. Changes to this switch take effect only at login. The full switch name is Session.Image_Tab_Stops.

Job_Context_First

Sets a Boolean value that specifies whether the job's context appears first in the header (true) or the job's name appears first in the header (false). The default is true. Changes to this switch take effect when a new job starts. The full switch name is Session.Job_Context_First.

Job_Context_Length

Specifies, as an integer value, the number of name segments displayed in the job's context portion of the job header. The default is 3. Changes to this switch take effect when a new job starts. The full switch name is Session.Job_Context_Length.

Job_Name_Length

Specifies, as an integer value, the number of name segments displayed in the job's name portion of the job header. The default is 2. Changes to this switch take effect when a new job starts. The full switch name is Session.Job_Name_Length.

Job_Name_Separator

Specifies a text string value that provides a separator between the job's context and the job's name portions of the job header. The default is % (percent). Changes to this switch take effect when a new job starts. The full switch name is Session.Job_Name_Separator.

Key_Directory

Specifies a text string value that resolves to the world in which a user's own set of key bindings and macros should be found. The default is the user's home world. Changes to this switch take effect only at login. The full switch name is Session.Key_Directory.

Library_Break_Long_Lines

Controls whether lines that exceed the value of the Library_Line_Length session switch are broken. The default is true. The full switch name is Session.Library_Break_Long_Lines.

Library_Capitalize

Determines whether identifiers in library images are capitalized. The default is true. The full switch name is Session.Library_Capitalize.

Library_Indentation

Specifies how much to indent subunit names displayed in the short form (that is, without their parent name as prefixes determined by the value of the Library_Shorten_Names session switch). The default is 2. The full switch name is Session.Library_Indentation.

Library_Lazy_Realignment

Determines whether the image is realigned when a longer name occurs. If true, it waits for a Redraw request. The default is true. The full switch name is Session.Library_Lazy_Realignment.

Library_Line_Length

Determines how long a line can be before it is eligible to be broken. The default is 80. The full switch name is Session.Library_Line_Length.

Library_Misc_Show_Edit_Info

Shows time and user of last update/edit for the version when displaying miscellaneous information. The default is true. The full switch name is Session.Library_Misc_Show_Edit_Info.

Library_Misc_Show_Frozen

Shows "Frz" for frozen objects when displaying miscellaneous information. The default is true. The full switch name is Session.Library_Misc_Show_Frozen.

Library_Misc_Show_Retention

Shows the retention count (for example, 10) when displaying miscellaneous information. The default is true. The full switch name is Session.Library_Misc_Show_Retention.

Library_Misc_Show_Size

Shows the size of the version in bytes when displaying miscellaneous information. The default is true. The full switch name is Session.Library_Misc_Show_Size.

Library_Misc_Show_Subclass

Shows the object's subclass when displaying miscellaneous information. The default is false. The full switch name is Session.Library_Misc_Show_Subclass.

Library_Misc_Show_Unit_State

Shows the shortened form of the unit state of Ada units when displaying miscellaneous information. The default is true. The full switch name is Session.Library_Misc_Show_Unit_State.

Library_Misc_Show_Volume

Shows the volume for libraries when displaying miscellaneous information. The default is true. The full switch name is Session.Library_Misc_Show_Volume.

Library_Shorten_Names

Determines whether the switch name of the parent is displayed for subunits. The default is true. The full switch name is Session.Library_Shorten_Names.

Library_Shorten_Subclass

Shows subclasses in shortened form. The default is true. The full switch name is Session.Library_Shorten_Subclass.

Library_Shorten_Unit_State

Shows only the first letter of the unit state. The default is true. The full switch name is Session.Library_Shorten_Unit_State.

Library_Show_Deleted_Objects

Shows deleted objects (for example, {Foo'Body}). Display of deleted objects is controlled by elision. The default is false. The full switch name is Session.Library-_Show_Deleted_Objects.

Library_Show_Deleted_Versions

Shows version numbers and information for all versions of an object. Display of version numbers is controlled by elision. The default is false. The full switch name is Session.Library_Show_Deleted_Version.

Library_Show_Miscellaneous

Shows miscellaneous information. Miscellaneous information is obtained by using the !Commands.Common.Explain command. The default is false. The full switch name is Session.Library_Show_Miscellaneous.

Library_Show_Standard

Shows standard information. Standard information is obtained by using the !Commands.Common.Explain command. The default is false. The full switch name is Session.Library_Show_Standard.

Library_Show_Subunits

Shows subunits in the initial display. Display of subunits is controlled by elision. The default is true. The full switch name is Session.Library_Show_Subunits.

Library_Show_Version_Number

Shows the version number of the default or maximum version as part of the object name (for example, Foo'V(4) or Bar'V(2)). The default is false. The full switch name is Session.Library_Show_Version_Number.

Library_Std_Show_Class

Shows class along with subclass—for example, File (Text) instead of Text—as part of the standard information display. The default is true. The full switch name is Session.Library_Std_Show_Class.

Library_Std_Show_Subclass

Shows subclass as part of the standard information display. The default is true. The full switch name is Session.Library_Std_Show_Subclass.

Library_Std_Show_Unit_State

Shows unit state for Ada units as part of the standard information display. The default is false. The full switch name is Session.Library_Std_Show_Unit_State.

Library_Uppercase

Determines whether identifiers in library images are uppercased. The default is false. The full switch name is Session.Library_Uppercase.

