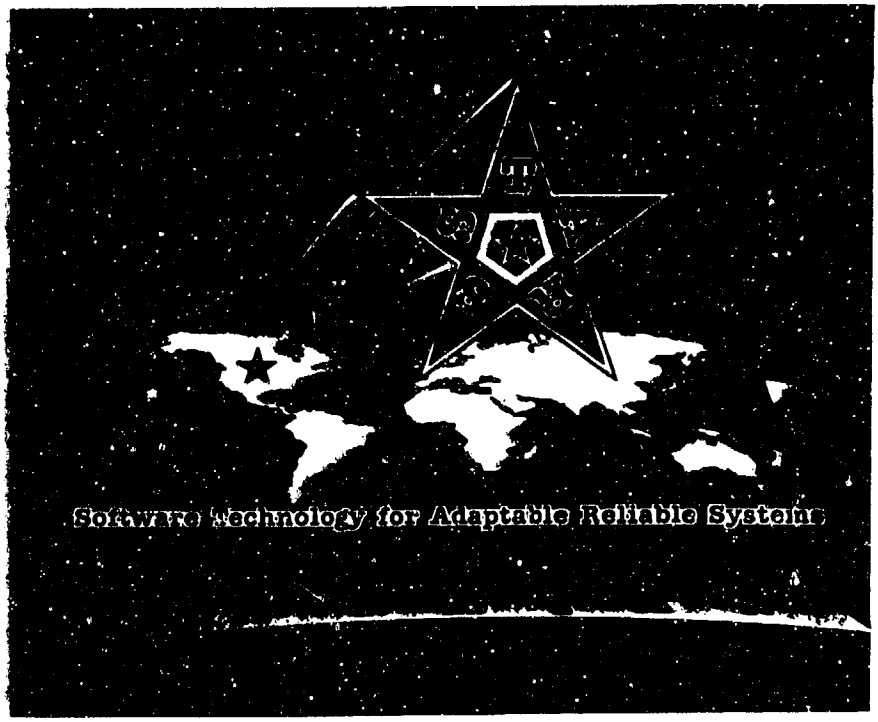
TASK: UR40
CDRL: 01210

UR40 — Repository Integration Librarian Software User's Manual

Informal Technical Data

UNISYS



STARS-RC-01210/001/00

26 October 1990

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	17 August 1990	Final

4. TITLE AND SUBTITLE
Reusability Library Framework (RLF) AdaTAU User's Manual

5. FUNDING NUMBERS
STARS Contract
F19628-88-D-0031

6. AUTHOR(S)
James J. Solderitsch

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

8. PERFORMING ORGANIZATION REPORT NUMBER
GR-7670-1171 (NP)

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Air Force
Headquarters, Electronic Systems Division (AFSC)
Hanscom AFB, MA 01731-5000

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
01210 Volume I

11. SUPPLEMENTARY NOTES
There are two other related Reusability Library Framework (RLF) reports:
(RLF) AdaKNET User's Manual and (RLF) Librarian Software User's Manual

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release;
distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

AdaTAU is the inferencing subsystem included in the RLF. TAU stands for "Think, Ask, Update" which is a summary of the inferencing approach taken by AdaTAU. The manual describes how to install AdaTAU, provides a brief background for the current structure of the AdaTAU system and provides a list of the programmatic interfaces available within AdaTAU. Advice is given to users for configuring applications to make use of AdaTAU. A sample session showing the use of the included AdaTAU test harness is also included. The manual closes with a description of the specification language used to define inference bases.

See p 1

14. SUBJECT TERMS	15. NUMBER OF PAGES
AdaTAU System Components Using AdaTAU	108
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

TASK: UR40
CDRL: 01210
26 October 1990

INFORMAL TECHNICAL REPORT

For The

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

Reusability Library Framework (RLF)
Librarian Software
User's Manual

STARS-RC-01210/001/00
Publication No. GR-7670-1169(NP)
26 October 1990

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0002

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

Unisys Defense Systems
Tactical Systems Division
12010 Sunrise Valley Drive
Reston, VA 22091

PREFACE

This document was prepared by Unisys Defense Systems, Valley Forge Operations, in support of the Unisys STARS Prime contract under the Repository Integration task (UR40). This CDRL, 01210, Volume I, is type A005 (Informal Technical Data) and is entitled "Librarian Software User's Manual".

This document has been reviewed and approved by the following Unisys personnel:

UR40 Task Manager: Richard E. Creps

Reviewed by: Richard E. Creps FOR
Teri F. Payton, System Architect

Approved by: Hans W. Polzer
Hans W. Polzer, Program Manager

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By per letter	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Table of Contents

1. Scope 1
1.1. Identification 1
1.2. Purpose 1
1.3. Organization 1
2. Referenced Documents 3
3. Librarian System Overview 4
3.1. The Library 4
3.2. The Librarian 5
3.2.1. Accessing a Library with the Librarian 5
3.2.2. Browser Mode 5
3.2.3. Query Mode 7
3.2.4. Editing Mode 7
3.2.5. Classifier Mode 7
3.2.6. Advisor Mode 7
4. Librarian System Components 8
4.1. Top-level Drivers 8
4.2. Software Unit Adder 8
4.3. Classifier 9
4.4. Differences 10
4.5. Display Differences 12
5. Using The Librarian 13
5.1. Installation 13
5.2. Commands - Excerpts from a Sample Session 13
5.2.1. Beginning a Session 13
5.2.2. Top Level Commands 14
5.2.2.1. Exiting the Library 14
5.2.2.2. Query Mode 15
5.2.2.3. Category and Component Context Display Commands 15
5.2.2.4. Display Items Most Recently Visited 19
5.2.2.5. Go To Category or Unit 20
5.2.2.6. Category and Unit Hierarchy Display Commands 25
5.2.2.7. Attribute Display Commands 31
5.2.2.8. Editing Commands 36
5.2.2.9. Display Category Differences 55
5.2.2.10. Display Component Differences 55
5.2.2.11. Invoke Classifier 56
5.2.2.12. Librarian Guidance 58

Appendices

A. Glossary of AdaKNET Terminology A-1
B. Library Hierarchy - Benchmarks B-1
C. AdaKNET Query Language Syntax and Summary C-1
C.1. Extended BNF (EBNF) Meta-Symbols 1
C.2. Query Language EBNF and Semantics 1
C.3. Query Language EBNF Syntax Summary 3
D. HKBDL Syntax and Summary D-1
D.1. Extended BNF (EBNF) Meta-Symbols 1
D.2. HKBDL EBNF and Semantics 1
D.3. HKBDL EBNF Syntax Summary 2

References

1. Scope

The Librarian is a knowledge-based tool developed as part of the Reusability Library Framework project. This document provides the information needed to use the Librarian to explore the contents of a software component library as well as the information needed to modify such a library.

The Librarian utilizes two underlying knowledge-based systems, AdaKNET and AdaTAU, which provide semantic network and rule-based inference capabilities. This User's Manual assumes some familiarity with AdaKNET and AdaTAU. The user wishing to use the Librarian for retrieving components does not need a detailed understanding of AdaKNET or AdaTAU; those wishing to extend or modify the Librarian are directed to the the AdaKNET and AdaTAU Software User's Manuals also delivered under this contract.

1.1. Identification

This User's Manual contains an overview of the Librarian, provides instructions sufficient to initiate and execute the Librarian, describes user inputs, and demonstrates a sample interactive session.

1.2. Purpose

The Librarian provides a means of effectively using the increasing amount of potentially reusable software. Through the combined use of its two underlying knowledge representation tools, AdaKNET and AdaTAU, the Librarian enables one to build a software component library that will evolve as its domain does.

Software component libraries created with this tool may contain both static and generated components, composite and stand-alone software, as well as partially specified software. The library may evolve independently of the Librarian application; i.e., there is separation of procedural (the Librarian) and declarative (the library) knowledge.

The Librarian is designed to allow both novice and experienced users effective access to a repository of software. It allows experienced users to rapidly focus on the software in which they are interested, while providing novice users sufficient support to get acquainted with the library.

1.3. Organization

The remainder of this document is organized as follows. Section 2 lists reference works that have relevance to this document. Section 3 contains an overview of the Librarian application, including information about how the library is represented. Section 4 reviews the package level components for the procedural portion of the Librarian application. A hands-on view of the Librarian is presented in section 5, including representative portions of transcripts from interactive sessions with the Librarian. Appendix A defines some of the AdaKNET terms used in this document. Appendix B contains the category taxonomy of the supplied benchmarks library. Appendix C gives a description of the syntax and semantics of the AdaKNET query language. Appendix D provides a detailed definition of the Hybrid Knowledge Base Description Language (HKBDL). A more complete glossary and description of AdaKNET is contained in the AdaKNET User Manual, whereas the SNDL and RBDL specifications for the

benchmarks library may be found in the "Sndl_specs" and "Rbdl_specs" directories of the final release respectively.

2. Referenced Documents

The following documents are useful as references in conjunction with this document. Documents marked with an asterisk (*) were delivered to the Naval Research Laboratory as part of the original STARS Foundation contract (number N00014-88-C-2052) that supported the initial development of the RLF.

AdaKNET Software User's Manual.

AdaTAU Software User's Manual.

The RLF Librarian: A Reusability Librarian Based on Cooperating Knowledge-Based Systems [McDowell89].

The Growing of an Organon: A Hybrid Knowledge-Based Technology and Methodology for Software Reuse [Simos88].

Construction of Knowledge-Based Components and Applications in Ada [Wallnau88].

Constructing Domain-Specific Ada Reuse Libraries [Solderitsch89].

(*) Reusability Library Framework AdaKNET/AdaTAU Design Report.

(*) Reusability Library Framework Intelligent Librarian Design Report.

(*) Gadfly User's Manual.

3. Librarian System Overview

In this section, a brief overview of the Librarian application is provided. The Librarian is a knowledge-based program used to create and manipulate software libraries. For most efficient use of the Librarian, it is necessary to understand the underlying structure of the library itself as well as the operations provided. Figure 1 presents a pictorial representation of the Librarian with an underlying library. The library is the part of the diagram within the circle with the Librarian sitting on top.

3.1. The Library

The software libraries that the Librarian manipulates are modelled using the structured inheritance network, AdaKNET. This has the following implications for a user of the library:

- The library is organized in a hierarchical manner. Typically, a user starts at a very general category and proceeds to more specific ones.
- A category may have more than one immediate subcategory and more than one immediate supercategory, i.e., the hierarchy may be represented as a graph. In the diagram, categories are the plain circles.

