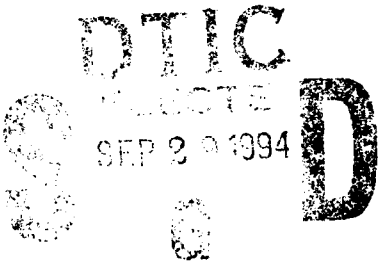
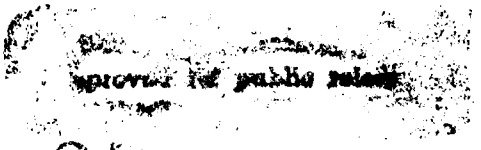
TASK: UU03
CDRL: 05156
February 1993

Reuse Library Framework Administrator Tutorial

Informal Technical Data



STARS-UC-05156/019/00
February 1993



TASK: U03
CDRL: 05156
February 1993

INFORMAL TECHNICAL REPORT

For The

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

RLF Administrator Tutorial

STARS-UC-05156/019/00
February 1993

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0011

Prepared for:

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

Prepared by:

Paramax Systems Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

TASK: U03
CDRL: 05156
February 1993

Data ID: STARS-UC-05156/019/00

Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.

Developed by: Paramax Systems Corporation

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA) under contract F19628-88-D-0031, the STARS program is supported by the military services, SEI, and MITRE, with the U.S. Air Force as the executive contracting agent.

TASK: U03
CDRL: 05156
February 1993

INFORMAL TECHNICAL REPORT
RLF Administrator Tutorial

Principal Author(s):

Jim Solderitsch *Date*

Approvals:

Task Manager *Richard E. Creps* *Date*

(Signatures on File)

TASK: U03
CDRL: 05156
February 1993

INFORMAL TECHNICAL REPORT
RLF Administrator Tutorial

Change Record:

Data ID	*Description of Change*	*Date*	*Approval*
STARS-UC-05156/019/00	Reissued: Minor changes to accompany RLF v4.1 release	February 1993	*on file*
STARS-UC-05156/009/00	Original Issue	November 1992	*on file*

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Informal Technical Report

4. TITLE AND SUBTITLE

RLF Administrator Tutorial

5. FUNDING NUMBERS

F19628-88-D-0031

6. AUTHOR(S)

Paramax Corporation

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Paramax Corporation
1210 Sunrise Valley Drive
Reston, VA 22090

8. PERFORMING ORGANIZATION REPORT NUMBER

STARS-UC-05156/019/00

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Air Force
Headquarter, Electronic Systems
Hanscom AFB, MA 01731-5000

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

05156

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution "A"

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This is an initial version of an RLF administration training package. This package complements training packages that cover RLF modeling and general RLF usage. The purpose of this package is to provide an orientation for new RLF administrators – those who are responsible for setting up and maintaining an RLF library. This package assumes that its audience is already familiar with the RLF from an end-users perspective, but it does not assume that its audience is familiar with RLF modeling concepts. In particular, having taken the RLF Modeler's tutorial is not required.

14. SUBJECT TERMS

15. NUMBER OF PAGES
40

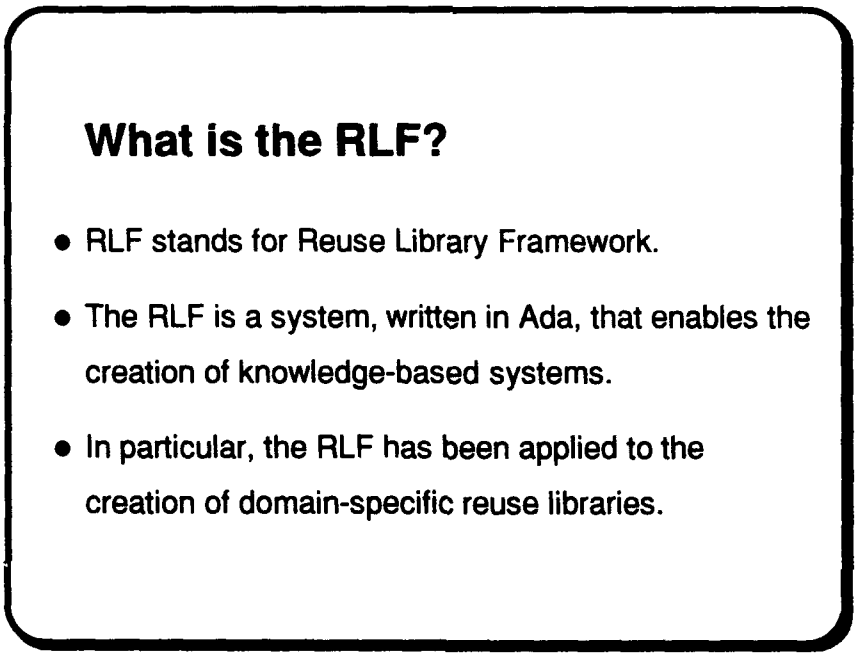
16. PRICE CODE

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

1 Introduction

The RLF is a knowledge-based system, developed by Paramax with support by the STARS program, whose primary application has been the design, implementation, and deployment of domain-specific reuse library systems. A reuse library supports software engineers by enabling them to quickly locate software assets (e.g. requirements, designs, code modules, test plans etc.) that can be of use in their construction of a software system.

Slide 1



A domain is an application area, typically the one of immediate relevance to the software engineer, and a domain model is a *machine representation* of information about the application-area and the library assets available for the application area. The model can contain general domain information along with data about the form, fit, and function of the available library assets.

This is an initial version of an RLF administration training package. This package complements training packages that cover RLF modeling and general RLF usage. The purpose of this package is to provide an orientation for new RLF administrators – those who are responsible for setting up and maintaining an RLF library. This package assumes that its audience is already familiar with the RLF from an end-users perspective, but it does not assume that its audience is familiar with RLF modeling concepts. In particular, having taken the RLF Modeler's tutorial is not required.