Many users prefer a default library image that appears as follows:

```
!Users.Lance :   Library (World);
  A_Generic_Instantiation  : I Ada (Pack_Inst);
  A_Generic_Package        : I Ada (Gen_Pack);
  A_Generic_Package        : I Ada (Pack_Body);
  A_Package                : I Ada (Pack_Spec);
  A_Package                : I Ada (Pack_Body);
    .Nested                : I Ada (Pack_Body);
    .T                     : I Ada (Task_Body);
  A_Procedure              : C Ada (Proc_Spec);
  A_Procedure              : C Ada (Proc_Body);
    ._Ada_1_               : S Ada (Statement);
  Control_Link             :   Pipe;
  Current_Release          :   File (Activity);
  File_From_Direct_Io      :   File (Binary);
  File_From_Text_Io        :   File (Text);
  . . .
```

To establish this as your default way of viewing libraries, modify the following session switch values:

- Library_Show_Standard := true
- Library_Std_Show_Unit_State := true

This causes the standard additional information to be displayed by default and unit state information to be added to this standard information.

Log_At_Sign_Msgs

Sets a Boolean value in the session profile that specifies whether messages denoted with *at* signs (@) should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_At_Sign_Msgs.

Log_Auxiliary_Msgs

Sets a Boolean value in the session profile that specifies whether auxiliary messages should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Auxiliary_Msgs.

Log_Diagnostic_Msgs

Sets a Boolean value in the session profile that specifies whether diagnostic messages should be put in the log file. The default is false. Changes to this switch take effect immediately. The full switch name is Profile.Log_Diagnostic_Msgs.

Log_Dollar_Msgs

Sets a Boolean value in the session profile that specifies whether messages denoted with dollar signs ($) should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Dollar_Msgs.

Log_Error_Msgs

Sets a Boolean value in the session profile that specifies whether error messages should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Error_Msgs.

Log_Exception_Msgs

Sets a Boolean value in the session profile that specifies whether exception messages should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Exception_Msgs.

Log_File

Specifies which log file to use by default for the current session. The default is Use_Output (which is Io.Current_Output). Other allowable values are Use_Error (which is Io.Current_Error), Use_Standard_Output (which is Io.Standard_Output), and Use_Standard_Error (which is Io.Standard_Error). The full switch name is Profile.Log_File.

Log_Line_Width

Specifies as a natural value in the session profile how wide to format the messages for the log file. The default value is 77 columns. Changes to this switch take effect immediately. The full switch name is Profile.Log_Line_Width.

Log_Negative_Msgs

Sets a Boolean value in the session profile that specifies whether negative messages should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Negative_Msgs.

Log_Note_Msgs

Sets a Boolean value in the session profile that specifies whether notes should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Note_Msgs.

Log_Position_Msgs

Sets a Boolean value in the session profile that specifies whether position messages should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Position_Msgs.

Log_Positive_Msgs

Sets a Boolean value in the session profile that specifies whether positive messages should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Positive_Msgs.

Log_Prefix_1

Specifies a value in the session profile of Profile.Log_Prefix type that indicates the first prefix to a log message. The default is the year/month/day form of the date. Changes to this switch take effect immediately. The full switch name is Profile.Log_Prefix_1.

Log_Prefix_2

Specifies a value in the session profile of Profile.Log_Prefix type that indicates the second prefix to a log message. The default is the hour/minute/second form of the time. Changes to this switch take effect immediately. The full switch name is Profile.Log_Prefix_2.

Log_Prefix_3

Specifies a value in the session profile of Profile.Log_Prefix type that indicates the third prefix to a log message. The default is the symbols that identify the kind of message. Changes to this switch take effect immediately. The full switch name is Profile.Log_Prefix_3.

Log_Sharp_Msgs

Sets a Boolean value in the session profile that specifies whether messages denoted by sharp signs (#) should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Sharp_Msgs.

Log_Warning_Msgs

Sets a Boolean value in the session profile that specifies whether warning messages should be put in the log file. The default is true. Changes to this switch take effect immediately. The full switch name is Profile.Log_Warning_Msgs.

Notify_Warnings

Sets a Boolean value that specifies whether minor errors that appear in Ada units should be displayed (underlined) with the other errors. The default is false. Changes made to this switch take effect only at login. The full switch name is Session.Notify-_Warnings.

Options

Specifies printing options. (For further information, see SMU, package Queue, procedures Pring and Print_Version.) The default specifies the Wrap and System_Header options. The full switch name is Queue.Options.

Password

Specifies the user's remote password for logging into a remote computer over an FTP connection. The default is null. The full switch name is Session_Ftp.Password.

Prompt_Delimiters

Specifies a set of characters that are not deleted when a prompt is overwritten. The defaults are the double-quote (") or single-quote (') characters. Changes to this switch take effect immediately. The full switch name is Session.Prompt_Delimiters.

Prompt_For_Account

Causes FTP to prompt the user to supply a user account on the connected remote computer when FTP operations are executed. The user is prompted only if the Account switch is null. The default is false. The full switch name is Session_Ftp-.Prompt_For_Account.

Prompt_For_Password

Causes FTP to prompt the user to supply a password for the connected remote computer when FTP operations are executed. The user is prompted only if the Account switch is null. The default is false. The full switch name is Session-_Ftp.Prompt_For_Password.

Reaction

Specifies a value in the session profile of Profile.Error_Reaction type that is part of the default session profile. The default is the value Persevere. Changes to this switch take effect immediately. The full switch name is Profile.Reaction.

Recovery_Locality

Currently not implemented.