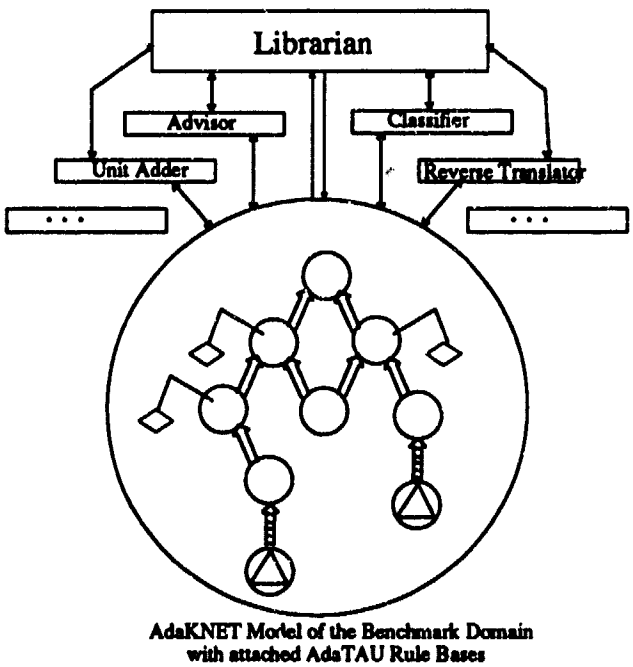


Figure 1. Overview of the Librarian

- A software unit may belong to more than one immediate supercategory. In the diagram, software units are the circles with embedded triangles.
- Categories and units may have attributes associated with them which serve to define them. These are not visible in this diagram.
- All subcategories of a category and units within that category must fall within the specification of that category as given by its attributes, i.e., AdaKNET "subsumption" is followed.
- Categories may have advice associated with them. This advice is represented in the diagram as diamonds tied to the circles.

3.2. The Librarian

The Librarian is primarily a menu-driven application designed to be useful to inexperienced users as well as expert users. Its fundamental mode of operation relies on a browsing paradigm; that is, a library user moves through the library under his own control, deciding which categories or software units to investigate next. In Figure 1, the browsing mode is represented by the largest, topmost rectangle. While browsing, the user is presented with a current context, that is, the name of the current category or unit, and a menu of legal commands to execute at this point.

In addition to the primary browsing mode, the Librarian provides the user with a variety of subordinate modes which are reachable from the main menu and are represented in Figure 1 by smaller rectangles beneath the browser rectangle. After operation of one of the subordinate commands is complete, control returns to the browser.

Commands are presented in detail in Section 5 - "Using the Librarian". The available commands may vary depending on whether one is examining a category or a software unit. In addition, two versions of the Librarian are available, the user version and the administrator version. Both versions of the Librarian allow access to the library; the administrator version also allows the library to be changed.

3.2.1. Accessing a Library with the Librarian

Immediately upon beginning a session with the Librarian, the user is asked for the library he wishes to access. The user version presents a menu of existing libraries from which the user may choose. The administrator version also allows the repository administrator to specify the name of a new library he wishes to create. This new library will contain the single category, "Thing". The knowledge base specifications for a library called "benchmark_library" is included with the RLF delivery.

3.2.2. Browser Mode

Exiting the Library

This allows one to exit the Librarian. The administrator is given the option of saving changes, destroying the library, or not altering the library at all. The user automatically exits the Librarian with no changes made to the library.

Displaying the Category and Component Context

Using these commands enables one to see the immediate context of the current category or software component in the library. Examples of these commands include "Immediate Super Categories", "Immediate Sub Categories", "Attributes of Category or Unit", "Categories Having Attributes that Reference this Category", "Components Having Attributes that Reference this Category", and "Units with this Unit as Attribute".

Displaying Items Most Recently Visited

This command provides a list of the categories and components that the user has most recently visited. It provides the user with a history mechanism to remind him of where he has been and the path he has taken through the library.

Going to a Category or Component

These commands allow the user to navigate through the library. The user may select from menus of categories and components that are related to the current one, or may directly specify the name of a category or component he wishes to visit. The options given to the user for places to move are "A user named category/component", "A previously visited category/component", "A generalization of the current category/component", "A specialization of the current category", "A component of the current category", "A value for an attribute of this category/component", and "A type for an attribute of this category/component".

Displaying the Category and Unit Hierarchy

Using these commands enables one to see the organizational framework of the library. They serve to display the categories and components available in the library as well as those categories and components that have some special relationship to the one currently being examined. Examples of these commands include "All Components Satisfying Current Category", "Sub Categories", "Common Super Categories - any two categories", and "SNDL dump entire library hierarchy to file (depth-first)".

Displaying Attributes

These commands serve to display the attributes in the library, particularly as they relate to the category or component currently being examined; i.e., they show the salient characteristics of library items. "Print aggregation structure alphabetically" is an example of such a command.

Displaying Differences

This command highlights the differences between two items in a library, ignoring their similarities. Thus, it is particularly useful for distinguishing between two similar items. There are two versions of this command. One displays the difference between two software categories based on the differences between their attributes. The other displays the difference between two software components.

3.2.3. Query Mode

This mode provides a query interface to the library. The user gives a component description in terms of categories and attributes and is shown the list of components that fit the description. Query mode is most useful for users familiar with the organizational framework of the library.

3.2.4. Editing Mode

These commands serve to allow one to alter the organization of the library and are only included in the administrator version of the Librarian. Any changes to the library component hierarchy will not become permanent until the library is explicitly saved. However, changes made to the integer and text values of attribute values become effective immediately. Examples of editing commands include "add subcategory", "remove unit", and "remove attribute".

3.2.5. Classifier Mode

This mode provides a quicker descent through the hierarchy of categories by repeatedly presenting the user with the next level of subcategories until the user requests to exit or until there are no more subcategories. It is most useful for library users who are already somewhat familiar with the domain of the library.

3.2.6. Advisor Mode

Advisor mode, a.k.a. "Librarian Guidance", gives the user advice about which category or component to investigate next based on a brief question and answer session. It is especially useful for users who are unfamiliar with the library domain.

4. Librarian System Components

Many of the subprograms called from the Librarian are actually AdaKNET manipulation commands that are available in the packages implementing the AdaKNET knowledge representation system. This section only presents an overview for those packages which are specific to the Librarian, although these components may make extensive use of packages from the AdaKNET and AdaTAU systems.

4.1. Top-level Drivers

Subprogram Name

`Admin_Browser`

Description

This subprogram runs the overall Librarian application. Admin_browser is an interactive browser and editor of a hybrid AdaKNET/AdaTAU structure. It provides a way of proceeding through a library while examining it or changing it. It is an interactive procedure for the use of the library administrator. Admin_browser loads a library specified by the user and proceeds from there based on user input.

Subprogram Name

`User_Browser`

Description

This subprogram is similar to Admin_Browser but does not allow the user to make modifications to the library.

4.2. Software Unit Adder

Subprogram Name

`Adder`

Description

The adder provides a guided insertion of a software unit into a library taxonomy. It interacts with the user to determine any modifications that should be made to inherited attributes, and to specify fillers for those attributes.

4.3. Classifier

Package Name

`Specialization_Selection`

Description

Specialization_Selection collects a number of different operations that walk the specialization links of an AdaKNET network, converging on one final desired concept as the result.

Objects

`Generic_Concept_Lists`
`Individual_Concept_Lists`

Operations

`Select_Direct_Specialization`

Version 1: Gets the user's choice as to which child (specialization) of a concept is desired, and passes this child back.
Version 2: Gets the user's choice as to which children (specializations) of a concept are desired, and passes these children back.

`Optionally_Select_Direct_Specialization`

Description: Version 1: Gets the user's choice as to which child (specialization) of Concept is desired, and passes this child back as Selection.
Versions 2, 3: Gets the user's choice as to which children (specializations) of Concept are desired, and passes these children back as Selection.

`Select_Leaf_Specialization`

Version 1: Walks the user down the specialization sub-network to select the most specific concept applicable. At each concept, the set of specializations is obtained and displayed to the user. One child is selected, and the process is repeated for this child. If a concept has no further specializations, it is assumed to be the user's choice.
Versions 2, 3: Walks the user down the specialization sub-network to select the most specific concepts applicable. At each concept, the set of specializations is obtained and displayed to the user. One or more children are selected, and the process is repeated for these children. If a concept has no further specializations, it is assumed to be the user's choice and added to the selection set.

`Select_Specialization`

Version 1: Walks the user down the specialization sub-network to select the most specific concept applicable. At each concept, the set of specializations is obtained and displayed to the user. One child may be selected, and the process is repeated for this child. If a concept has no further specializations or if the user does not select any of the specializations, that concept is assumed to be the user's choice.

Versions 2, 3: Walks the user down the specialization sub-network to select the most specific concepts applicable. At each concept, the set of specializations is obtained and displayed to the user. One or more children may be selected, and the process is repeated for these children. If a concept has no further specializations or if the user does not select any of the specializations, that concept is assumed to be the user's choice and is added to the selection set.

`Classify_To_leaf`

Version 1: Walks the user down the specialization sub-network to select the applicable "leaf" concept (concept without specializations). An individual is then created at that concept and passed back.

Versions 2, 3: Walks the user down the specialization sub-network to select the applicable "leaf" concept (concept without specializations). An individual is then created at each concept selected and the set of new individuals is passed back.

`Classify`

Version 1: Walks the user down the specialization sub-network to select the most specific concept applicable. An individual is then created at that concept and passed back.

Versions 2, 3: Walks the user down the specialization sub-network to select the most specific concepts applicable. An individual is then created at each concept selected and the set of new individuals is passed back.

4.4. Differences

Package Name

`Differences`

Description

Differences provides operations that compare two concepts in a network, reporting the differences between their rolesets.

Objects

`Roleset_Kind`
`Roleset_Difference`

Operations

`Difference`

Returns true if two difference descriptions describe the same roleset difference.

`Unique_Role`

Returns true if a difference description describes a roleset with no corresponding roleset at the other concept.

`Different_Ranges`
Returns true if a difference description describes a difference of ranges at the corresponding rolesets of concepts.

`Different_Types`
Returns true if a difference description describes a difference of filler types at the corresponding rolesets of concepts.

`Different_Fillers`
Returns true if a difference description describes a difference of fillers at the corresponding rolesets of concepts.

`Associated_Role`
Returns the role of the differing rolesets described by a difference description.

`Roleset_Owner`
If a difference description describes a unique role, returns the concept at which the role exists.

`First_Roleset_Owner`
If a difference description describes the difference between corresponding rolesets at two concepts, returns the owner of the first roleset.

`Second_Roleset_Owner`
If a difference description describes the difference between corresponding rolesets at two concepts, returns the owner of the second roleset.

`First_Roleset_Range`
If a difference description describes the difference between corresponding rolesets at two concepts, returns the range of the first roleset.

`Second_Roleset_Range`
If a difference description describes the difference between corresponding rolesets at two concepts, returns the range of the second roleset.

`First_Roleset_Type`
If a difference description describes the difference between corresponding rolesets at two concepts, returns the filler type of the first roleset.

`Second_Roleset_Type`
If a difference description describes the difference between corresponding rolesets at two concepts, returns the filler type of the second roleset.

`First_Roleset_Fillers`
If a difference description describes the difference between corresponding rolesets at two concepts, returns the fillers of the first roleset which are not also fillers of the second.

`Second_Roleset_Fillers`
If a difference description describes the difference between corresponding rolesets at two concepts, returns the fillers of the second roleset which are not also fillers of the first.

4.5. Display Differences

Package Name

`Difference_Display`

Description

This package contains routines that will print the differences between two library objects based on their network structure.

Objects

<none>

Operations

`Display_Differences`

Two versions. One version prints the differences between two categories; the other prints the differences between two units.