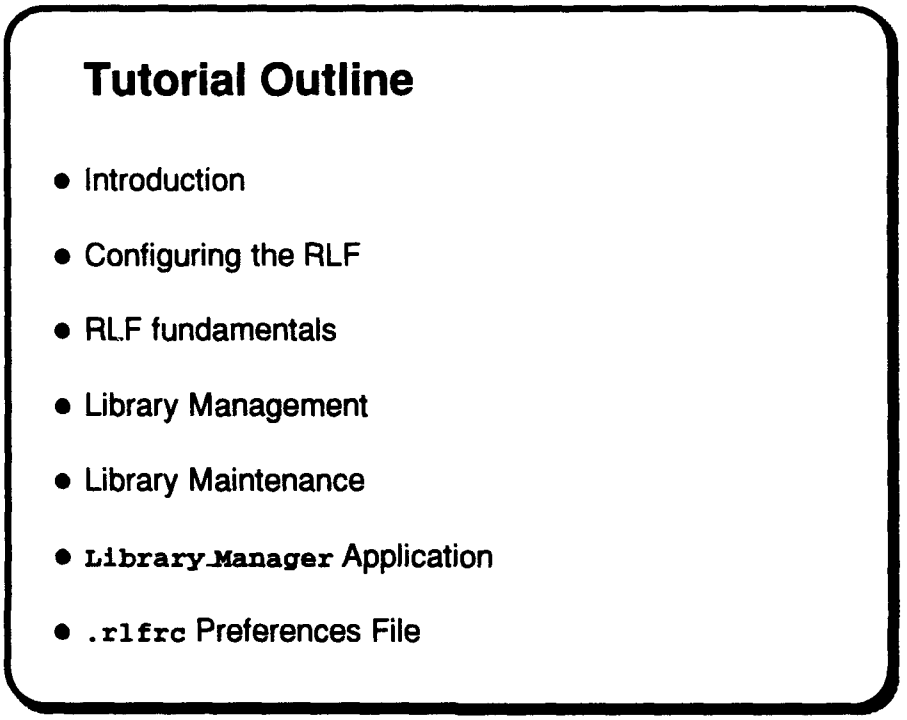
However, the audience is expected to have a general understanding of the UNIX operating system and the X Window System. If X Window System experience is lacking, the prospective RLF administrator should identify a local source of expertise in this area.

This orientation follows the organization of the RLF Adminstrator's manual which is part

of the RLF v4.1 release. However, the tutorial does not fully cover all of the topics in the manual. Nor does it cover RLF installation – these topics are carefully presented in the **RLF Source Code Release Installation Guide** and the **RLF Binary Release Installation Guide**.

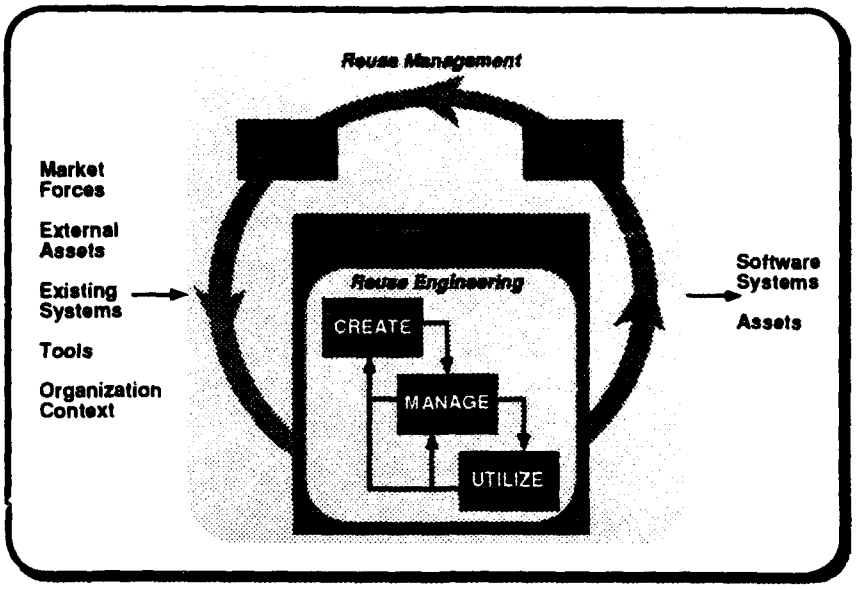
The orientation follows the following general outline.

Slide 2



The material in this orientation supports a portion of a domain-specific, reuse-based, process-driven approach to system engineering that is the cornerstone of the STARS program. This approach is embodied in the STARS Conceptual Framework for Reuse Processes (CFRP) shown on SLIDE 3.

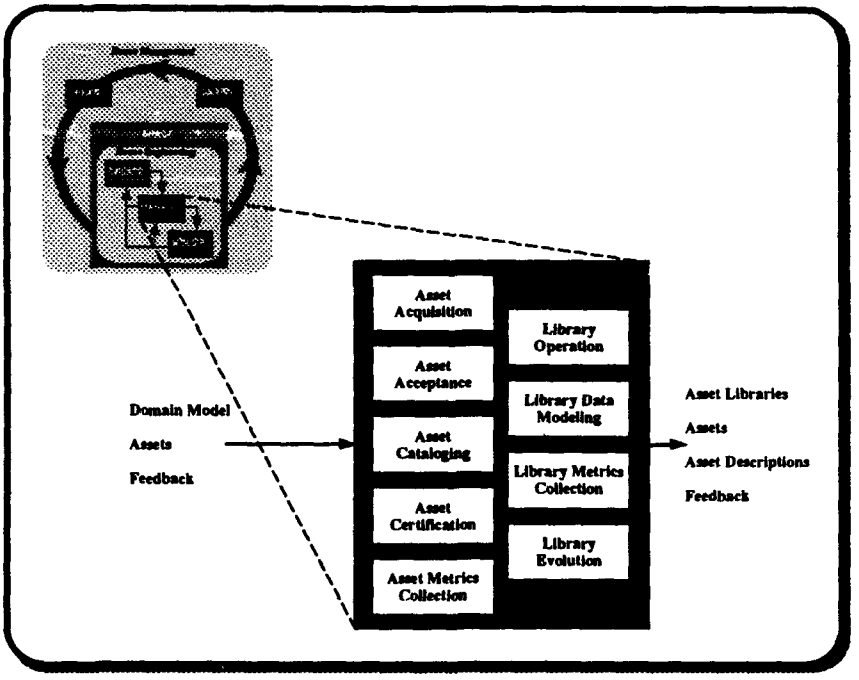
Slide 3



In particular, the RLF supports the creation of a reuse support structure and the use and management of assets stored within this system. An RLF library manager is most likely concerned with the management of assets that are modeled and stored within a library structure along with managing the library itself.