Remote_Directory

Sets a remote directory to which to connect when establishing an FTP connection to a remote computer. The default is null. The full switch name is Session_Ftp.Remote_Directory.

Remote_Machine

Specifies the name of the remote computer to which Telnet is to connect. The default is null. The full switch name is Session_Ftp.Remote_Machine.

Remote_Machine

Specifies the name of the remote computer to which FTP is to connect. The default is null. The full switch name is Telnet.Remote_Machine.

Remote_Passwords

Specifies a file in the session profile containing passwords to be used in networking with other machines. The default is no file. The full switch name is Profile.Remote_Passwords.

Remote_Roof

Specifies a directory on a remote computer connected to a local computer through an FTP connection. This directory is an ancestor directory for a group of files that FTP is to transfer. The default is the null string. The full switch name is Session_Ftp.Remote_Roof.

Remote_Sessions

Specifies a file in the session profile containing sessions to be used in networking with other machines. The full switch name is Profile.Remote_Sessions.

Remote_Type

Specifies the type of the remote computer to which FTP is to connect. The default is "Rational". The full switch name is Session_Ftp.Remote_Type.

Screen_Dump_File

Specifies a filename for the file created by the !Commands.Editor.Screen.Dump command. The default is "SCREEN_DUMP". The full switch name is Session.Screen_Dump_File.

Search_Ignore_Case

Sets a Boolean value that specifies whether the search facility is case-sensitive. The default is true (not case-sensitive). Changes to this switch take effect immediately. The full switch name is Session.Search_Ignore_Case.

Search_Preserve_Case

Sets a Boolean value that specifies whether the search and replace facilities maintain the case of the string being replaced. The default is false, which specifies that the search is not case-sensitive. Changes to this switch take effect immediately. The full switch name is Session.Search_Preserve_Case.

Search_Regular_Expr

Sets a Boolean value that specifies whether search strings are regular expressions. The default is false. Changes to this switch take effect immediately. The full switch name is Session.Search_Regular_Expr.

Send_Port_Enabled

Sets a Boolean value that helps diagnose failing FTP transfers by verifying that local and remote machines are using the same connection. The switch should be set to true for transfer of multiple files. The default is false. The full switch name is Session_Ftp.Send_Port_Enabled.

Text_Bottom_Stripe

Specifies a text string value that is placed below the header of an I/O window. The default is a line of dashes. Changes to this switch take effect when the I/O window is created. The full switch name is Session.Text_Bottom_Stripe.

Text_Convert_Tabs

Converts Control-I characters into the proper number of spaces for correct tabbing. The default is true. The full switch name is Session.Text_Convert_Tabs.

Text_Header

Sets a Boolean value that specifies whether the header for a job is displayed in the I/O window. The default is true. Changes to this switch take effect when the I/O window is created. The full switch name is Session.Text_Header.

Text_Print_Name

Sets a Boolean value that specifies whether to include the name of the job in the header. The default is true. Changes to this switch take effect when the I/O window is created. The full switch name is Session.Text_Print_Name.

Text_Print_Number

Sets a Boolean value that specifies whether to include the job number in the header. The default is true. Changes to this switch take effect when the I/O window is created. The full switch name is Session.Text_Print_Number.

Text_Print_Time

Sets a Boolean value that specifies whether to include in the header the time when the job started. The default is true. Changes to this switch take effect when the I/O window is created. The full switch name is Session.Text_Print_Time.

Text_Reuse_Window

Sets a Boolean value that specifies whether the I/O window should be reused every time an I/O window is needed or should be created for every need. The default is true. Changes to this switch take effect when the I/O window is created. The full switch name is Session.Text_Reuse_Window.

Text_Scroll_Output

Sets a Boolean value that controls the scrolling of I/O windows. When false, output windows do not scroll when output is displayed past the bottom of the window. When true, the window scrolls to display current output. The default is true. Changes to this switch take effect immediately. The full switch name is Session.Text_Scroll_Output.

Text_Top_Stripe

Specifies a text string value that is placed above the header of an I/O window. The default is a line of dashes. Changes to this switch take effect when the I/O window is created. The full switch name is Session.Text_Top_Stripe.

Transfer_Mode

Sets the mode of an FTP file transfer. Currently, only Stream type is supported. The default is stream. The full switch name is Session_Ftp.Transfer_Mode.

Transfer_Structure

Sets the structure of an FTP file transfer. Currently, only File type is supported. The default is file. The full switch name is Session_Ftp.Transfer_Structure.

Transfer_Type

Sets the type of an FTP file transfer. It may be set to allow text, image, and binary transfers. The default is ASCII. The full switch name is Session_Ftp.Transfer_Type.

Username

Sets the remote username for logging into a remote computer over an FTP connection. The default is the null string. The full switch name is Session_Ftp.Username.

Window_Command_Size

Specifies, as an integer value, the number of lines created in a Command window. The default is 2. Changes to this switch take effect when a Command window is created. The full switch name is Session.Window_Command_Size.

Window_Frames

Specifies, as an integer value, the initial maximum number of frames. The number of frames can be changed with the Editor.Window.Frames procedure. The default is 3. Changes to this switch take effect only at login. The full switch name is Session.Window_Frames.

Window_Frames_Startup

Specifies, as an integer value, the number of frames created at login. The default is 3. Changes to this switch take effect only at login. The full switch name is Session.Window_Frames_Startup.