5. Using The Librarian

This section describes how to install the Librarian on a new host, how to create AdaKNET and AdaTAU knowledge bases, and how to use the Librarian to examine and manipulate the knowledge bases. The librarian shares many characteristics with the browser-editor application from which it evolved, so a look at the AdaKNET user's manual provides useful background information. Readers interested in additional discussion of knowledge base inferencing should consult the AdaKNET/AdaTAU user manuals as well as [McDowell89] and [Wallnau88].

5.1. Installation

Installing the Librarian requires the installation of the AdaKNET, AdaTAU, and hybrid components, the RBDL and SNDL knowledge bases, and the Librarian application itself. Each of these components was implemented using the Verdix Ada Compilation System (versions 5.5 and 6.0) and the Telesoft compiler on a Sun network running Sun OS 3.5, 4.0.3 and 4.1. Detailed instructions for installation are provided in the Version Description Document (VDD) that accompanies the full RLF release.

5.2. Commands - Excerpts from a Sample Session

In the current version of the RLF, UNIX environment variables enable the user to specify the file system locations of RLF knowledge bases. The VDD discusses how to make use of this RLF feature. Before actually executing the librarian for a particular library model, the RLF_LIBRARIES environment variable must be set to reference the UNIX file system location of the model.

The following sections will provide a guided tour to using the Librarian. Many of the commands will be illustrated with portions of an actual transcript. Where used, the transcript will be indented slightly and will be in a slightly smaller font. Uninteresting or redundant portions may be omitted and will be indicated with ellipses (...).

Commands are available via menus. Most menus have an entry that allows one to exit the menu without actually executing any commands.

Some commands require textual input from the user. The input is not parsed or regularized, so correct case and spacing is important. Usually, a command that requires textual input can be aborted by typing "*abort*".

Typing Control-C at any prompt will also abort the current command or menu.

5.2.1. Beginning a Session

To begin a session, type the executable name at the operating system prompt. On our system, this was "ADMIN_BROWSER" or "USER_BROWSER". For the administrator browser, the user will be asked if he wishes to access an existing library or create a new one. If he chooses to create a new one, the Librarian will prompt him for the name of the new library and create a library by that name containing the single category, "Thing". Otherwise he is given a menu of existing libraries from which he may choose the library he wishes to use. The user browser automatically displays the menu of existing libraries upon invocation.

```
% ADMIN_BROWSER

      1. Create a new library
      2. Choose from existing libraries

Enter number of desired command <CR>: 2

What Library do you want to browse?

      1. benchmark_library
      2. unisys_ada_repository

Enter number of desired command <CR>: 1
```

5.2.2. Top Level Commands

The following is the top level menu presented after the chosen library is loaded. The current category or component is listed to show the user's current position in the library. Each menu selection represents either a command or a submenu. These will be discussed in subsequent sections. The "Editing Commands" option is not given in the user version of the Librarian. The "Invoke Classifier" and "Provide Librarian Guidance" options are only available for categories and not for specific components.

```
CURRENT CATEGORY --> thing

      1. Exit the Browser
      2. Query Mode
      3. Display Category/Component Context
      4. Display Items Most Recently Visited
      5. Go To Category or Component
      6. Display Category/Component Hierarchy
      7. Display Attributes
      8. Editing Commands
      9. Display Category Differences
      10. Display Component Differences
      11. Invoke Classifier
      12. Provide Librarian Guidance

Enter number of desired command <CR>:
```

5.2.2.1. Exiting the Library

For the administrator version of the Librarian, there are three main ways of exiting the library. The administrator may save his changes, ignore the changes, or destroy the library. If he saves his changes, the library structure is saved as it was when he exits except that no record is made of where he was in the library, nor is any record made of the advice he had received. If the administrator opts to have his changes ignored, the state of the library will remain the same as when he entered it, except that any modifications to existing text and integer values of attribute fillers will be in effect. If he opts to destroy the library, he will be asked for confirmation. If confirmed, there will be no record of the library after destruction. THESE COMMANDS ARE NOT

PROTECTED. BACKUPS ARE NOT MADE.

The user version of the Librarian simply exits, since the user has not been given any opportunity to make changes to the library.

```
      1.  Exit the Browser
...

Enter number of desired command <CR>: 1

What do you want to do?


      1.  Exit - save changes.
      2.  Exit - ignore changes from this session.
      3.  Destroy this library.
      4.  Resume editing.

Enter number of desired command <CR>: 2

Exiting...
```

5.2.2.2. Query Mode

Query mode allows the user to request a list of units conforming to a specified network structure. This is useful for quickly identifying units having specific desired characteristics. The syntax of the query language is described in an appendix to this user manual.

```
...
      2.  Query Mode
...

Enter number of desired command <CR>: 2
1> retrieve procedure_call_benchmark
>           with iteration_schemes having type fixed_loop_within_test.
ind000_acec_benchmark_drpca2
ind000_acec_benchmark_nrpca2
ind000_acec_benchmark_prpca2
ind000_acec_benchmark_nppca2
ind000_acec_benchmark_nulla2
2> quit.
```

5.2.2.3. Category and Component Context Display Commands

These commands are available after choosing "Display Category/Component Context" from the top level menu at a category.

```
CURRENT CATEGORY --> ada_benchmark
...
      3. Display Category/Component Context
...

Enter number of desired command <CR>: 3
What do you want to display?

      1. Immediate Super Categories
      2. Immediate Sub Categories
      3. Attributes of Category
      4. Categories Having Attributes that Reference this Category
      5. Components Having Attributes that Reference this Category
      6. *** Return to Previous Menu ***

Enter number of desired command <CR>:
```

The following are the context commands available for a component. For commands in common between the category and component context display, we will show one example from either category display or component display.

```
CURRENT UNIT --> ind000_piwg_performance_benchmark_p000006
...
      3. Display Category/Component Context
...

Enter number of desired command <CR>: 3
What do you want to display?

      1. Immediate Super Categories
      2. Attributes of Unit
      3. Units with this Unit as Attribute
      4. *** Return to Previous Menu ***

Enter number of desired command <CR>:
```

Immediate Super Categories

This command lists the next more general categories in the hierarchy than the current category or component.

```
CURRENT UNIT --> ind000_piwg_performance_benchmark_p000006
...
What do you want to display?

      1. Immediate Super Categories
...

Enter number of desired command <CR>: 1

Next More General Categories:
   parameterized_procedure_call_benchmark
   piwg_performance_benchmark
```

Immediate Sub Categories

This command lists the next more specific categories than the current category or component.

```
CURRENT CATEGORY --> ada_benchmark
...
What do you want to display?

...

      2. Immediate Sub Categories
...

Enter number of desired command <CR>: 2

Immediate Sub Categories:
   acec_benchmark
   capacity_benchmark
   composite_benchmark
   performance_benchmark
   piwg_benchmark
```

Attributes of Category or Component

This command shows the specifications for the current category or component. All subcategories and components MUST satisfy the specifications of their supercategory. Components may have their attributes satisfied by specific units in the library, and these are listed when known.

```
CURRENT UNIT --> ind000_piwg_performance_benchmark_p000006
...
What do you want to display?

...

      2. Attributes of Unit
...

Enter number of desired command <CR>: 2

Attributes/Components:
   auxiliary_required_components( 1 .. 1 ) of library_unit;
      Satisfied by:
         ind000_piwg_proc_package_6
   calls_procedure( 1 .. 1 ) of procedure;
      Satisfied by:
         ind000_proc_package_6_proc_0
   component_name( 1 .. 1 ) of name;
      Satisfied by:
         p000006
   description( 1 .. 1 ) of software_component_description;
      Satisfied by:
         ind000_p000006_description
   io_package( 1 .. 1 ) of piwg_io_package;
      Satisfied by:
         ind000_piwg_io_package
   iteration_control_package( 1 .. 1 ) of piwg_iteration_package;
      Satisfied by:
         ind000_piwg_iteration_package
   iteration_schemes( 1 .. 1 ) of clock_resolution_based_loop;
   location( 1 .. 1 ) of component_location;
      Satisfied by:
         ind000_p000006_location
...
```

Categories or Components Having Attributes that Reference this Category

These commands indicate those category or component attributes for which the current category serves as a value restriction. For example, selecting this option for the ada_benchmark category shows that "benchmark_generator" generates "ada_benchmarks".

```
CURRENT CATEGORY --> ada_benchmark
...
What do you want to display?

...

      4. Categories Having Attributes that Reference this Category
...

Enter number of desired command <CR>: 4

Categories described using this category:
   generates of benchmark_generator
```

Units with this Unit as Attributes

This command shows those component attributes for which the current category serves as a filler. For example, selecting this option for the "ind000_proc_package_6_proc_0" component shows that "ind000_piwg_performance_benchmark_p000006" calls this procedure.

```
CURRENT UNIT --> ind000_proc_package_6_proc_0
...
What do you want to display?

...

      3. Units with this Unit as Attribute
...

Enter number of desired command <CR>: 3

Satisfies attribute/component of Unit:
   calls_procedure of ind000_piwg_performance_benchmark_p000006
```

5.2.2.4. Display Items Most Recently Visited

This command displays a list of the categories and units in the library where the user has recently visited. This serves as a history mechanism to remind the user where he has been.

```
...
      4. Display Items Most Recently Visited
...

Enter number of desired command <CR>: 4

Categories/Components Most Recently Visited:
   ada_benchmark
   ind000_piwg_remote_global_package
   ind000_piwg_performance_benchmark_p000006
   clock_resolution_based_loop
   ind000_piwg_performance_benchmark_p000006
   performance_benchmark
   ada_benchmark
   thing
```

5.2.2.5. Go To Category or Unit

These commands provide a way to move around in the library. A user may go to any category or component in the library if he knows its name. Alternatively, the user may explore the library by moving to a category or component directly related to his current position in the library.

```
...
      5. Go To Category or Component
...

Enter number of desired command <CR>: 5

What category/component do you wish to examine?

      1. A user named category/component
      2. A previously visited category/component
      3. A generalization of the current category/component
      4. A specialization of the current category
      5. A component of the current category
      6. A value for an attribute of this category/component
      7. A type for an attribute of this category/component
      8. *** Return to Previous Menu ***

Enter number of desired command <CR>:
```

A User Named Category/Component

This is the most flexible means of moving through the library. The Librarian will prompt the user for the name of the category or component that he wishes to visit, and then take him there. Here we go to "ada_benchmark".

```
What category/component do you wish to examine?

      1. A user named category/component
...

Enter number of desired command <CR>: 1
category/component> ada_benchmark

CURRENT CATEGORY --> ada_benchmark
...
```

A Previously Visited Category or Component

This command displays a menu of the most recently visited categories and components. The user selects from this menu his desired location in the library, and the Librarian moves him there.

```
What category/component do you wish to examine?

...
      2. A previously visited category/component
...

Enter number of desired command <CR>: 2

Categories/Units Most Recently Visited:

Which concept do you wish to visit?

      1. ada_benchmark
      2. ind000_piwg_remote_global_package
      3. ind000_piwg_performance_benchmark_p000006
      4. clock_resolution_based_loop
      5. ind000_piwg_performance_benchmark_p000006
      6. performance_benchmark
      7. ada_benchmark
      8. thing
      9. *** Return to Previous Menu ***

Enter number of desired command <CR>: 3

CURRENT UNIT --> ind000_piwg_performance_benchmark_p000006
...
```