SLIDE 4 breaks out the Manage activity on the CFRP. A well-developed and presented RLF model helps the manager accomplish all of the tasks shown on the slide.

Slide 4

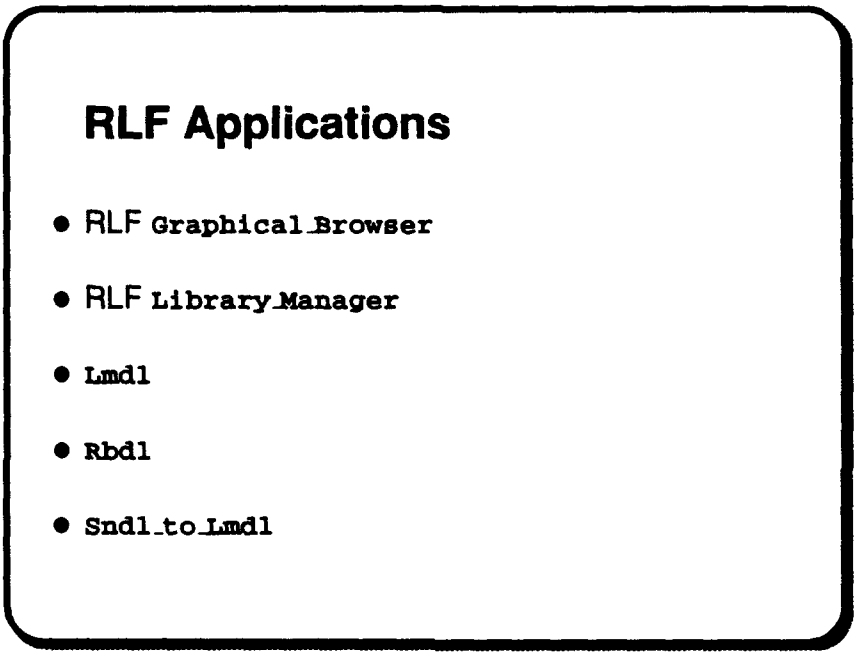


2 RLF Configuration

The RLF installation guides are important resources for the library administrator and should be consulted to learn the mechanics of installing and configuring an RLF library system. The details presented in the guides are not repeated here and because of the largely automated installation process, the administrator should experience little difficulty in creating an RLF installation for the first time or updating a prior installation with a new release.

A successful installation should provide the RLF user with five distinct applications along with a set of RLF sample libraries. SLIDE 5 lists the basic applications.

Slide 5



The installation guide tells how to verify if these applications have been installed correctly. The rest of this orientation assumes that installation has been successfully completed. The administrator should test the installation of the applications by executing them using the sample model and library data provided in the installation.

General RLF usage and administration can be simplified by creating an RLF execution environment through the use of RLF-specific UNIX environment variables. The role of these variables is summarized on the next slide.

Slide 6

RLF Environment Variables

RLF_LIBRARIES -- UNIX path where RLF library data is located

RLF_WORKING_DIR -- UNIX path which RLF should use for user and OS actions (e.g. asset extraction)

RLF_EDITOR -- the name of a UNIX program that is invoked when RLF text-editing actions are invoked

RLF_PAGER -- the name of a UNIX program that is invoked when textual library assets are inspected

XAPPLRESDIR -- UNIX path where X resources are located that determine the layout of the **Graphical_Browser**

The RLF applications make selective use of these variables, with the `Graphical_Browser` affected by all five of them and the `Library_Manager` affected by all except `XAPPLRESDIR`. `LMDL` and `RBDL` are only affected by the `RLF_LIBRARIES` variable. The administrator can make RLF usage more convenient by setting these variables for the user in a startup script. For example, assuming that the RLF has been installed at the top of a UNIX file system named /libraries, the next slide defines an appropriate start-up script.

Slide 7

Start-up Script for Graphical_Browser

```
#! /bin/csh -f

setenv RLF_LIBRARIES /libraries/instances
setenv XAPPLRESDIR /libraries/rlf/bin
setenv RLF_WORKING_DIR /home/johndoe
Graphical_Browser
```

In addition, a library user can personally tailor her usage of the RLF through the use of the

`.rlfrc` file. The RLF administrator should become familiar with the syntax and features of the `.rlfrc` file so that several RLF user profiles can be made available to serve the local needs of new RLF users. The `.rlfrc` file is presented in some detail later in this tutorial.

Along with configuring the RLF executables, the administrator needs to establish access rights at various levels. For the executables themselves, the UNIX file permissions and executable placement within the local file system are used to determine accessibility. Except for the `Graphical_Browser`, access to the RLF executables should be restricted to an explicitly determined number of users, typically the RLF administrator and designated RLF model developers.

Other finer-grained access restrictions to RLF objects and actions that can be applied to objects are handled by the RLF action mechanism. In particular, actions can be declared as "privileged" and therefore only available through the `Library_Manager` application (such as editing the contents of an asset) which is itself controlled through UNIX access controls. If access to RLF library objects are to be restricted, suitable actions can be installed in the RLF library action sub-model to provide the necessary restrictions. These capabilities are discussed further later in this orientation. The next two slides summarize access control issues.

Slide 8

RLF Access Control Issues

- Access to RLF Applications

  Only the `Graphical_Browser` should be freely executable

- Access to RLF Actions

  Certain RLF actions which are declared *privileged* are available only from `Library_Manager` Application

- Access to RLF Asset Objects

  Access controls on library objects are provided through appropriate actions within RLF library model

Since library objects and UNIX executables used in RLF actions are themselves UNIX file entities, the proper UNIX permissions must be set for the RLF user to access them. Failure to do so can result in anomalous behavior as experienced by the RLF user. For this reason, the administrator should take care in checking these aspects of an RLF model installation. Another potential troublespot can occur if the UNIX tools that are invoked as part of RLF actions are not available to the user (usually because the user's UNIX path does not include the locations where these tools reside). The action call will report a failure message – these

messages can be disconcerting to new users and should generally be avoided.

3 RLF fundamentals

This administrator's orientation does not go into detail regarding RLF domain modeling capabilities. These are covered in the Modeler's orientation and in the RLF Modeler's manual. However, the administrator should have a reading knowledge of RLF model specifications and be able to edit them to account for user-reported problems and model update requests from RLF model developers. Some of the basic slides in the RLF User Tutorial are reused here to provide some higher level context as needed.