Window_Have_Sides

Sets a Boolean value that specifies whether windows are created with left edges. The default is true. Changes to this switch take effect only at login. The full switch name is Session.Window_Have_Sides.

Window_Is_Staggered

Sets a Boolean value that specifies whether windows should be staggered at their left edges. The default is true. Changes to this switch take effect only at login. The full switch name is Session.Window_Is_Staggered.

Window_Message_Life

Specifies, as an integer value, the number of keystrokes required before the Message window is scrolled. Error messages that appear in the Message window have a life that is twice this value. The default is 2. Changes to this switch take effect immediately. The full switch name is Session.Window_Message_Life.

Window_Message_Size

Specifies, as an integer value, the number of lines in the Message window. The default is 2. Changes to this switch take effect only at login. The full switch name is Session.Window_Message_Size.

Window_Scroll_Overlap

Represents, as an integer value, the percentage of window overlap in scrolling up or down. When the image in the window is scrolled with the !Commands.Up or the Image.Down procedure,, the next full window overlaps the previous by this percentage. The value should be in the range of 0 through 100. The default is 20. Changes to this switch take effect when an image is scrolled. The full switch name is Session.Window_Scroll_Overlap.

Window_Shift_Overlap

Represents, as an integer value, the percentage of overlap when the image in the window is shifted left or right. When the image in the window is shifted with the !Commands.Editor.Image.Left or the Image.Right procedure, the next full window overlaps the previous by this percentage. The value should be in the range of 0 through 100. The default is 80. Changes to this switch take effect when an image is shifted. The full switch name is Session.Window_Shift_Overlap.

Word_Breaks

Specifies a set of characters that designate valid places where words can be broken. This set is defined as the word breaks at login. Changes to the set can be made with the !Commands.Editor.Word.Break procedure. The default word breaks (when none are explicitly set) are " `"#%&'()*+,-./:;<=>?[]_`{|}~`". Changes to this switch take effect only at login. The full switch name is Session.Word_Breaks.

Commands from Package Common

Many of the operations in package !Commands.Common apply to session switches. Session switches can be brought into a window with the !Commands.Switches.Edit_Session_Attributes procedure and the edited with common editing operations that apply to switches. The following procedures from package !Commands.Common apply to switches. Other operations from package !Commands.Common that do not apply to switches produce a message to that effect in the Message window when used on switches.

procedure Common.Abandon

Abandons the editing of the switches and removes the window from the screen. Any changes made to the switches since the last commit operation are lost.

procedure Common.Commit

Commits changes to the switches. Changes to the switches are made in a temporary area of the Environment. Changes must be committed before they are permanent and take effect.

procedure Common.Create_Command

Creates a Command window below the current window. A *use* clause in the Command window (`use Editor, Ada, Switches, Common;`) includes this package, so operations in this package are visible in the Command window without qualification.

procedure Common.Definition

Finds the definition of the selected switch value if that value is a library or library unit. The procedure produces an error for switches that are Booleans, integers, or nonswitch name strings. If the switch is a switch name, a window is brought up with the definition of that object in the window.

procedure Common.Edit

Creates a Command window and places in it the command `Change ("<current switch value>");`, where the parameter is the switch value of the switch on which the cursor is located, whether or not there is a selection. Providing a new switch value and promoting the command changes the value of the switch. If the current switch is of Boolean type, the command toggles the value of the switch without creating a Command window.

procedure Common.Elide

Reduces (elides) the number of switches displayed in the window. The window can display all switches in the system (the greatest number displayed) or the nondefault switches in the file (the least number displayed). This procedure reduces the number displayed to the next smaller set. Reducing the number below the nondefault switches has no effect.

procedure Common.Enclosing

Finds the directory or world that contains the switches that are in the current window. If the window contains session switches, the procedure finds the home world for that session. The In_Place parameter specifies whether the library window replaces the switch window.

procedure Common.Expand

Increases (expands) the number of switches displayed in the window. The window can display all switches in the system (the most number displayed) or the nondefault switches in the file (the least number displayed). This procedure increases the number displayed to the next larger set. Increasing the number above all switches in the system has no effect.

procedure Common.Explain

Inserts an explanation of the switch below the current switch. If there already is an explanation, this procedure removes it.

procedure Common.Promote

Commits changes to the switches. Changes to switches are made in a temporary area of the Environment. Changes must be committed before they are permanent and take effect.

procedure Common.Release

Commits changes and ends the editing of the switches. The window is removed from the screen after any changes to the switches are saved.

procedure Common.Revert

Redraws the switches in the current window. If the switches have been changed by another user or program, this procedure redraws the switches to ensure that the image is up to date.

procedure Common.Object.Child

Selects the switch on which the cursor is located. Specifically, if no switch is selected, the procedure selects the switch on which the cursor is located. If a single switch is already selected, the procedure has no effect. If all switches are already selected, the procedure selects the switch on which the cursor is located.

procedure Common.Object.Copy

Copies a highlighted switch from one set of switches to the set of switches in which the cursor is located. If the selected switch and the cursor are both in the same set of switches, the procedure has no effect.

procedure Common.Object.Delete

Deletes the selected switch or the switch on which the cursor is located. A deleted switch assumes a system-defined default value.

procedure Common.Object.First_Child

Selects the first switch of the set.

procedure Common.Object.Insert

Creates a Command window and places in it the command `Insert ("[Processor.] Switch := Value;");`, where the parameter must be specified to provide a switch and its value. Specifying a switch and a value for that switch and promoting the command inserts or changes a switch value. Multiple switches can be inserted with one command by separating them with semicolons. Parameters specified to this procedure use the same format as an Options parameter.