A Generalization of the Current Category or Component

This command displays a menu of the direct supercategories of the current category or component. The Librarian will move the user to the category he selects from this menu.

```
CURRENT UNIT --> ind000_piwg_performance_benchmark_p000006
...
What category/component do you wish to examine?

...
      3. A generalization of the current category/component
...

Enter number of desired command <CR>: 3

Which generalization of the current concept do you wish to visit?

      1. parameterized_procedure_call_benchmark
      2. piwg_performance_benchmark
      3. *** Return to Previous Menu ***

Enter number of desired command <CR>: 1

CURRENT CATEGORY --> parameterized_procedure_call_benchmark
...
```

A Specialization of the Current Category

This command displays a menu of the direct subcategories of the current category. The Librarian will move the user to the category he selects from this menu.

```
CURRENT CATEGORY --> ada_benchmark
...
What category/component do you wish to examine?

...
      4. A specialization of the current category
...

Enter number of desired command <CR>: 4

Which specialization of the current concept do you wish to visit?

      1. acec_benchmark
      2. capacity_benchmark
      3. composite_benchmark
      4. performance_benchmark
      5. piwg_benchmark
      6. *** Return to Previous Menu ***

Enter number of desired command <CR>: 4

CURRENT CATEGORY --> performance_benchmark
...
```

A Component of the Current Category

This command displays a menu of the components directly in the current category. The user may select one of the options from the menu, and the Librarian will move him to that component.

```
CURRENT CATEGORY --> ada_benchmark
...
What category/component do you wish to examine?

...
      5. A component of the current category
...

Enter number of desired command <CR>: 5

Which component?

      1. ind000_piwg_performance_benchmark_p000005
      2. ind000_piwg_performance_benchmark_p000006
      3. ind000_piwg_performance_benchmark_p000007
      4. ind000_piwg_performance_benchmark_p000010
      5. ind000_piwg_performance_benchmark_p000011
      6. ind000_piwg_performance_benchmark_p000012
      7. ind000_piwg_performance_benchmark_p000013
      8. *** Return to Previous Menu ***

Enter number of desired command <CR>: 2

CURRENT UNIT --> ind000_piwg_performance_benchmark_p000006
...
```

A Value for an Attribute of this Component

Since only components have attribute values, this command is only useful at components. The Librarian will first display a menu of the current component's attributes. Once the user has selected an attribute, the values for that attribute will be displayed. The Librarian will then move the user to the component he selects from this menu.

```
CURRENT UNIT --> ind000_piwg_performance_benchmark_p000006
...
What category/component do you wish to examine?

...
      6. A value for an attribute of this category/component
...

Enter number of desired command <CR>: 6

Which attribute?

      1. optimizable_via_inlining
      2. calls_procedure
      3. measured_features
      4. parameters
      5. recursive
      6. iteration_schemes
      7. optimization_control_package
      8. iteration_control_package
      9. io_package
      10. component_name
      11. description
      12. location
      13. auxiliary_required_components
      14. subunits
      15. *** Return to Previous Menu ***

Enter number of desired command <CR>: 7

Which value?

      1. ind000_piwg_remote_global_package
      2. *** Return to Previous Menu ***

Enter number of desired command <CR>: 1

CURRENT UNIT --> ind000_piwg_remote_global_package
...
```

A Type for an Attribute of this Category or Component

This command allows the user to move to the type restriction for an attribute of the current category or component. The Librarian will display a menu of the current component's attributes. Once the user has selected an attribute, the Librarian will move the user to the category that is the type restriction of the selected attribute.

```
CURRENT CATEGORY --> ada_benchmark
...
What category/component do you wish to examine?

...
      7. A type for an attribute of this category/component
...

Enter number of desired command <CR>: 7

Which attribute?

      1. parameters
      2. recursive
      3. measured_features
      4. component_name
      5. description
      6. location
      7. auxiliary_required_components
      8. subunits
      9. *** Return to Previous Menu ***

Enter number of desired command <CR>: 7

CURRENT CATEGORY --> library_unit
...
```

5.2.2.6. Category and Unit Hierarchy Display Commands

These are the commands available after choosing "Display Category/Component Hierarchy" from the top level menu while the current position is a category.

```
What display command do you want?

      1. All Components Satisfying Current Category
      2. Super Categories
      3. Sub Categories
      4. Common Super Categories   - any two categories
      5. Common Sub Categories - any two categories
      6. Common Components         - any two categories
      7. Display entire library hierarchy (depth-first)
      8. SNDL dump entire library hierarchy to file (depth-first)
      9. SNDL dump library hierarchy to file starting here
       (depth-first)
      10. *** Return to Previous Menu ***

Enter number of desired command <CR>:
```

These are the hierarchy display commands available from a component. Because they are a subset of those for a category, we will not show examples of these commands for both categories and components.

```
What display command do you want?

      1. Super Categories
      2. Common Super Categories   - any two categories
      3. Display entire library hierarchy (depth-first)
      4. SNDL dump entire library hierarchy to file (depth-first)
      5. *** Return to Previous Menu ***

Enter number of desired command <CR>:
```

All Components Satisfying Current Category

This command retrieves all components in the library that satisfy the specifications of the current category. Usually, this will produce a listing of software units that have been explicitly installed in the library. In some cases, there may be generators that can produce components that match the current specification, although all of the components that such a generator could produce are not explicitly represented. This shows up in the current example where "bgt_generated_composite_benchmark" describes a form of "ada_benchmark" that can be generated by "ind000_bgt".

```
CURRENT CATEGORY --> ada_benchmark
...
What display command do you want?

      1. All Components Satisfying Current Category
...

Enter number of desired command <CR>: 1
```

```
Generator--> ind000_bgt generates bgt_generated_composite_benchmark
Matching Units
   ind000_acec_benchmark_addsa2
   ind000_acec_benchmark_drpca2
   ind000_acec_benchmark_nppca2
   ind000_acec_benchmark_nrpca2
   ind000_acec_benchmark_nulla2
   ind000_acec_benchmark_prpca2
   ind000_acec_benchmark_pruaa2
   ind000_acec_benchmark_wheta2
   ind000_acec_benchmark_wheta3
   ind000_piwg_composite_benchmark_a000091
   ind000_piwg_composite_benchmark_a000092
   ind000_piwg_composite_benchmark_a000093
   ind000_piwg_composite_benchmark_a000094
   ind000_piwg_performance_benchmark_a000098
   ind000_piwg_performance_benchmark_a000099
   ind000_piwg_performance_benchmark_1000001
   ind000_piwg_performance_benchmark_1000002
   ind000_piwg_performance_benchmark_1000003
   ind000_piwg_performance_benchmark_p000001
   ind000_piwg_performance_benchmark_p000002
   ind000_piwg_performance_benchmark_p000003
   ind000_piwg_performance_benchmark_p000004
   ind000_piwg_performance_benchmark_p000005
   ind000_piwg_performance_benchmark_p000006
   ind000_piwg_performance_benchmark_p000007
   ind000_piwg_performance_benchmark_p000010
   ind000_piwg_performance_benchmark_p000011
   ind000_piwg_performance_benchmark_p000012
   ind000_piwg_performance_benchmark_p000013
```

Super Categories

This command prints out all ancestors of the current category up to the root of the hierarchy in alphabetical order.

```
CURRENT CATEGORY --> ada_benchmark
...
What display command do you want?

...
      2. Super Categories
...

Enter number of desired command <CR>: 2

Ancestors:
   ada_entity
   library_unit
   subprogram
   thing
```

Sub Categories

Typing this command causes a list of all of the subcategories of the current category to be produced with each successive level in the taxonomy being further indented. Because a category may be a direct subcategory of more than one category, there may be duplication of information. For instance, "acec_composite_benchmark" shows up under both "acec_benchmark" and "composite_benchmark". Because each interconnected subcategory may have many subcategories of its own, one should use this command with discretion for libraries with many interconnections.

```
CURRENT CATEGORY --> ada_benchmark
...
What display command do you want?

...
      3. Sub Categories
...

Enter number of desired command <CR>: 3
ada_benchmark
   acec_benchmark
      acec_composite_benchmark
      acec_performance_benchmark
   capacity_benchmark
   composite_benchmark
      acec_composite_benchmark
      bgt_generated_composite_benchmark
      piwg_composite_benchmark
   performance_benchmark
      acec_performance_benchmark
      arithmetic_optimization_benchmark
         fixed_point_arithmetic_benchmark
         floating_point_arithmetic_benchmark
         integer_arithmetic_benchmark
      assignment_benchmark
      control_structure_benchmark
         case_statement_benchmark
         looping_benchmark
            loop_optimization_benchmark
            unoptimized_loop_benchmark
      dynamic_sized_variable_elaboration_benchmark
      exception_handling_benchmark
      io_benchmark
      piwg_performance_benchmark
      subprogram_call_benchmark
         procedure_call_benchmark
            parameterized_procedure_call_benchmark
            parameterless_procedure_call_benchmark
            recursive_procedure_call_benchmark
      tasking_benchmark
         task_creation_benchmark
      variable_access_benchmark
         variable_assignment_benchmark
         variable_reference_benchmark
            numeric_variable_reference_benchmark
```

```
                integer_reference_benchmark
                   global_integer_variable_reference
                   local_integer_variable_reference
                   uplevel_integer_variable_reference
             scalar_variable_reference_benchmark
             structured_variable_reference_benchmark
                array_component_reference_benchmark
                record_component_reference_benchmark
piwg_benchmark
   piwg_composite_benchmark
   piwg_performance_benchmark
```

Common Super Categories

This command prints out those categories which are ancestors of both of the input categories. It is particularly useful for discovering the shared specifications between two categories.

```
What display command do you want?

...
      4. Common Super Categories   - any two categories
...

Enter number of desired command <CR>: 4

Enter the first category name.
category-1> piwg_benchmark

Enter the second category name.
category-2> performance_benchmark

Common Ancestor Categories:
   ada_benchmark
   ada_entity
   library_unit
   subprogram
   thing
```

Common Sub Categories

This command prints out those categories which are subcategories of both of the input categories.

```
What display command do you want?

...
      5. Common Sub Categories - any two categories
...

Enter number of desired command <CR>: 5

Enter the first category name.
category> performance_benchmark

Enter the second category name.
category> piwg_benchmark

Common Descendants:
   piwg_performance_benchmark
```

Common Components

This command prints out those components which are members of both of the input categories.

```
What display command do you want?

...
      6. Common Components       - any two categories
...

Enter number of desired command <CR>: 6

Enter the first category name.
category> performance_benchmark

Enter the second category name.
category> piwg_benchmark
```

```
Common Units:
   ind000_piwg_performance_benchmark_a000098
   ind000_piwg_performance_benchmark_a000099
   ind000_piwg_performance_benchmark_1000001
   ind000_piwg_performance_benchmark_1000002
   ind000_piwg_performance_benchmark_1000003
   ind000_piwg_performance_benchmark_p000001
   ind000_piwg_performance_benchmark_p000002
   ind000_piwg_performance_benchmark_p000003
   ind000_piwg_performance_benchmark_p000004
   ind000_piwg_performance_benchmark_p000005
   ind000_piwg_performance_benchmark_p000006
   ind000_piwg_performance_benchmark_p000007
   ind000_piwg_performance_benchmark_p000010
   ind000_piwg_performance_benchmark_p000011
   ind000_piwg_performance_benchmark_p000012
   ind000_piwg_performance_benchmark_p000013
```

Display Entire Library Hierarchy

This command displays all of the categories in the library in the same format that "Sub Categories" does. It also duplicates information for those categories which have multiple immediate supercategories, so one should exercise caution when using this command. The output of this command for the benchmarks library is included in Appendix B.