An RLF library model is a knowledge network describing an application domain. The first step in creating such a model is to identify the area common to all the assets to be available in the domain-specific reuse library. A domain-specific approach is summarized on SLIDE 9.

Slide 9

Domain Model approach

- Domain is an application area
- Domain Analysis produces knowledge about Domain
- Encoding knowledge leads to Domain Model
- RLF knowledge-base capabilities used to represent model
- Knowledge-Based approach leads to improved user effectiveness

RLF model specifications are made using the Library Model Definition Language (LMDL). A summary of LMDL features and syntax is given in the next few slides.

Slide 10

Basic Lmdl Components

- Categories
- Objects
- Relationships
- Attributes
- Actions

SLIDE 11 summarizes the important features of model categories.

Slide 11

Categories

- general descriptions of a kind of thing
- classify what a thing is
- arranged in an inheritance hierarchy called a specialization hierarchy
- relationships inherited along specialization hierarchy
- most general category occurs at root of the model

An LMDL category definition defines a domain model class which stands for a concept or kind of thing that is of potential interest to a user of the model. For example, SLIDE 12 shows some high level category declarations for a model of the domain of sorting.

Slide 12

Lmdl Category Declarations

```
root category Thing is
end root category;

category Algorithms (Thing) is
end category;

category "Search Algorithms" (Algorithms) is
end category
```

SLIDE 12 shows that categories are declared hierarchically where each category names its parents – note that the RLF supports a network model structure since a category can be declared to have multiple parents (not shown in the simple example). Every model must declare a root category which is an implicit ancestor of all classes in the model.

Library elements are called objects and must be defined in terms of a parent category. Objects are all members of categories and are distinguished by the relationships they have as a result of their location in the specialization hierarchy. SLIDE 13 summarizes some key properties of objects.

Slide 13

Objects

- actual things instead of classifications of things
- associated with the most appropriate category (or categories) which describe them
- reusable assets are objects which are identified through attributes as well as category relationships which are "instantiated" for the asset
- objects are said to "individuate" the containing category

The next slide shows some simple LMDL object definitions.

Slide 14

Lmdl Object Declarations

```
object Ada ("source Language") is
end object

object "Example Quicksort" (Quicksort) is
end object
```

Categories and objects are distinguished primarily through the relationships and attributes which they possess. SLIDE 15 describes some basic properties of relationships in the RLF.

Slide 15

Relationships

- describe features of categories and objects
- express associations between different categories or objects
- are inherited by categories or objects below that category in the hierarchy at which they are defined
- have a name, an owner, a type and a cardinality
- can be restricted by type or by cardinality when inherited (but never generalized)
- object relationships "fill" corresponding relationships defined between corresponding categories

The most important aspect of RLF models is that categories can be declared to have relationships (to other categories) and that these relationships are then inherited by subcategories of a category. In turn, objects which are members of a class can then have particular instances of these relationships attached to them. Relationships are defined in terms of the "owning" class, the relationship "type" which is the name of another model class, and a "cardinality" which declares how many repetitions of the relationship a particular object may have. The LMDL specification given on the next slide provides an example of the declaration of a set of category relationships.

Slide 16

Lmdl Relationships

```
category Algorithms ( Thing ) is
  relationships
     is_written_in (0 .. infinity)
           of "Source Language";
     works_on (0 .. infinity)
           of "Data Structure";
     has_best_case_of (0 .. 1) of Performance;
     has_avg_case_of (0 .. 1) of Performance;
     has_worst_case_of (0 .. 1) of Performance;
     has_size_of (0 .. 1) of "Lines of Code";
  end relationships;
end category;

category "quick sort" ( exchange_sorts ) is
  restricted relationships
     has_best_case_of (1 .. 1) of Logarithmic;
     has_avg_case_of (1 .. 1) of Logarithmic;
     has_worst_case_of (1 .. 1) of Quadratic;
  end restricted;
end category;
```

As relationships are inherited, they can be restricted by type and cardinality at a particular subcategory. A simple example of this kind of restriction is shown in the previous slide. These restrictions are also permitted for objects and, in addition relationships can be instantiated (sometimes called "filled") which means a link is established between an object in the owning category with a corresponding one in the type category. The destination object is said to "fill" or satisfy the relationship that was originally defined between the two classes. For example, the next slide shows some role restrictions and subsequent instantiation for particular objects.

Slide 17

Lmdl Relationships (cont.)

```
object example_quicksort ( "quick sort" ) is
  restricted relationships
     is_written_in (1 .. 1)
          of "Source Language";
     works_on (1 .. 1) of "Data Structure";
     has_worst_case_of (1 .. 1) of Quadratic;
     has_size_of (0 .. 1) of Number;
  end restricted;
  fillers
     Ada satisfies is_written_in;
     Array satisfies works_on;
     "N^2" satisfies has_worst_case_of;
     "Twenty-Four" satisfies has_size_of;
  end fillers;
end object;
```

In addition to typed and inherited relationships of the kind described above, categories and objects can be equipped with non-inherited attributes which are used to further characterize them. RLF attributes are used to annotate categories and objects with static information that provides additional data about them. These attributes are often used in combination with RLF actions which process the attributes and present them to the user. In particular, attributes are the means by which actual file contents are attached to reusable assets which are modeled as objects in an RLF model. SLIDE 18 illustrates some features of attributes.

Slide 18

Attributes

- can be integers, strings of characters, or files
- have names so they can be referenced and used by RLF *actions*
- file attributes can be viewed, extracted, or otherwise manipulated
- attributes are not inherited -- they must defined at each category or object where they are useful

The RLF currently supports string, integer or file attributes. The next slide includes some simple LMDL attribute declarations.

Slide 19

Lmdl Attributes

```
category Extract ( Action ) is
  attributes
     string is "Extract Asset";
  end attributes;
end category;

category Quicksort ( "Exchange Sorts" ) is
  attributes
     file desc_source is
        "sort_and_search/exchange_sort_desc";
  end attributes;
end category;

object "Example Quicksort" ( Quicksort ) is
  attributes
     file desc_source is
        "sort_and_search/exchange_sort_desc";
     file source is
        "sort_and_search/quick_sort_.a";
     string size_of is "24";
  end attributes;
end object;
```

Actions let the user process categories and objects and permit access to system and library resources within a context appropriate to specific categories and objects. The context for these actions is provided by RLF attributes. Actions are summarized in SLIDE 20.