procedure Common.Object.Last_Child

Selects the last switch of the set.

procedure Common.Object.Move

Moves highlighted switches from one set of switches to the set of switches in which the cursor is located. If the selected switch and the cursor are both in the same set of switches, the procedure has no effect.

procedure Common.Object.Next

Selects the next switch. If no switch is selected, the procedure selects the switch on which the cursor is located. If all switches are selected, this procedure produces an error.

procedure Common.Object.Parent

Selects the switch on which the cursor is located. If no switch is selected, the procedure selects the switch on which the cursor is located. If a switch is selected, the procedure selects all switches in the set. Otherwise, the procedure has no effect.

procedure Common.Object.Previous

Selects the previous switch. If no switch is selected, the procedure selects the switch on which the cursor is located. If all switches are selected, this procedure produces an error.

RATIONAL

package What

The procedures in package What return information about the current session and the Environment.

The Does and Command procedures are help commands. The Help facility and how to use it are documented in EST, Help.

For information on what a particular key does or is called, see the !Commands.Editor.Key.Name procedure. See also the Keymap, in Volume 1 of the ***Rational Environment Reference Manual***, for information about keys.

The Does, Command, and Jobs procedures create a window that allows type-specific editing. In this window the user can enter commands from package !Commands-.Common to peruse the contents of the window.

To terminate the display, the user presses Window - D, Common.Abandon (Object - G), or Common.Release (Object - X).

Commands from Package Common

A few of the operations in package !Commands.Common apply to the windows created by some of the procedures in package What. The following procedures from package !Commands.Common apply to this window. Other operations from package !Commands.Common that do not apply to this window produce a message to that effect in the Message window when used on this window.

procedure Common.Abandon

Removes the window created by the Does, Command, and Jobs procedures from the screen. This command has the same effect as the Release command.

procedure Common.Create_Command

Creates a Command window below the Help window or set of jobs if one does not exist; otherwise, the procedure puts the cursor in the existing Command window below the Help window or What.Jobs display. This Command window initially has a use clause: `use Editor, Common;`. This use clause provides direct visibility to the declarations in packages Editor and Common for names resolved in the command.

procedure Common.Definition

Brings up on the screen, for help windows, an image of the Ada specification unit containing the designated declaration in a Help window declaration. This procedure has no effect on displays created by What.Jobs.

procedure Common.Elide

Selects which set of jobs is displayed in the window. The procedure steps the display from all jobs, to all running jobs, to the user's jobs, to the user's running jobs, to all commands, to all running commands, to the user's commands, to the user's running commands. The default is all running jobs. This procedure has no effect on Help windows.

procedure Common.Expand

Selects which set of jobs is displayed in the window. The procedure steps the display from the user's running commands, to the user's commands, to all running commands, to all commands, to the user's running jobs, to the user's jobs, to all running jobs, to all jobs. The default is all running jobs. This procedure has no effect on Help windows.

procedure Common.Explain

Adds an entry to the Help window for the designated item in a Help window menu. This procedure has no effect on displays created by What.Jobs.

procedure Common.Release

Removes the window from the screen for the What.Jobs display. For Help windows, this procedure removes the Help window from the screen and from the Window Directory. This procedure has the same effect as the Abandon procedure.

procedure Common.Revert

Redraws the set of jobs to reflect the current state for the What.Jobs display. This procedure has no effect on Help windows.

procedure Common.Object.Child

Selects the job on the line on which the cursor is located for the What.Jobs display, if one or more jobs are already selected when the procedure is entered. If there is a selection, the procedure leaves the current line selected. For Help windows, if the whole Help window is selected, the procedure selects the declaration on the line on which the cursor is located. If there is a selection, it leaves the current line selected.

procedure Common.Object.Delete

Kills the selected job or the job on which the cursor is located for the What.Jobs display. If a job is selected, that job is deleted (terminated). If no job is selected, the job on which the cursor is located is deleted. Note that if the job is not for the current session and user, the command will fail. In the What.Jobs display, Object.Delete is equivalent to Job.Kill. This procedure has no effect on Help windows.

procedure Common.Object.First_Child

Selects the first job of the set for the What.Jobs display. For Help windows, this procedure selects the first line of the Help window.

procedure Common.Object.Last_Child

Selects the last job of the set for the What.Jobs display. For Help windows, this procedure selects the last line of the Help window.

procedure Common.Object.Next

Selects the job that is listed after the currently selected job, provided that the cursor is on currently selected job, for the What.Jobs display. If the cursor is not on the currently selected job or if no job is already selected, the procedure selects the job on which the cursor is located. If all jobs are selected, this procedure produces an error message.

For Help windows, this procedure selects the next item past the current item declaration if the cursor is in the current selection; otherwise, it selects the item corresponding to the line on which the cursor is located. If nothing is currently selected, the procedure selects the item corresponding to the line on which the cursor is located. If the entire Help window is selected, this procedure produces an error message.

procedure Common.Object.Parent

Selects the job the cursor is on for the What.Jobs display. If a job is already selected and the cursor is on the currently selected job, the procedure selects all jobs in the set. If all jobs are already selected, the procedure has no effect.