SNDL Dumps

These commands print out the library in SNDL (Semantic Network Description Language) format. See the AdaKNET user's manual for a description of SNDL. These commands allow one to save all or parts of the network to a file in a system independent manner. At this time, these commands work slowly, so use them sparingly.

```
What display command do you want?

...
      9. SNDL dump library hierarchy to file starting here
       (depth-first)
...

Enter number of desired command <CR>: 9

To which file do you wish to write?
(To abort, type '*abort*').
file> sndl_dump_ada_benchmark
```

5.2.2.7. Attribute Display Commands

These are the commands available after choosing "Display Attributes" from the main menu of a category.

```
What do you want to display?

      1. print aggregation structure breadth first - simple type
      2. print aggregation structure depth first - simple type
      3. print aggregation structure breadth first - all types
      4. print aggregation structure depth first - all types
      5. print aggregation structure alphabetically by concept name
      6. *** Return to Previous Menu ***

Enter number of desired command <CR>:
```

These are the commands available after choosing "Component Display Commands" from the main menu of a unit display.

```
What display command do you want?

      1. Display required units in aggregation structure
      2. Display full text value of attribute
      3. *** Return to Previous Menu ***

Enter number of desired command <CR>:
```

Print Aggregation Structure

These commands print out all of the attributes of the current category and, recursively, all of the attributes of each of the value restrictions. Each command offers a different organizational scheme for the order in which the attributes are printed. The "all types" option will also point out the subcategories of each value restriction, in the same format that the "Sub Categories" hierarchy display command does.

```
CURRENT CATEGORY --> ada_benchmark
...
What do you want to display?

...
      5. print aggregation structure alphabetically by concept name
...

Enter number of desired command <CR>: 5
```

```
ada_benchmark
  Attributes/Components:
    auxiliary_required_components(0..infinity) of library_unit;
    component_name(1..1) of name;
    description(1..1) of software_component_description;
    location(1..1) of component_location;
    measured_features(0..infinity) of benchmark_feature;
    parameters(0..infinity) of ada_parameter_specification;
    recursive(1..1) of boolean;
    subunits(0..infinity) of software_subcomponent;


ada_data_type
  Attributes/Components: *none*

ada_parameter_specification
  Attributes/Components:
    mode(1..1) of parameter_mode;
    name(1..1) of identifier;
    position(1..1) of pos_int;
    type(1..1) of ada_data_type;

benchmark_feature
  Attributes/Components: *none*


boolean
  Attributes/Components: *none*

component_location
  Attributes/Components:
    directory_name(1..1) of text;
    file_name(1..1) of text;
    repository_name(1..1) of text;


identifier
  Attributes/Components: *none*

library_unit
  Attributes/Components:
    auxiliary_required_components(0..infinity) of library_unit;
    component_name(1..1) of name;
    description(1..1) of software_component_description;
    location(1..1) of component_location;
    subunits(0..infinity) of software_subcomponent;


name
  Attributes/Components: *none*

parameter_mode
  Attributes/Components: *none*

pos_int
  Attributes/Components: *none*
```

```
software_component_description
  Attributes/Components:
    abstract(1..1) of text;
    authors(0..infinity) of text;
    design(1..1) of text;
    restrictions(1..1) of text;

software_subcomponent
  Attributes/Components:
    auxiliary_required_components(0..infinity) of library_unit;
    component_name(1..1) of name;
    description(1..1) of software_component_description;
    location(1..1) of component_location;
    subunits(0..infinity) of software_subcomponent;

text
  Attributes/Components: *none*
```

Display Required Units in Aggregation Structure

This command prints out all of the required components of the current component. After this, it recursively prints out the required components of each of these. These are organized alphabetically by the name of the satisfying components (fillers).

```
CURRENT UNIT --> ind000_p000006_location
...
What display command do you want?

...
      1. Display required units in aggregation structure
...

Enter number of desired command <CR>: 1
```

```
ind000_p000006_location
Required Fillers:
    directory_name(1..1) of text;
        Satisfied by:
            demo_dir
    file_name(1..1) of text;
        Satisfied by:
            p000006_a
    repository_name(1..1) of text;
        Satisfied by:
            simtel_20

demo_dir
Required Fillers: *none*

p000006_a
Required Fillers: *none*

simtel_20
Required Fillers: *none*
```

Display Full Text Value of Attribute

This command is only useful for text valued attributes. First, the Librarian will offer a menu of attributes with text values. It will then display the full text value of the selected attribute, which may be too long for full display by the "Attributes" context display command. The user may scroll forward and back in this text display by typing space and "b", respectively. The user exists the display by typing "q".

```
CURRENT UNIT --> ind000_pruaa?_description
...
What display command do you want?

...
      2. Display full text value of attribute
...

Enter number of desired command <CR>: 2

Which attribute?

      1. abstract
      2. design
      3. *** Exit This Menu ***

Enter number of desired command <CR>: 1
```

```
This test evaluates the code efficiency of a reference from within a
procedure to an uplevel variable, i.e., a variable declared in a
declarative region that statically encloses the procedure.  The referenc
is to a variable declared in the immediately enclosing region (one
level up).
(END)
```

5.2.2.8. Editing Commands

These are the commands available after choosing "Editing Commands" from the main menu of a category display. This option is only included in the administrator version of the Librarian. These commands are not protected in any way, so be careful.

```
What do you want to do?

      1. add subcategory
      2. remove subcategory
      3. add software unit
      4. add software unit after qualifying with Gadfly
      5. remove unit
      6. add attribute
      7. add sub-attribute
      8. narrow attribute definition
      9. remove attribute
      10. rename current category
      11. rename attribute
      12. *** Return to Previous Menu ***

Enter number of desired command <CR>:
```

The following are the commands available after choosing "Editing Commands" from the main menu of a component display. They are essentially a subset of those for categories, so we will not show examples of the same command working for both categories and components.

```
What do you want to do?

      1. *** Return to Previous Menu ***
      2. narrow attribute definition
      3. rename attribute
      4. specify attribute value
      5. remove attribute value
      6. rename current unit
      7. remove integer attribute value
      8. remove text attribute value
      9. change integer attribute value
      10. change text attribute value

Enter number of desired command <CR>:
```

Add Subcategory

This command adds a subcategory to the current category. If the subcategory does not already exist, it is created. If the subcategory name already exists, the Librarian checks to see whether it should really be made a subcategory here as well, i.e., whether the user really wants an interconnection between different parts of the library. Notice that the new subcategory automatically *inherits* the attributes and components from its parent category, ada_benchmark.

```
CURRENT CATEGORY --> ada_benchmark
...

What do you want to do?

      1.  add subcategory
...

Enter number of desired command <CR>: 1

What do you want the new subcategory to be called?
(To abort, type '*abort*').
> my_benchmark


...
      5.  Go To Category or Component
...

Enter number of desired command <CR>: 5

What category/component do you wish to examine?

...
      4. A specialization of the current category
...

Enter number of desired command <CR>: 4

Which specialization of the current concept do you wish to visit?

...
      4. my_benchmark
...

Enter number of desired command <CR>: 4
```

```
CURRENT CATEGORY --> my_benchmark

...
      3. Display Category/Component Context
...

Enter number of desired command <CR>: 3

What do you want to display?

...
      3. Attributes of Category
...

Enter number of desired command <CR>: 3


Attributes:
   auxiliary_required_components( 0 .. infinity ) of library_unit;
   component_name( 1 .. 1 ) of name;
   description( 1 .. 1 ) of software_component_description;
   location( 1 .. 1 ) of component_location;
   measured_features( 0 .. infinity ) of benchmark_feature;
   parameters( 0 .. infinity ) of ada_parameter_specification;
   recursive( 1 .. 1 ) of boolean;
   subunits( 0 .. infinity ) of software_subcomponent;
```

Remove Subcategory or Component

These commands remove a subcategory or component of the current category. This operation is not allowed when there is other library structure dependent on the subcategory targeted, e.g., when the targeted subcategory itself has subcategories.

```
CURRENT CATEGORY --> ada_benchmark
...
What do you want to do?

...
      2.  remove subcategory
...

Enter number of desired command <CR>: 2

Which category?

      1. acec_benchmark
      2. piwg_benchmark
      3. capacity_benchmark
      4. composite_benchmark
      5. performance_benchmark
      6. my_benchmark
      7. *** Return to Previous Menu ***

Enter number of desired command <CR>: 5

Cannot remove category without affecting other structure.

What do you want to do?

...
      2.  remove subcategory
...

Enter number of desired command <CR>: 2

Which category?

      1. acec_benchmark
      2. piwg_benchmark
      3. capacity_benchmark
      4. composite_benchmark
      5. performance_benchmark
      6. my_benchmark
      7. *** Return to Previous Menu ***

Enter number of desired command <CR>: 3

capacity_benchmark has been removed from the network.
```

Add Software Unit

This command adds a software unit to the current category. In doing this, it attempts to fulfill all of the inherited specifications, recursively. This may require adding additional subcomponents to the library or choosing from existing subcomponents.