Slide 20

Actions

- provide library users with appropriate system and library services
- basis for asset viewing and extraction
- defined within model definition language -- sample Action sub-model provided in RLF distribution
- implemented as system calls, or internal Ada procedures
- inherited like relationships
- have names, an action category, a list of action targets, and a list of action agents
- targets reference attributes which provide input for the action
- agents reference attributes which can affect how the action is performed
- can be privileged which means they are unavailable through the RLF GB (restricted to administrative use)
- can be restricted at subcategories or lower level objects

The RLF administrator should become familiar with the basic action model hierarchy that is included with the sample RLF libraries included in the RLF v4.1 release. A LMDL specification for a subset of this model is shown on the next slide. Modifying this model to add, delete or modify actions is covered later in this tutorial.

Slide 21

Action Model Subset

```
category "Action Definition" (thing) is
end category;

category "Action Type" ("Action Definition") is
end category;

  category "System String" ("Action Type") is
  end category;

  category "Ada Procedure" ("Action Type") is
  end category;

  category "Message Pass" ("Action Type") is
  end category;

category Action ("Action Definition") is
  relationships
    has_action_type (1 .. 1) of "Action Type";
  end relationships;
end category;

category View (Action) is
  restricted relationships
    has_action_type of "System String";
  end restricted;
  attributes
    string is "xterm -e $RLF_PAGER ## &";
  end attributes;
end category;
```

Library actions are connected to the content of a library model through LMDL action and attribute specifications at the appropriate model catagories and objects. A model fragment showing some of these specifications is shown on SLIDE 22.

Slide 22

Local Action Specs

```
category "Insertion Sorts" ( "Internal Sorts" ) is
  attributes
     file desc_source is "sort_and_search/insertion_sort_desc";
  end attributes;
  actions
     "Read Description" is "Display Description" on desc_source;
  end actions;
end category;

object "Example Quicksort" ( Quicksort ) is
  attributes
     file desc_source is "sort_and_search/exchange_sort_desc";
     file source is "sort_and_search/quick_sort_.a";
     string size_of is "24";
  end attributes;
  actions
     "View Code Size" is "Display Integer" on size_of;
     "View Source" is View on source;
     "Extract Source" is Extract;
  end actions;
end object;
```

Finally, RLF models can make use of a special, built-in action called inferencing. This capability has been used primarily to direct a user browsing an RLF model and to provide that user with advice about the model in a context-dependent, goal-oriented fashion. The user's responses to inferencing-induced questions lead the user to categories and objects within the model that are expected to meet user needs as these are elicited during the inferencing process. SLIDE 23 describes some inferencing features.

Slide 23

Library Inferencers

- defined in special-purpose RLF language
- attached only to modeler-specified categories or objects
- ask the user questions about the user's needs and intentions
- based on responses, make deductions about model locations and contents that are of interest to the user
- can focus user to specific category or object
- are distributed over a model and can invoke and exchange information with each other

SLIDE 24 gives a LMDL example showing how inferencers which are declared in separate

Rule Base Description Language (RBDL) specifications are attached to network nodes and are thus made accessible to the library end-user.

Slide 24

Attaching RBDL Inferencer

```
category Quicksort ( "Exchange Sorts" ) is
  restricted relationships
     has_best_case_of (1 .. 1) of Logarithmic;
     has_avg_case_of (1 .. 1) of Logarithmic;
     has_worst_case_of (1 .. 1) of Quadratic;
  end restricted;
  attributes
     file desc_source is "sort_and_search/exchange_sort_desc";
  end attributes;
end category;

attach inferencer quicksort to Quicksort;
```

4 Library Management

An outline of the topics presented in this section is given in SLIDE 25. Examples in this section assume that the **RLF_LIBRARIES** environment variable has been set to the directory containing the reuse library being modified and that the LMDL translator, **Lmdl**, appears in the library administrator's path.

Slide 25

Management Topics

- convert v3.0 libraries
- build new libraries
- remove libraries
- modify library elements -- change
  - assets
  - actions
  - advice

LMDL is significantly different from its specification language predecessors. RLF versions prior to v4.0 included the Semantic Network Description Language (SNDL) which has now been superceded by LMDL. Since LMDL is not an upwards compatible extension of SNDL, a translator tool is provided to convert old network specifications. This tool is called `Sndl_to_Lmdl`. Its features are summarized on the next slide.

Slide 26

Translator Features

- Syntax:

  `Sndl_to_Lmdl [-help] [<filename>] [-pc] [-indent <num>] [-o <name>] [-l <name>]`

- **Command line arguments:**

  `-help` – get list of command line arguments

  `<filename>` – name of file to translate

  `-pc` – preserve case of model element names

  `-indent <num>` – indent major model constructs by `<num>` spaces (default is 3)

  `-o <name>` – write output to named file; default is standard out

  `-l <name>` – write execution log to named file

The LMDL language processor provides the basic mechanism for creating and maintaining

libraries. Once a library has been created, it can only be deleted through the `Library_Manager` application. SLIDE 27 summarizes some basic library maintenance techniques.

Slide 27

Management Techniques

- **`Lmdl spec.lmdl`** builds new library
  - -- Replaces any existing identically named library
  - -- Re-**`Lmdl`**-ing a library is the usual technique to modify a library
- **`Library_Manager`** used to delete an existing library
- **`RLF_LIBRARIES`** determines the location for libraries being processed by **`Lmdl`**
- **`Lmdl -state spec.lmdl`** will only process attribute and inferencer specs for model

  significantly reduces LMDL processing time

For incremental processing of entire model segments, it is often convenient to package the segments separately and to use the incremental specification feature of LMDL to refer to the basic parent model. The syntax and an example of incremental network specification is given on the next two slides.

Slide 28

Top Level LMDL Syntax

```
library_model_spec ::=
    library model model_name is
      [incremental_indication]
      [root_category]
      {category_or_object_or_inferencer}
    end model_name ;
```

Incremental LMDL Clause

```
incremental_indication =
    'extend' 'library' 'model' model_name ';'
```

In addition to this syntax, which specifies the name of the pre-existing library which will be extended, incremental LMDL specifications can not make root category definitions other than to add new definitions to the root category definition of the library being extended.