For Help windows, this procedure selects the item corresponding to the line on which the cursor is located if there are no selections; otherwise, it selects the entire Help window. If the entire Help window is already selected, the procedure has no effect.

procedure Common.Object.Previous

Selects the job that is listed before the currently selected job, provided that the cursor is on the currently selected job, for the What.Jobs display. If the cursor is not on the currently selected job or if no job is already selected, the procedure selects the job on which the cursor is located. If all jobs are selected, this procedure produces an error.

For Help windows, the procedure selects the previous item before the currently selected item, if the cursor is in the current selection; otherwise, it selects the item corresponding to the line on which the cursor is located. If nothing is currently selected, the procedure selects the item corresponding to the line on which the cursor is located.

procedure Command

```
procedure Command (Clue : String := "<CURSOR>");
```

Description

Describes the named procedure, function, type, or other Environment element in the Help window.

If no Help window is currently displayed on the screen, this procedure displays the Help window.

The Environment inserts a description of Clue, along with all keys to which Clue is bound, at the end of the Help window.

The !Commands.Common commands available for Help windows are described in the introduction to this package.

Parameters

```
Clue :  String := "<CURSOR>";
```

Specifies the fully qualified name of the procedure, function, type, or other Environment element for which help is requested.

Example

The command:

```
    what.does ("!commands.editor.line.join");
```

displays a help message for the named procedure.

References

procedure Does

procedure Does

```
procedure Does (Name : String := "<CURSOR>");
```

Description

Accepts the Name parameter as a keyword to match with descriptions in the Help files and opens a Help window displaying all Environment commands containing Name.

The !Commands.Common commands available for Help windows are documented in the introduction to this package.

Parameters

```
Name : String := "<CURSOR>";
```

Specifies a keyword. This command searches through the directory system for relevant occurrences of the keyword. The default is the current cursor location. Name can be any portion of a full command name.

Example 1

The command:

```
    what.does ("write");
```

produces the following menu in the Help window:

```
    !Commands.Access_List.Write
    !Commands.Activity.Write
    !Commands.Common.Write_File
    !Commands.Switches.Write
    !Commands.Tape.Write
    !Commands.Tape.Write_Mt
    !Io.Direct_Io.Write
    !Io.Polymorphic_Sequential_Io.Operations.Write
    !Io.Sequential_Io.Write
    !Io.Window_Io.Overwrite
    !Tools.Access_List_Tools.Write
```

To see the entry for !Commands.Common.Write_File procedure, you designate the line and then use the !Commands.Common.Explain procedure.

Example 2

The command:

```
    what.does ("change_password");
```

displays the help message for the !Commands.Operator.Change_Password procedure.

procedure Home_Library

```
procedure Home_Library;
```

Description

Places the cursor in the user's home library.

Finds the window containing the user's home library and puts the cursor in that window. If such a window does not exist, the Environment creates it.

procedure Jobs

```
procedure Jobs (Interval          : Positive := 10;
                User_Jobs_Only    : Boolean  := False;
                My_Jobs_Only      : Boolean  := False;
                Running_Jobs_Only : Boolean  := True);
```

Description

Creates a display of jobs in the Environment and updates that display once every Interval number of seconds.

The Jobs display allows you to monitor system resource allocation. You can also use the Jobs display to view a list of your own jobs, from which you can kill selected jobs (see the !Commands.Common.Object.Delete procedure, in the introduction to this package).

The procedure creates a window that displays the specified set of jobs. The set of jobs is determined by the last three parameters as well as by commands that operate on the window. The window is updated at the specified interval. Press [Window] - [D], Common.Abandon ([Object] - [G]), or Common.Release ([Object] - [X]) to terminate the display.

A few of the operations in package !Commands.Common apply to the window created by this procedure. The procedures from package !Commands.Common that apply to this window are documented in the introduction to this section. Other operations from package !Commands.Common that do not apply to this window produce a message to that effect in the Message window when used on this window.

To temporarily stop the updating of the Jobs display, select one of the lines in the display. You might want to do this to simply look at the display or to perform a Common operation without the display changing.

The interpretation of the display is described below in an example.

Parameters

`Interval : Positive := 10;`

Specifies the minimum interval, in seconds, between updates of the screen. The default is 10 seconds.

`User_Jobs_Only : Boolean := False;`

Specifies whether the display should include all user jobs. The default is not to display them.

```
My_Jobs_Only : Boolean := False;
```

Specifies whether the display should include only your current jobs. The default is not to display them.

```
Running_Jobs_Only : Boolean := True;
```

Specifies whether the display should include only the running jobs. The default is to display them.

Example

The command:

```
    what.jobs;
```

produces a display like the following and continues to update the display until you terminate it using [Window] - [D], [Object] - [G], or [Object] - [X]:

```
  User   Session Job  S Elapsed      CPU      % CPU  Cache Disk  Job Name
======= ======= ===  = ========= ========= =====  ===== ====  ========
        SYSTEM    4  R Ø5:1Ø:15  34:52.294  5.77    11K  87k
        DAEMON    5  R Ø5:13:35  ØØ:17.238  Ø.Ø2    131  468
TDG     S_1     232  I Ø3:51.184 ØØ:14.974  2.45    2Ø5  5Ø3  [EDITOR]
SJL     S_1     236  I Ø3:Ø1:Ø8  Ø1:Ø1.499  1.Ø4    197  513  [EDITOR]
TDG     S_1     244  I 19:25.119 ØØ:46.945  8.17    212  349  [COMMAND]
        SYSTEM  253  I Ø9:Ø8:47  ØØ:33.975  3.26     22  12Ø7 "!Tools.Ftp.

= What.Jobs (jobs) Ø5/Ø7/87 Ø2:41:56 PM (Load: Ø.27)      All Running Jobs
```

The headings are described below:

`User`	Indicates the username of the user who started the job. Entries for jobs started by the system appear with blanks in the `User` column.
`Session`	Indicates the name of the session in which the job was started.
`Job`	Indicates the number of the job.
`S`	Indicates the status of the job. Values for the job status are defined by the !Commands.Scheduler.Job_State type. The status values are:

`R` (Run)	Indicates that a job is executing.
`I` (Idle)	Indicates that a job is waiting for input or requests for service.
`W` (Wait)	Indicates that a job is able to run but the system is withholding it because too many jobs are in the run queue.