Here, we add a new unit to the category "my_benchmark". Some parts of this transcript that show nothing new or interesting have been deleted in order to conserve space. These excised portions are indicated by ellipses (...).

```
CURRENT CATEGORY --> my_benchmark
...
What do you want to do?

...
      3. add software unit
...

Enter number of desired command <CR>: 3

What do you want the my_benchmark to be called?
(To abort, type '*abort*').
> kinder_gentler_benchmark


As a my_benchmark, kinder_gentler_benchmark has 1 component_name
 of kind name.

Would you like to further specify kinder_gentler_benchmark's
 component_name?

      1. Further specify the kind of component_name
      2. Specify a value for the component_name
      3. Leave the component_name specification as is

Type in the number(s) of your choice(s), all on one line, separated
 by blanks:
> 3


As a my_benchmark, kinder_gentler_benchmark has 1 description of
 kind software_component_description.

Would you like to further specify kinder_gentler_benchmark's
 description?

      1. Specify a value for the description
      2. Leave the description specification as is

Type in the number(s) of your choice(s), all on one line, separated
 by blanks:
> 2
```

Next, we will fill in a value for the location of our new benchmark. The component_location has some attributes itself (e.g. directory_name), so we attempt to fill those as well. After doing this, we return to filling the attributes of kinder_gentler_benchmark.

```
As a my_benchmark, kinder_gentler_benchmark has 1 location of kind
 component_location.

Would you like to further specify kinder_gentler_benchmark's location?

      1. Specify a value for the location
      2. Leave the location specification as is

Type in the number(s) of your choice(s), all on one line, separated
 by blanks:
> 1


What would you like to do to specify a value for the location?

      1. Specify the value
      2. Browse to find the value
      3. Add the value to the library
      4. Leave the value unspecified

Enter number of desired command <CR>: 3

What is the name of the new component_location?
> kind_location


As a component_location, kind_location has 1 directory_name of kind
 text.

Would you like to further specify kind_location's directory_name?

      1. Specify a value for the directory_name
      2. Leave the directory_name specification as is

Type in the number(s) of your choice(s), all on one line, separated
 by blanks:
> 2
```

```
...
As a component_location, kind_location has 1 repository_name
 of kind text.

Would you like to further specify kind_location's repository_name?

      1. Specify a value for the repository_name
      2. Leave the repository_name specification as is

Type in the number(s) of your choice(s), all on one line, separated
 by blanks:
> 1


What would you like to do to specify a value for the repository_name?

      1. Specify the value
      2. Browse to find the value
      3. Add the value to the library
      4. Leave the value unspecified

Enter number of desired command <CR>: 3


What is the name of the new text?
> my_repository

my_repository has been added to the library.


Resuming the addition of kind_location to the library.

kind_location has been added to the library.

Resuming the addition of kinder_gentler_benchmark to the library.


As a my_benchmark, kinder_gentler_benchmark has 0 or more
 auxiliary_required_components of kind library_unit.

Would you like to further specify kinder_gentler_benchmark's
 auxiliary_required_components?

      1. Further specify the number of auxiliary_required_components
      2. Further specify the kind of auxiliary_required_components
      3. Specify a value for the auxiliary_required_components
      4. Leave the auxiliary_required_components specification as is

Type in the number(s) of your choice(s), all on one line, separated
 by blanks:
> 1
```

```
Current Min:  0     Current Max:  infinity

Enter the new range at the prompts.
The min must be >= the current min and <= the current max.
The max must be >= the min and <= the current max.
If you want a max of infinity, enter '-1' at the 'max' prompt.
min> 0
max> 0

...

As a my_benchmark, kinder_gentler_benchmark has 1 recursive of kind
 boolean.

Would you like to further specify kinder_gentler_benchmark's
 recursive?

      1. Further specify the kind of recursive
      2. Leave the recursive specification as is

Type in the number(s) of your choice(s), all on one line, separated
 by blanks:
> 1


What would you like to do to further specify the kind of recursive?

      1. Specify the new kind
      2. Browse to find the new kind
      3. Leave the kind as is

Enter number of desired command <CR>: 1

current type: boolean
new type> false


...
As a my_benchmark, kinder_gentler_benchmark has 0 or more
 measured_features of kind benchmark_feature.

Would you like to further specify kinder_gentler_benchmark's
 measured_features?

      1. Further specify the number of measured_features
      2. Further specify the kind of measured_features
      3. Leave the measured_features specification as is

Type in the number(s) of your choice(s), all on one line, separated
 by blanks:
> 2
```

```
What would you like to do to further specify the kind of
 measured_features?

      1. Specify the new kind
      2. Browse to find the new kind
      3. Leave the kind as is

Enter number of desired command <CR>: 2
```

One may not be familiar with enough of the library when adding a subcomponent of a system. For this reason, it is possible to enter a special "Mini-browser" when adding units. In this example, the user needs to familiarize himself with the available benchmark_features before specifying one.

```
CURRENT CATEGORY --> benchmark_feature
Next More General Categories:
    db_domain
Next More Specific Categories:
    language_feature
    system_capacity
Attributes/Components: *none*
Delimits attribute/component of Category:
    measured_features of ada_benchmark
    measured_features of acec_composite_benchmark
    measured_features of piwg_composite_benchmark
Delimits attribute/component of Unit: *none*

What kind of command do you want?

      1. Go to Category
      2. Invoke Classifier
      3. Display Descendant Categories
      4. Display Matching Units
      5. Quit Browsing

Enter number of desired command <CR>: 3
benchmark_feature
    language_feature
        block_statements
        control_feature
            case_statements
            loop_statements
        declarations
            access_types_declarations_and_object_declarations
            array_types_declarations_and_object_declarations
            record_types_declarations_and_object_declarations
            scalar_types_declarations_and_object_declarations
        identifiers
        indexed_components
        local_names_and_references_to_objects
        non_local_names_and_references_to_objects
```

```
        operators_and_expression_evaluation
            assignment_statements
            binary_adding_operators
            highest_precedence_operators
            logical_operators_and_short_circuit_control_forms
            multiplying_operators
            relational_operators_and_membership_tests
        package_bodies
        package_specifications_and_declarations
        pragma_suppress
        record_aggregates
        record_representation_clauses
        references_to_local_and_non_local_objects
        selected_components
        subprogram_calls
            parameter_associations
            procedure_call
        tasking
            entries_entry_calls_and_accept_statements
            task_and_entry_attributes
            task_execution_task_activation
            task_types_and_task_objects
    system_capacity
```

```
CURRENT CATEGORY --> benchmark_feature
Next More General Categories:
    db_domain
Next More Specific Categories:
    language_feature
    system_capacity
Attributes/Components: *none*
Delimits attribute/component of Category:
    measured_features of ada_benchmark
    measured_features of acec_composite_benchmark
    measured_features of piwg_composite_benchmark
Delimits attribute/component of Unit: *none*

What kind of command do you want?

      1. Go to Category
      2. Invoke Classifier
      3. Display Descendant Categories
      4. Display Matching Units
      5. Quit Browsing

Enter number of desired command <CR>: 5
```

```
What do you want to do?

      1. Specify Category To Be Used As Kind
      2. Quit Browsing Without Specifying Kind
      3. Resume Browsing

Enter number of desired command <CR>: 1
new type> system_capacity

kinder_gentler_benchmark has been added to the library.


...
CURRENT CATEGORY --> my_benchmark

...
      5. Go To Category or Unit
...

Enter number of desired command <CR>: 5

What category/component do you wish to examine?

...
      5. A component of the current category
...

Enter number of desired command <CR>: 5

Which component?

      1. kinder_gentler_benchmark
      2. *** Return to Previous Menu ***

Enter number of desired command <CR>: 1


CURRENT UNIT --> kinder_gentler_benchmark

...
      3. Display Category/Component Context
...

Enter number of desired command <CR>: 3

What do you want to display?

...
      2. Attributes of Unit
...

Enter number of desired command <CR>: 2
```

```
Attributes:
    associated_something(0..50) of function;
        Satisfied by: *none*
    auxiliary_required_components(0..0) of library_unit;
    component_name(1..1) of name;
        Satisfied by: *none*
    description(1..1) of software_component_description;
        Satisfied by: *none*
    location(1..1) of component_location;
        Satisfied by:
            kind_location
    measured_features(0..infinity) of system_capacity;
    parameters(0..0) of ada_parameter_specification;
    recursive(1..1) of false;
    subunits(0..0) of software_subcomponent;


...
CURRENT UNIT --> kinder_gentler_benchmark

...
      5. Go To Category or Unit
...

Enter number of desired command <CR>: 5

What category/component do you wish to examine?

      1. A user named category/component
...

Enter number of desired command <CR>: 1
category/component> kind_location


CURRENT UNIT --> kind_location

...
      3. Display Category/Component Context
...

Enter number of desired command <CR>: 3

What do you want to display?

...
      2. Attributes of Unit
...

Enter number of desired command <CR>: 2
```

```
Attributes:
    directory_name(1..1) of text;
        Satisfied by: *none*
    file_name(1..1) of text;
        Satisfied by: *none*
    repository_name(1..1) of text;
        Satisfied by:
            my_repository
```

Add Software Unit After Qualifying with Gadfly

This option forces the user to run Gadfly prior to adding his software unit to the library. (See the Gadfly User's Manual for a description of how Gadfly works.) This option has not been implemented, because Gadfly's mode of operation is inapplicable to the benchmark domain. Gadfly concentrates on testing parameters; most benchmarks have none.

Add Attribute or Sub-Attribute

The "Add Attribute" command allows one to add attributes to a category. All subcategories and components of this category will automatically receive these attributes as well.

The "Add Sub-Attribute" command allows one to add sub-attributes to an attribute of a category or component. A sub-attribute describes a subset of the values of its super-attribute. As with attributes, all subcategories and components of a category will automatically receive new sub-attributes.

```
What do you want to do?

...
      6. add attribute
...

Enter number of desired command <CR>: 6

What do you want the new attribute/component to be called?
(To abort, type '*abort*').
> associated_something


Enter the range at the prompts.  The min must be >= 0.  The max must
 be >= the min.  If you want a max of infinity, enter '-1' at the
 'max' prompt.
min> 0
max> -1

filler type> thing

associated_something
```

```
...
CURRENT CATEGORY --> my_benchmark

...
      3. Display Category/Component Context
...

Enter number of desired command <CR>: 3

What do you want to display?

...
      3. Attributes of Category
...

Enter number of desired command <CR>: 3


Attributes:
    associated_something(0..infinity) of thing;
    auxiliary_required_components(0..infinity) of library_unit;
    component_name(1..1) of name;
    description(1..1) of software_component_description;
    location(1..1) of component_location;
    measured_features(0..infinity) of benchmark_feature;
    parameters(0..infinity) of ada_parameter_specification;
    recursive(1..1) of boolean;
    subunits(0..infinity) of software_subcomponent;
```

Narrow Attribute Definition

This command allows one to more narrowly specify the attributes of a category or component. All subcategories and units of a category will automatically receive the narrowed attributes as well.

```
What do you want to do?

...
      8. narrow attribute definition
...

Enter number of desired command <CR>: 8

Which attribute?

      1. measured_features
      2. parameters
      3. recursive
      4. associated_something
      5. subunits
      6. auxiliary_required_components
      7. location
      8. description
      9. component_name
      10. *** Return to Previous Menu ***

Enter number of desired command <CR>: 4


Do you want to restrict the range?y

Current Min:  0     Current Max:  infinity

Enter the new range at the prompts.
The min must be >= the current min and <= the current max.
The max must be >= the min and <= the current max.
If you want a max of infinity, enter '-1' at the 'max' prompt.
min> 0
max> 50


Do you want to restrict the filler type?y

current type: thing
new type> library_unit

associated_something has been restricted.
```

```
...
CURRENT CATEGORY --> my_benchmark

...
      3. Display Category/Component Context
...

Enter number of desired command <CR>: 3

What do you want to display?

...
      3. Attributes of Category
...

Enter number of desired command <CR>: 3


Attributes:
    associated_something(0..50) of library_unit;
    auxiliary_required_components(0..infinity) of library_unit;
    component_name(1..1) of name;
    description(1..1) of software_component_description;
    location(1..1) of component_location;
    measured_features(0..infinity) of benchmark_feature;
    parameters(0..infinity) of ada_parameter_specification;
    recursive(1..1) of boolean;
    subunits(0..infinity) of software_subcomponent;
```