Slide 29

Incremental LMDL Example

```
library model "Sort, Search, and Math Algorithms" is

  extend library model "Sort and Search Algorithms";

  -- additional category and object definitions
  -- and inferencers attachments describing math functions
  -- connected to categories and objects in the
  -- sort and search algorithms library model

end "Sort, Search, and Math Algorithms";
```

Translating this specification would create a new library named "Sort, Search, and Math

Algorithms" containing all the constructs of the sort and search algorithms library plus any additional entities describing math function algorithms.

The rest of the slides in this section address the modification of model elements which are usually the responsibility of the library manager. These model elements include assets (model objects with "filled" relationships and attribute values), actions and inferencers.

SLIDE 30 discusses some typical asset change possibilities and the following two slides show a category definition and various object definitions which define the asset. An object declaration typically includes an attribute which locates the file containing the asset contents.

Slide 30

Modifying Assets

- Add assets by inserting or modifying LMDL object declarations
- Corresponding LMDL category must already be defined
- Assets are deleted by changing LMDL object spec and Re-**Lmdl**-ing the entire spec
- **Library_Manager** can also delete assets by deleting object attributes

  But, *machine readable network no longer in sync with LMDL spec*
- Contents can be changed by copying a new file to the location stored as part of object attribute definition

  But, old asset version is no longer available to library user

Slide 31

Category Declaration

```
category Quicksort ( "Exchange Sorts" ) is
  restricted relationships
    has_best_case_of (1 .. 1) of Logarithmic;
    has_avg_case_of (1 .. 1) of Logarithmic;
    has_worst_case_of (1 .. 1) of Quadratic;
  end restricted;
  attributes
    file desc_source is
       "sort_and_search/exchange_sort_desc";
  end attributes;
end category;
```

Simple Object Declaration

```
object "Example Quicksort" ( Quicksort ) is
  attributes
     file source is
         "sort_and_search/quick_sort_.a";
  end attributes;
end object;
```

Note that the path location for the `source` attribute is given relative to the `$RLF_LIBRARIES/Text` path.

Slide 32

Full Object Declaration

```
object "Example Quicksort" ( Quicksort ) is
  restricted relationships
     is_written_in (1 .. 1) of "Source Language";
     works_on (1 .. 1) of "Data Structure";
     has_worst_case_of (1 .. 1) of Quadratic;
     has_size_of (0 .. 1) of Number;
  end restricted;
  fillers
     Ada satisfies is_written_in;
     Array satisfies works_on;
     "N^2" satisfies has_worst_case_of;
     "Twenty-Four" satisfies has_size_of;
  end fillers;
  attributes
     file desc_source is
              "sort_and_search/exchange_sort_desc";
     file source is "sort_and_search/quick_sort_.a";
     string size_of is "24";
  end attributes;
  actions
     "View Code Size" is "Display Integer" on size_of;
     "View Source" is View on source;
     "Extract Source" is Extract;
  end actions;
end object;
```

Action management involves maintaining and enhancing the library's built-in action model and allocating action and attribute declarations for categories and objects declared for the library. While the `Library_Manager` provides a limited form of action editing, library managers will likely want to perform most of their management functions directly on the LMDL specification file(s) for the library.

SLIDE 33 presents some action management concepts. A portion of the provided RLF example action model is included on SLIDE 21.

Slide 33

Modifying Actions

- New actions are added within the library action submodel
- **`System String`** actions invoke system services -- action attributes specify actual string with parameters
- **`Ada Procedure`** actions are internal -- the RLF must be modified to add more
- Built-in internal actions are **`Import`**, **`Export`**, **`Extract`**, **`Display_Attributes`**
- Actions can be deleted and modified within the library models LMDL spec
- Actions can be deleted within the **`Library_Manager`**

  But, model spec should be changed appropriately

Note that if actions are removed from the action submodel, references to these actions must be removed wherever they appear in the model. Local action declarations can be removed to make the action unavailable to particular categories or objects.

The next few slides show some examples of action modification and explore some additional action capabilities (such as action agents). These capabilities are treated more fully in the RLF Modeler's and Administrator's Manuals.

Slide 34

New Action Definition

```
category Print ( Action ) is
-- this action category describes a general print action
   restricted relationships
      has_action_type (1 .. 1) of "System String";
   end restricted;
   attributes
      -- ## marks the file to be printed
      -- %%1 marks the UNIX print command to use
      -- %%2 marks any options to the print command
      -- also run the action in the UNIX background
      -- so the RLF application continues
      string print_command is "%%1 %%2 ## &";
   end attributes;
end category;
```

SLIDE 34 defines a new system action to be used to print other asset contents. The strings %%1 and %%2 are placeholders for action agents that provide qualifiers to the system command that ultimately implements the action. Values for these placeholders (and for the action target(s) identified by ##) are obtained locally at the invoking category or object.

The next slide show two simple action references for the Print action. The following slide shows a more complex reference.

Slide 35

Partial Action Reference

```
object "Example Quicksort" ( Quicksort ) is
  actions
     "Print Source" is Print on source
           with print_command, print_options;
  end actions;
end object;
```

Complete Action Reference

```
object "Example Quicksort" ( Quicksort ) is
  attributes
     file source is "sort_and_search/quick_sort_.a";
     string print_command is "lpr";
     string print_options is "-Pprinter1";
  end attributes;
  actions
     "Print Source" is Print on source
          with print_command, print_options;
  end actions;
end object;
```

In the first example, since there are no object attributes to provide values for the action target and agents, the RLF action mechanism will look first in the `Quicksort` category declaration for corresponding attributes to provide values, and then if necessary in `Quicksort` ancestor categories. The second example provides object values for the action targets and agents.

Slide 36

Complex Action Reference

```
object "Example Quicksort" ( Quicksort ) is
  attributes
     file abstract is "sort_and_search/quick_sort_abstract";
     file performance_study is "sort_and_search/quick_sort_perf";
     file source is "sort_and_search/quick_sort_.a";
     string print_command is "lpr";
     string print_options is "-Pprinter1";
  end attributes;
  actions
     "Print All Data" is Print on
        abstract, performance_study, source
        with print_command, print_options;
     "Print Source" is Print on
        source with print_command, print_options;
  end actions;
end object;
```

This would provide a "`Print Source`" action to just print the implementation's source and a "`Print All Data`" action which would print the implementation's abstract, performance study, and source. When the "`Print All Data`" action is invoked, it will iterate over the list of targets performing the action described by action category `Print` for each file in the list.