	D (Disabled)	Indicates that a job has been explicitly disabled with the !Commands.Job.Disable procedure. It will remain unable to run until explicitly enabled with the Job.Enable procedure.
	Q (Queued)	Indicates that a job is waiting to run in one of the background queues. See SMU, package Scheduler, for more information about background queues.

Elapsed	Indicates the elapsed time since the job began. • mm:ss.fff indicates the number of minutes, seconds, and decimal fractions of seconds of elapsed time. • hh:mm:ss indicates the number of hours, minutes, and seconds of elapsed time. • dd/hh:mm indicates the number of days, hours, and minutes of elapsed time.
CPU	Indicates the total amount of CPU time the job has used since the job began executing.
% CPU	Indicates the percent of the CPU time over the last Interval that was consumed by the job.
Cache	Indicates the number of pages of memory that the job is currently using (that is, the working set size).
Disk	Indicates the number of times the job had to wait for the disk since it began.
Job Name	Indicates the name of the job, either a command name or the name of an editor job started automatically by a user's login (enclosed in brackets). The job names in brackets cannot be killed by the !Commands.Common.Object.Delete or !Commands.Job.Kill procedures.

References

EST, package Common

SMU, package Scheduler

procedure Line

```
procedure Line;
```

Description

Displays the line number and column position for the cursor location in the current image.

Example

The command:

```
    what.line;
```

produces a display such as the following in the Message window:

```
    Line Number 22, Column Number  12
```

procedure Load

```
procedure Load (Verbose : Boolean := True);
```

Description

Displays the current system load in the Message window.

When the Verbose parameter is false, this procedure displays four numbers in the Message window. Each of these numbers expresses how many tasks are eligible for CPU time, averaged over one of four sampling intervals: the last 100 milliseconds, the last minute, the last 5 minutes, and the last 15 minutes.

When the Verbose parameter has the value true (the default), the display shows 12 numbers expressing:

- The average number of runnable tasks, calculated over the four sampling intervals listed above.
- The average number of tasks waiting on disk operations, calculated over the four sampling intervals listed above.
- The average number of tasks that are withheld from running, calculated over the four sampling intervals listed above. A task is withheld from running if it is consuming more than its share of resources or if it has been queued or disabled.

Parameters

```
Verbose : Boolean := True;
```

Specifies whether to display a verbose set of load numbers or the summary. The default is to display the verbose set.

Example 1

The command:

```
    what.load (false);
```

prints a message such as the following in the Message window:

```
    Load: 1.95, 0.25, 0.48, 0.41
```

1.95	The average number of tasks in the run queue during the last 100 milliseconds.
0.25	The average number of tasks in the run queue in the last minute.
0.48	The average number of tasks in the run queue in the last 5 minutes.
0.41	The average number of tasks in the run queue in the last 15 minutes.

Example 2

The command:

```
    what.load;
```

prints a message such as the following in the Message window:

```
L: 1.98, 0.35, 0.62, 0.44 D: 0.00, 0.04, 0.06, 0.04 H: 0.00, 0.00, 0.00, 0.00
```

where:

L (Load)	Lists the number of runnable tasks, averaged over the last 100 milliseconds, 1 minute, 5 minutes, and 15 minutes, respectively.
D (Disk)	Lists the number of tasks waiting for pages from disk, averaged over the last 100 milliseconds, 1 minute, 5 minutes, and 15 minutes, respectively.
H (Held)	Lists the number of withheld jobs, averaged over the last 100 milliseconds, 1 minute, 5 minutes, and 15 minutes, respectively.

References

SMU, procedure Scheduler.Get_Disk_Wait_Load

SMU, procedure Scheduler.Get_Run_Queue_Load

SMU, procedure Scheduler.Get_Withheld_Task_Load

procedure Locks

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

Description

Displays the locks that exist on the current or named object.

The locks are displayed in the current output window.

The Environment places a lock on an object when a job is run against the object or when someone is editing the object. Therefore, if you are unable to gain access to an object, you can inspect the locks display to find out who is editing it or what job is accessing it.

A write lock is placed on an object when someone edits it, so that two users cannot be simultaneously editing the same object.

A read lock is placed on an object when you display it.

The editor can maintain a lock on just the image of an Ada unit. If this is the case, the unit's name with the suffix "'Image" appears in the What.Locks display.

Parameters

```
Name : String := "<IMAGE>";
```

Specifies the object whose locks should be displayed. The default is the current image.

procedure Message

```
procedure Message (File : String := "Daily_Message");
```

Description

Displays the daily message in Standard_Output.

This procedure is typically included in login procedures.

Parameters

```
File : String := "Daily_Message";
```

Specifies the name of the message file to be displayed. The system first attempts to find the file in the current context if File is not fully qualified. If it does not find the file in the current context, it attempts to open a file called !Machine.Editor_Data-.Daily_Message, if such a file exists.