Remove Attribute

This command removes an attribute from a category. This operation is not allowed when there is other library structure dependent on the attribute, e.g., when the targeted attribute has been filled at a component. The attribute will be completely removed from the library, including any other categories and components which had the attribute.

```
CURRENT CATEGORY --> ada_benchmark
...
What do you want to do?

...
      9.  remove attribute
...

Enter number of desired command <CR>: 2

Which attribute?

      1. measured_features
      2. parameters
      3. recursive
      4. associated_something
      5. subunits
      6. auxiliary_required_components
      7. location
      8. description
      9. component_name
      10. *** Return to Previous Menu ***

Enter number of desired command <CR>: 4

associated_something has been removed.
```

Renaming a Category/Unit

It is a simple matter to rename a category or component. The only requirement is that the new name may not be the name of any other category or component in the library.

```
CURRENT UNIT --> ind000_piwg_performance_benchmark_p000006
...

What do you want to do?

...
      6.  rename current unit
...

Enter number of desired command <CR>: 6
new unit name> just_a_benchmark
```

Renaming an Attribute

This command changes the name of an attribute. Because the attribute may be inherited from some supercategory, it is important to realize that this may not be a strictly local change; one may also be changing the name of that attribute for one or more supercategories and their descendants.

```
CURRENT UNIT --> just_a_benchmark
...

What do you want to do?

...
      3. rename attribute
...

Enter number of desired command <CR>: 3

Which roleset?

      1. optimizable_via_inlining
      2. calls_procedure
      3. measured_features
      4. parameters
      5. recursive
      6. iteration_schemes
      7. optimization_control_package
      8. iteration_control_package
      9. io_package
      10. subunits
      11. auxiliary_required_components
      12. location
      13. description
      14. component_name
      15. *** Return to Previous Menu ***

Enter number of desired command <CR>: 1
new attribute name> inlinable


CURRENT UNIT --> just_a_benchmark

...
      3. Display Category/Component Context
...

Enter number of desired command <CR>: 3

What do you want to display?

...
      2. Attributes of Unit
...

Enter number of desired command <CR>: 2
```

```
Attributes:
    auxiliary_required_components(1..1) of library_unit;
        Satisfied by:
            ind000_piwg_proc_package_6
    calls_procedure(1..1) of procedure;
        Satisfied by:
            ind000_proc_package_6_proc_0
    component_name(1..1) of name;
        Satisfied by:
            p000006
    description(1..1) of software_component_description;
        Satisfied by:
            ind000_p000006_description
    inlinable(1..1) of false;
    io_package(1..1) of piwg_io_package;
        Satisfied by:
            ind000_piwg_io_package
...
```

5.2.2.9. Display Category Differences

This command displays the difference between two categories based on network structure. It is useful for highlighting the distinguishing characteristics between two categories by printing out the differences between the attributes that the two categories possess.

```
...
      9.  Display Category Differences
...

Enter number of desired command <CR>: 9
category-1> acec_benchmark
category-2> piwg_benchmark


a acec_benchmark has the following characteristics which a
 piwg_benchmark does not have:

   1 instrument_package, which is a acec_instrument_package.
   1 control_measurement_component, which is a library_unit.
```

5.2.2.10. Display Component Differences

This command displays the difference between two components based on network structure. It is useful for highlighting the distinguishing characteristics between two components by printing out the differences between the attributes and satisfying values that the two components possess.

```
...
      10. Display Component Differences
...

Enter number of desired command <CR>: 10
unit-1> ind000_piwg_performance_benchmark_p000006
unit-2> ind000_piwg_performance_benchmark_p000007


ind000_piwg_performance_benchmark_p000006 and
 ind000_piwg_performance_benchmark_p000007 differ in the following
 characteristics:

   ind000_piwg_performance_benchmark_p000006's component_name include:
      p000006
   while ind000_piwg_performance_benchmark_p000007's include:
      p000007
   ind000_piwg_performance_benchmark_p000006's description include:
      ind000_p000006_description
   while ind000_piwg_performance_benchmark_p000007's include:
      ind000_p000007_description
   ind000_piwg_performance_benchmark_p000006's location include:
      ind000_p000006_location
   while ind000_piwg_performance_benchmark_p000007's include:
      ind000_p000007_location
   ind000_piwg_performance_benchmark_p000006's
    auxiliary_required_components include:
      ind000_piwg_proc_package_6
   while ind000_piwg_performance_benchmark_p000007's include:
      ind000_piwg_proc_package_7
   ind000_piwg_performance_benchmark_p000006's calls_procedure
    include:
      ind000_proc_package_6_proc_0
   while ind000_piwg_performance_benchmark_p000007's include:
      ind000_proc_package_7_proc_0
```

5.2.2.11. Invoke Classifier

The classifier provides high speed navigation through the category hierarchy. It has the user choose the category to visit from the next level of the taxonomy until there are no more levels in the hierarchy, or until the user decides to stop by choosing "None of the above". This is particularly useful for a user who is moderately familiar with the library. The excerpt below shows what happens when the classifier is invoked at "ada_benchmark".

```
Which of the following more specifically describes the ada_benchmark?

      1.  acec_benchmark
      2.  piwg_benchmark
      3.  capacity_benchmark
      4.  composite_benchmark
      5.  performance_benchmark
      6.  my_benchmark
      7.  None of the above

Enter number of desired command <CR>: 5


Which of the following more specifically describes the
 performance_benchmark?

      1.  arithmetic_optimization_benchmark
      2.  assignment_benchmark
      3.  control_structure_benchmark
      4.  dynamic_sized_variable_elaboration_benchmark
      5.  exception_handling_benchmark
      6.  io_benchmark
      7.  subprogram_call_benchmark
      8.  tasking_benchmark
      9.  variable_access_benchmark
      10.  piwg_performance_benchmark
      11.  acec_performance_benchmark
      12.  None of the above

Enter number of desired command <CR>: 7


Which of the following more specifically describes the
 subprogram_call_benchmark?

      1.  procedure_call_benchmark
      2.  None of the above

Enter number of desired command <CR>: 1

Which of the following more specifically describes the
 procedure_call_benchmark?

      1.  parameterized_procedure_call_benchmark
      2.  parameterless_procedure_call_benchmark
      3.  recursive_procedure_call_benchmark
      4.  None of the above

Enter number of desired command <CR>: 1
```

5.2.2.12. Librarian Guidance

Librarian guidance helps determine which category or component to investigate next. It does this by asking questions of the user, and positioning him at an appropriate library category based on his responses. This command is particularly useful when exploring unfamiliar parts of the library.

Guidance may terminate in three ways. If no guidance is available at the category, the message "No guidance is available at this concept" will appear, and the user will remain at the category from which he invoked the command. If the Librarian can go no further based on questions it can ask the user, it will quietly exit, depositing the user at the most appropriate category based on previous answers. Lastly, if the Librarian can recommend particular components, it will do so. The user must confirm these choices, after which he will be placed in a category to which the components belong.

```
CURRENT CATEGORY --> piwg_composite_benchmark

...
      12.  Provide Librarian Guidance

Enter number of desired command <CR>: 8

Choose guidance mode

      1. Continuous guidance
      2. User interruptable guidance

Enter number of desired command <CR>: 1

Choose explanation mode

      1. Explain internal reasoning
      2. Explain questioning
      3. Explain possible moves through library
      4. Full explanations
      5. No explanations

Enter number of desired command <CR>: 5


Will this benchmark be used for numerical computations?


      1.  yes
      2.  no
      3.  not solely

Enter number of desired command <CR>: 1
```

```
Which response best describes the kind of computation done?


      1.  systems and program
      2.  science and math
      3.  classic and various computations

Enter number of desired command <CR>: 2


What type of math package does the benchmark utilize?


      1.  the manufacturer's package
      2.  the standard, internal package
      3.  another math package

Enter number of desired command <CR>: 2


The benchmark which bests suits your needs is
 ind000_piwg_composite_benchmark_a000093, which is a unit at this
 category.

      1.  confirm

Enter number of desired command <CR>: 1


CURRENT CATEGORY --> piwg_composite_benchmark
...
```

APPENDIX A: Glossary of AdaKNET Terminology

Concept – A *concept* is an AdaKNET object which models a thing (*individual concept*) or category of things (*generic concept*). Concepts are the principle objects in AdaKNET.

Filler – A roleset's *filler* is an individual concept which models a specific thing that plays the roleset's role for the roleset's owner. A role's fillers are the fillers of all the rolesets associated with the role.

Role – A *role* is an AdaKNET object which models an attribute.

Roleset – A *roleset* is an AdaKNET object which associates a role with a concept. The concept is called the roleset's owner. A roleset describes, for the thing or category of things modeled by the owner, what kind of things the fillers of the role can be and how many fillers the role can have. The kind of things the fillers can be is expressed by a generic concept called the roleset's *value restriction* or *type*. The number of fillers the role can have is expressed by a pair of numbers called the roleset's *range restriction* (or, simply *range*).

APPENDIX B: Library Hierarchy - Benchmarks

- thing
  - ada_entity
    - ada_data_type
    - ada_parameter_specification
      - ada_in_out_parameter_specification
      - ada_in_parameter_specification
      - ada_out_parameter_specification
    - assignment_statement
      - bgt_assignment_statement
    - constant_declaration
      - bgt_constant_declaration
    - enumerated_type_declaration
      - bgt_enumerated_type_declaration
    - library_unit
      - generic_library_unit
      - package
        - acec_instrument_package
        - bgt_package
          - bgt_package_specification
          - bgt_package_specification_body_pair
        - calendar
        - piwg_cpu_time_clock
        - piwg_io_package
        - piwg_iteration_package
        - piwg_optimization_control_package
      - software_subcomponent
      - subprogram
        - ada_benchmark
          - acec_benchmark
            - acec_composite_benchmark
            - acec_performance_benchmark
          - capacity_benchmark
          - composite_benchmark
            - acec_composite_benchmark
            - bgt_generated_composite_benchmark
            - piwg_composite_benchmark
          - performance_benchmark
            - acec_performance_benchmark
            - arithmetic_optimization_benchmark
              - fixed_point_arithmetic_benchmark
              - floating_point_arithmetic_benchmark
              - integer_arithmetic_benchmark
            - assignment_benchmark
            - control_structure_benchmark

```
            case_statement_benchmark
            looping_benchmark
              loop_optimization_benchmark
              unoptimized_loop_benchmark
          dynamic_sized_variable_elaboration_benchmark
          exception_handling_benchmark
          io_benchmark
          piwg_performance_benchmark
          subprogram_call_benchmark
            procedure_call_benchmark
              parameterized_procedure_call_benchmark
              parameterless_procedure_call_benchmark
              recursive_procedure_call_benchmark
          tasking_benchmark
            task_creation_benchmark
          variable_access_benchmark
            variable_assignment_benchmark
            variable_reference_benchmark
              numeric_variable_reference_benchmark
                integer_reference_benchmark
                  global_integer_variable_reference
                  local_integer_variable_reference
                  uplevel_integer_variable_reference
              scalar_variable_reference_benchmark
              structured_variable_reference_benchmark
                array_component_reference_benchmark
                record_component_reference_benchmark
        piwg_benchmark
          piwg_composite_benchmark
          piwg_performance_benchmark
      benchmark_generator
      function
      procedure
        bgt_procedure
        non_recursive_procedure
        recursive_procedure
      sort_routine
    name
      enumeration_literal
      subprogram_designator
        identifier
        operator_symbol
    object_declaration
      bgt_object_declaration
  component_location
  db_domain
    benchmark_feature
```