Actions are modified by editing the library's specification file and modifying the action sub-model and/or by changing the action declarations at particular categories or objects. For example, the action sub-model's `View` action can be changed to one of the variants shown on the next slide – see SLIDE 21 for the original definition for this action.

Slide 37

Alternate View Action

```
category View ( Action ) is
  restricted relationships
     has_action_type of "System String";
  end restricted;
  attributes
     string is "xterm -e /usr/ucb/view ##";
  end attributes;
end category;
```

Another View Action

```
category View ( Action ) is
  restricted relationships
     has_action_type of "System String";
  end restricted;
  attributes
     string is "asset_view.csh ## &";
  end attributes;
end category;
```

For the second example, when the library model had been retranslated using the LMDL translator, `Lmdl`, the view action, when invoked, would execute the *csh* shell script named `asset_view.csh` passing the name of the file to the script as a parameter. This script would also execute in the UNIX background allowing the RLF application to continue.

Note: To save time when modifying a library model's action categories, if the only changes made were changes to the string attributes at action categories, the `Lmdl` translator should be invoked with the `-state` command line option when retranslating.

The last kind of model element that the library administrator may need to maintain are inferencers. The definition and placement of inferencers for use within a complex network model is normally handled by the model developer. The administrator may be called upon to relocate inferencers, replace them with new versions, or to delete them entirely. Again, although deletion of an inferencer can be done within the `Library_Manager` application, it is usually easy to make changes to the LMDL spec directly. Executing `Lmdl -state` will quickly modify the internal network when only inferencers (and attributes) have changed but the specialization and aggregation hierarchies remain stable.

The next slide shows a specification fragment with and without an attached inferencer. Inferencer definition is accomplished by executing `Rbdl`. Note that if no changes to inferencer

placement or naming occur, inferencer updates can be accomplished independently of LMDL. Rbdl and AdaTAU are discussed fully in the RLF Modeler's Manual.

Slide 38

Category With Inferencer

```
category Quicksort ( "Exchange Sorts" ) is
  restricted relationships
     has_best_case_of (1 .. 1) of Logarithmic;
     has_avg_case_of (1 .. 1) of Logarithmic;
     has_worst_case_of (1 .. 1) of Quadratic;
  end restricted;
  attributes
     file desc_source is
          "sort_and_search/exchange_sort_desc";
  end attributes;
end category;

attach inferencer quicksort to Quicksort;
```

Category Without Inferencer

```
category Quicksort ( "Exchange Sorts" ) is
  restricted relationships
  . . .
  end restricted;
  attributes
  . . .
  end attributes;
end category;
```

When only a portion of the model defining library advice has been removed, it will be sufficient and quicker to retranslate the specification using the LMDL translator's `-state` command line option.

5 Library Maintenance

An outline of the topics presented in this section is given in SLIDE 39. The library administrator is responsible for maintaining the performance and day-to-day operations of an RLF reuse library. The administrator is also responsible for improvements to the library whether they come from user feedback or the library modeler or other sources.

Slide 39

Maintenance Topics

- managing assets
- collecting metrics
- coordinating user feedback
- optimizing performance
- maintaining RLF software

The first item on the list is essentially covered in the previous section. The library administrator should maintain some sort of configuration management on the library's assets so that old versions can be recovered, and differences between old and new versions can be revealed. The extent of this activity and its visibility to the user and the reuse library is up to the library administrator and may be affected by the nature of the assets in the library.

The RLF does not come pre-configured to support the collection and reporting of usage metrics. However, the RLF action mechanism can be applied to collect and report library metrics. Some ideas in this direction are suggested on the next slide.

Slide 40

Metrics Ideas

- Modify Extract action to count asset retrievals
- Modify Extract action to add user to list of asset customers
- Convert View action to a shell script that counts number of viewer executions

  record user name of library user executing the action
- Modify RLF start-up scripts to record user info and RLF usage length
- Provide top-level action to report library statistics to administrators and perhaps users

The RLF can automate user feedback through its action mechanism. The library modeler or administrator could include an action which would appear at every category or object which would mail the library administrator a message which the user could enter while still running the `Graphical_Browser` and the question, problem, or suggestion is still fresh. Help actions could also be modeled this way. The action could mail the administrator the file created as the result of spawning a text editor from the `Graphical_Browser`.

The perceived performance of the RLF software can be affected by many factors. The RLF implementation team is constantly working to address raw performance issues. The next slide provides some additional factors to consider. Some of these factors must be addressed by those responsible for the library model.

Slide 41

Performance Factors

- Is the model overly complex?
- Has sufficient model guidance been provided (inferencers)?
- Are user workstations properly equipped (memory and swap space)?
- Have the X resources for the `Graphical_Browser` been properly adjusted (bitmaps and Browser file definitions)?
- Have sufficient user profiles been developed (startup scripts and `.rlfrc` files)?
- Have user suggestions been solicited and honored where possible?

An outline of areas to consider in regard to the maintenance of RLF software is shown on SLIDE 42.

Slide 42

Maintaining RLF Software

- getting software
- doing software updates
- reporting errors
- getting help

Most aspects of software installation and maintenance are covered in the RLF installation guides. In addition there are mailing lists which cover the RLF from different perspectives. These lists are outlined on the next slide. The RLF itself is available via anonymous FTP at the internet address `stars.rosslyn.paramax.com` in the directory `/pub/RLF`.

Slide 43

RLF Mailing Lists

rlf@stars.ballston.paramax.com
Public forum for discussing RLF issues

rlf-request@stars.ballston.paramax.com
Address for submission of RLF registration forms and mailing list maintenance requests

rlf-bugs@stars.ballston.paramax.com
Completed Program Problem Reports and New Feature Requests