To name another file, you are advised to specify an absolute name (starting with "!"), because a relative name will be resolved relative to the context from which the command is executed.

Example

The command:

```
    what.message;
```

puts a message called "Message of the Day" in the Message window and prints the contents of the daily message file !Machine.Editor_Data.Daily_Message.

See the chapter "Sending Messages" in the *Rational R1000 Development System: System Manager's Guide* for examples of using nondefault message files.

procedure Object

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

Description

Displays in the Message window the fully qualified name of the current or named object.

Parameters

```
Name :  String := "<IMAGE>";
```

Specifies the name whose fully qualified name is to be displayed. The default is the current image.

procedure Tabs

```
procedure Tabs;
```

Description

Displays in the Message window all of the current tab stops.

procedure Time

```
procedure Time;
```

Description

Returns the current date and time of day in the Message window.

Example

The command:

```
    what.time;
```

produces a message such as the following in the Message window:

```
    June 2, 1987 at 10:24:23 AM
```

procedure Users

```
procedure Users (All_Users : Boolean := True);
```

Description

Displays a list of users who are currently logged in and the active jobs for each user in Standard_Output.

The list, which includes how long each user has been logged in, the port number on which each user is logged in, and a static list of sessions and jobs started by each user, is displayed in the current output window. The list is static in that it is not updated automatically when users log in and out, when jobs terminate, and the like. The user must enter the command again to display a fresh list of current information.

The Users display is useful to get a user's terminal number before using !Commands.Operator.Force_Logoff or to get a job number before using !Commands.Job-.Kill.

Parameters

```
All_Users : Boolean := True;
```

Includes, when true (the default), information in the list about all users currently logged in. When false, the procedure includes information only about the user who entered the Users command.

Example 1

The command:

```
    what.users;
```

produces a display such as the following:

```
User Status on June 17, 1987 at 04:22:52 AM

  USER     LINE  JOB    S       TIME                     JOB NAME
========   ====  ====  ====  =========  ============================
*SYSTEM     -       4  RUN     1:40:06  System
                    5  RUN     1:40:06  Daemons

DHE        247    185  IDLE    1:02:48  [DHE.S_1 Editor]
                  247  IDLE    1:02:43  [DHE.S_1 Command]

JLS        244    234  IDLE  40:05.774  [JLS.S_1 Editor]
                  254  IDLE  39:38.366  [JLS.S_1 Command]

GZC        241    188  IDLE    1:29:58  [GZC.S_1 Command]
                  228  RUN       2.187  GZC_WORK.REV1_WOR
                  250  RUN     1:30:14  [GZC.S_1 Editor]
```

The headings are described below:

User	Indicates the username of the account. The username is shown only for the user's session. It is not repeated for each job running under that session.
Line	Indicates the port number on which the user is logged in. A dash (-) in the LINE column means that the user is not logged into the Environment but has one or more jobs running.
Job	Indicates the job number of a job the user has started.
S	Indicates the status of a user job or session. Values for job status are defined by the !Commands.Scheduler.Job_State type. The status values are:

RUN	Indicates that a job is executing.
IDLE	Indicates that a job is waiting for input or requests for service.
WAIT	Indicates that a job is able to run but the system is withholding it because too many jobs are in the run queue.
DISABLED	Indicates that a job has been explicitly disabled with the !Commands.Job.Disable procedure. It will remain unable to run until explicitly enabled with the Job.Enable procedure.
QUEUED	Indicates that a job is waiting to run in one of the background queues. See SMU, package Scheduler, for more information about background queues.

Time — Indicates the elapsed time since the job or session began. For each logged-in user, the time on the session line indicates how long that user has been logged in.

- mm:ss.fff indicates the number of minutes, seconds, and decimal fractions of seconds of elapsed time. In the above example, the job What.Users has run for a little over 3 seconds, or 00:03.395.

- hh:mm:ss indicates the number of hours, minutes, and seconds of elapsed time. In the above example, the Operator has been logged in for 8 hours, 17 minutes, and 6 seconds, or 08:17:06.
- dd/hh:mm indicates the number of days, hours, and minutes of elapsed time. In the above example, user DHE has been logged in for over 2-1/2 days, or 2/15:21.

Job Name — Indicates the name of the user's session or the name of a job the user has started. Note that one username (in the above example, Anderson) can have more than one session (session'(S_1) and session'(DOCS)). Although jobs are grouped together by session name, session names are not necessarily grouped by usernames.

Note that the entry for *SYSTEM indicates how long the system has been up. Jobs listed under *SYSTEM are jobs that were started by !Machine.Initialize.

Example 2

The command:

```
what.users(false);
```

executed by the user GZC, produces a display such as the following:

```
User Status on June 17, 1987 at 04:22:52 AM

  User     Line  Job   S        Time              Job Name
========  ====  ====  ====  =========  ========================
GZC        241   188  IDLE    1:29:58  [GZC.S_1 Command]
                 228  RUN       2.187  GZC_WORK.REV1_WOR
                 250  RUN     1:30:14  [GZC.S_1 Editor]
```

References

procedure Jobs

procedure Version

```
procedure Version;
```

Description

Displays the current Environment version number in the Message window.

end What;

Index

This index contains entries for each unit and its declarations as well as definitions, topical cross-references, exceptions raised, errors, enumerations, pragmas, switches, and the like. The entries for each unit are arranged alphabetically by simple name. An italic page number indicates the primary reference for an entry.

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