- language_feature
  - block_statements
  - control_feature
    - case_statements
    - loop_statements
  - declarations
    - access_types_declarations_and_object_declarations
    - array_types_declarations_and_object_declarations
    - record_types_declarations_and_object_declarations
    - scalar_types_declarations_and_object_declarations
  - identifiers
  - indexed_components
  - local_names_and_references_to_objects
  - non_local_names_and_references_to_objects
  - operators_and_expression_evaluation
    - assignment_statements
    - binary_adding_operators
    - highest_precedence_operators
    - logical_operators_and_short_circuit_control_forms
    - multiplying_operators
    - relational_operators_and_membership_tests
  - package_bodies
  - package_specifications_and_declarations
  - pragma_suppress
  - record_aggregates
  - record_representation_clauses
  - references_to_local_and_non_local_objects
  - selected_components
  - subprogram_calls
    - parameter_associations
    - procedure_call
  - tasking
    - entries_entry_calls_and_accept_statements
    - task_and_entry_attributes
    - task_execution_task_activation
    - task_types_and_task_objects
- system_capacity
  - boolean
    - false
    - true
- iteration_scheme
  - clock_resolution_based_loop
  - fixed_loop_within_test
  - redundancy_without_loop
  - user_supplied_number_of_iterations
- loop_type

```
      for_loop
      simple_exit_loop
      while_loop
    parameter_mode
      in_out_parameter_mode
      in_parameter_mode
      out_parameter_mode
  number
    pos_int
  software_component_description
  text
```

APPENDIX C: AdaKNET Query Language Syntax and Summary

This appendix contains a description of the AdaKNET query language. After an overview of the BNF variant used to describe the query language, individual language features are presented syntactically, with the syntactic description followed by a short summary of the semantics of the feature. Following the description of the individual features, the appendix closes with a complete syntactic summary of the query language.

C.1. Extended BNF (EBNF) Meta-Symbols

The syntax of the language is described using an extended BNF. The notation used is the same as the notation used throughout the Ada LRM. A brief description is given below. For a complete description see section 1.5 of the LRM.

`lower_case_word`
: nonterminal (e.g. `statement`).

italicized_part`_lower_case_word`
: refers to same nonterminal as the lower case word without italicized part. The italicized part is used to convey some semantic information (e.g. *generic_concept*`_identifier`).

`bold_face_word`
: language token (e.g. **`retrieve`**).

`{item}`
: braces enclose item which may be repeated zero or more times.

`[item]`
: brackets enclose optional item.

`item1 | item2`
: alternation; either item1 or item2

C.2. Query Language EBNF and Semantics

Statements

```
statement ::= open_statement | quit_statement | query_statement

open_statement ::= open network_identifier .

quit_statement ::= quit .

query_statement ::= retrieve query_expression .
```

The three types of statements in the AdaKNET query language correspond to the three basic actions permissible. The open statement specifies the AdaKNET network to which subsequent queries will refer. When the query language processor is invoked from the Librarian application, the network being browsed by the Librarian is used as the default network to be queried. The query statement allows the user to formulate queries on the current network. The query session is terminated with the quit statement.

Query Expressions

```
query_expression ::=
   query
   | query_expression union query_expression
   | query_expression intersect query_expression
   | ( query_expression )

query ::= generic_concept_identifier [facet_expression]
```

The basic query consists of a generic concept name which identifies the general class of individuals to be returned in response to the query. This can be followed by a facet expression to provide further requirements on the roles of the individuals. Such basic queries can be combined with the **union** and **intersect** operations. Parentheses may be used to specify the desired order of operation evaluation; the default precedence is left to right.

Facet Expressions

```
facet_expression ::=
   facet
   | facet_expression , or facet_expression
   | facet_expression , and facet_expression
   | ( facet_expression )

facet ::= with role_identifier [having constraint_expression]
```

Facet expressions further narrow queries by specifying roles which must be present on an individual concept for it to match the query, and by specifying constraints on the properties of those roles. The **and** and **or** operations can be used to combine facet expressions into compound expressions. The default precedence of operations is left to right; different evaluation order can be specified with parentheses.

Constraint Expressions

```
constraint_expression ::=
   constraint
   | constraint_expression or constraint_expression
   | constraint_expression and constraint_expression
   | ( constraint_expression )

constraint ::= range_min_constraint | range_max_constraint
   | type_constraint | filler_constraint

range_min_constraint ::=
   min > number
   | min >= number
   | min = number
```

```
range_max_constraint ::=
   max < number_or_infinity
   | max <= number_or_infinity
   | max = number_or_infinity

number_or_infinity ::= number | infinity

type_constraint ::= type generic_concept_identifier

filler_constraint ::= filler individual_concept_identifier
```

Constraints are used to further restrict or specify the properties of roles required by the query. The range is restricted by the range min and max constraints and the type is restricted by the type constraint. The filler constraint specifies individual concepts that must fill the role. Such basic constraints can be combined with the **and** and **or** operations. Parentheses may be used to specify the desired order of operation evaluation; the default precedence is left to right.

Lexical Elements

```
identifier ::= unquoted_id | quoted_id

unquoted_id ::= letter {[underline] letter_or_digit}

quoted_id ::= " unquoted_id "

letter ::= upper_case_letter | lower_case_letter

number ::= digit {digit}
```

C.3. Query Language EBNF Syntax Summary

The following is a summary of the EBNF description of the query language syntax.

```
statement ::= open_statement | quit_statement | query_statement

open_statement ::= open network_identifier .

quit_statement ::= quit .

query_statement ::= retrieve query_expression .

query_expression ::=
   query
   | query_expression union query_expression
   | query_expression intersect query_expression
   | ( query_expression )

query ::= generic_concept_identifier [facet_expression]
```

```
facet_expression ::=
   facet
   | facet_expression , or facet_expression
   | facet_expression , and facet_expression
   | ( facet_expression )

facet ::= with role_identifier [having constraint_expression]

constraint_expression ::=
   constraint
   | constraint_expression or constraint_expression
   | constraint_expression and constraint_expression
   | ( constraint_expression )

constraint ::= range_min_constraint | range_max_constraint
   | type_constraint | filler_constraint

range_min_constraint ::=
   min > number
   | min >= number
   | min = number

range_max_constraint ::=
   max < number_or_infinity
   | max <= number_or_infinity
   | max = number_or_infinity

number_or_infinity ::= number | infinity

type_constraint ::= type generic_concept_identifier

filler_constraint ::= filler individual_concept_identifier

identifier ::= unquoted_id | quoted_id

unquoted_id ::= letter {[underline] letter_or_digit}

quoted_id ::= " unquoted_id "

letter ::= upper_case_letter | lower_case_letter

number ::= digit {digit}
```

APPENDIX D: HKBDL Syntax and Summary

This appendix contains a description of the Hybrid Knowledge Base Description Language (HKBDL). After an overview of the BNF variant used to describe HKBDL, individual language features are presented syntactically, with the syntactic description followed by a short summary of the semantics of the feature. Following the description of the individual features, the appendix closes with a complete syntactic summary of HKBDL.

D.1. Extended BNF (EBNF) Meta-Symbols

The syntax of the language is described using an extended BNF. The notation used is the same as the notation used throughout the Ada LRM. A brief description is given below. For a complete description see section 1.5 of the LRM.

`lower_case_word`
: nonterminal (e.g. `hybrid_spec`).

italicized_part`_lower_case_word`
: refers to same nonterminal as the lower case word without italicized part. The italicized part is used to convey some semantic information (e.g. *generic_concept*`_identifier`).

`bold_face_word`
: language token (e.g. **`end`**).

`{item}`
: braces enclose item which may be repeated zero or more times.

`[item]`
: brackets enclose optional item.

`item1 | item2`
: alternation; either item1 or item2

D.2. HKBDL EBNF and Semantics

Hybrid Knowledge Base Specification

```
hybrid_spec ::=
   hybrid hybrid_kb_identifier ( network_identifier ) is
      state_mapping
      {state_mapping}
   end hybrid_kb_identifier ;
```

The identifier following the **`hybrid`** keyword indicates the name of the hybrid knowledge base. This identifier must match the one at the **`end`** keyword. The parenthesized identifier specifies the AdaKNET network upon which the hybrid knowledge base is built.

State Mappings

```
state_mapping ::=
   inference base inferencer_base_identifier
      at generic_concept_identifier ;
   | inference base inference_base_identifier ;
   | integer value number at individual_concept_identifier ;
   | text value string at individual_concept_identifier ;
   | text file file_name_string at individual_concept_identifier ;
```

State mappings describe the association of objects, such as AdaTAU inference bases, integers, and text, with concepts in the underlying AdaKNET network. Inference bases are associated with generic concepts; if a concept name is not provided, it is assumed that the concept name is the same as the inference base name. Integers and text are associated with individual concepts; both the integer or text value must be specified. The text value may be supplied directly, or a file name may be given. If a file name is given, the file becomes associated with the concept and the contents of the file become the text value of the concept. This method is more convenient for associating extended text with a concept.

Lexical Elements

```
identifier ::= letter {[underline] letter_or_digit}

letter ::= upper_case_letter | lower_case_letter

number ::= digit {digit}

string ::= "{graphic_character}"
```

D.3. HKBDL EBNF Syntax Summary

The following is a summary of the EBNF description of the HKBDL syntax.

```
hybrid_spec ::=
   hybrid hybrid_kb_identifier ( network_identifier ) is
      state_mapping
      {state_mapping}
   end hybrid_kb_identifier ;

state_mapping ::=
   inference base inferencer_base_identifier
      at generic_concept_identifier ;
   | inference base inference_base_identifier ;
   | integer value number at individual_concept_identifier ;
   | text value string at individual_concept_identifier ;
   | text file file_name_string at individual_concept_identifier ;

identifier ::= letter {[underline] letter_or_digit}

letter ::= upper_case_letter | lower_case_letter

number ::= digit {digit}
```

```
string ::= "{graphic_character}"
```

References

[McDowell89] R. McDowell and K. Cassell, "The RLF Librarian: A Reusability Librarian Based on Cooperating Knowledge-Based Systems," *Proceedings of RADC 4th Annual Knowledge-Based Software Assistant Conference*, Utica, NY, September 1989.

[Simos88] M. Simos, "The Growing of an Organon: A Hybrid Knowledge-Based Technology and Methodology for Software Reuse," *Proceedings of 1988 National Institute for Software Quality and Productivity (NISQP) Conference on Software Reusability*, April 1988, pp. E-1 through E-25.

[Solderitsch89] J. Solderitsch, K. Wallnau, and J. Thalhamer, "Constructing Domain-Specific Ada Reuse Libraries," *Proceedings of Seventh Annual National Conference on Ada Technology*, March 1989.

[Wallnau88] K. Wallnau, J. Solderitsch, M. Simos, R. McDowell, K. Cassell, and D. Campbell, "Construction of Knowledge-Based Components and Applications in Ada," *Proceedings of AIDA-88, Fourth Annual Conference on Artificial Intelligence & Ada*, November 1988, pp. 3-1 through 3-21.