6 Library Manager Application

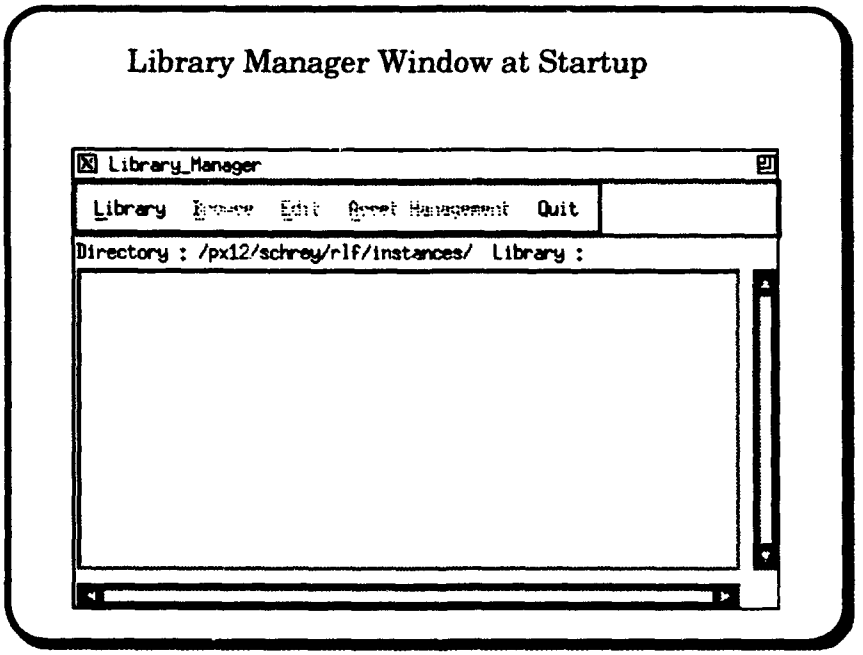
The `Library_Manager` tool performs tasks associated with library administration. These include manipulating entire RLF reuse libraries built from LMDL library model specifications,

browsing and examining the contents of reuse libraries, and performing smaller scale manipulation of library parts. The **Library_Manager** only supports limited library model editing including the deletion of attributes and inferencers from a model. Any editing of this sort done with the **Library_Manager** should be also be done in the LMDL specification of the library model in order to maintain integrity between the specification and the model, although the using the **Library_Manager** allows the changes to be made without rebuilding the entire model at that time.

The slides in this section survey the use and features of the **Library_Manager** application. Before executing the application, the user should select which RLF library location to process by setting a value in the **.rlfrc** file, setting the **RLF_LIBRARIES** environment variable, or supplying the library path on the command line through the **-I** option.

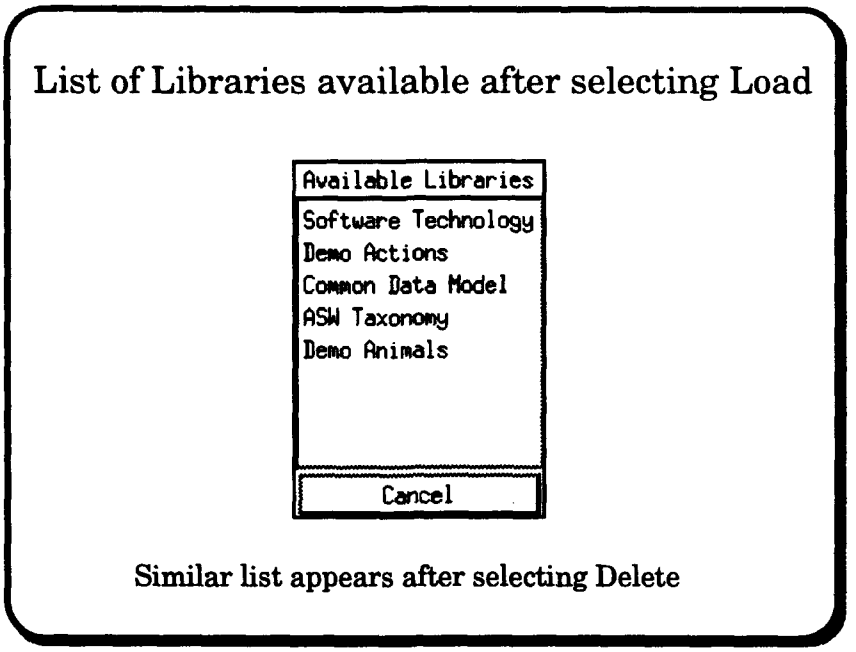
At startup, the **Library_Manager**'s screen appears as in SLIDE 44. Initially, only the **Library** and **Quit** menu items are active. The title bar also lists the current working directory and selected library.

Slide 44



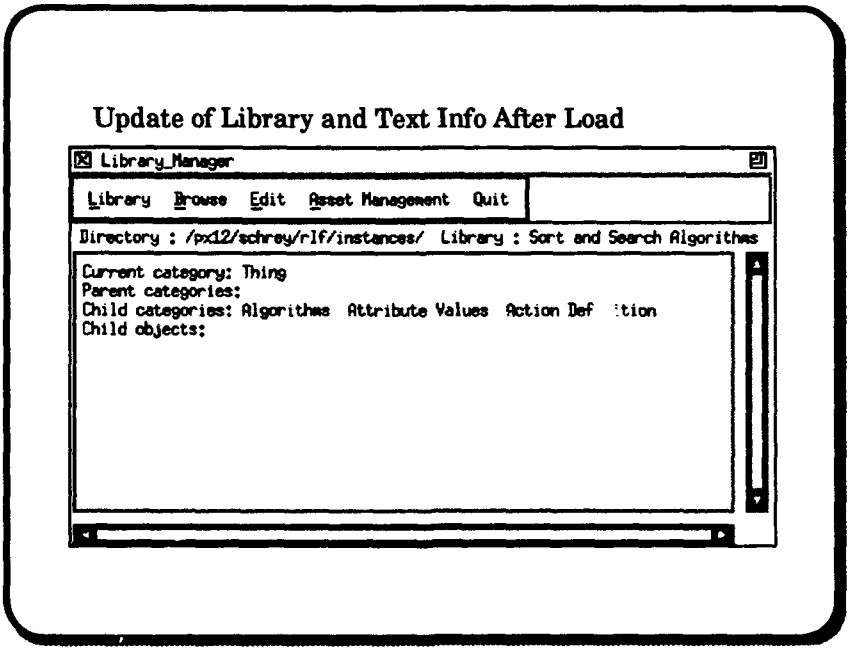
The user selects a library to administer using the **Library** button. The next slide shows that a list of available libraries is presented to the user. A similar list of libraries appears if the user has not yet selected a library and wishes to delete one of the ones available in the chosen Instances directory. A cancel button is available if the user decides not to proceed. The **Close** and **Save** menu items are available after a library has been selected. **Save** makes permanent the library changes made since the library was opened. **Close** closes access to the library but does not save changes.

Slide 45



After a library has been loaded, the screen appears as shown on the next slide. The remaining menu bar choices are now active. Data about the top level concept is displayed in the main window.

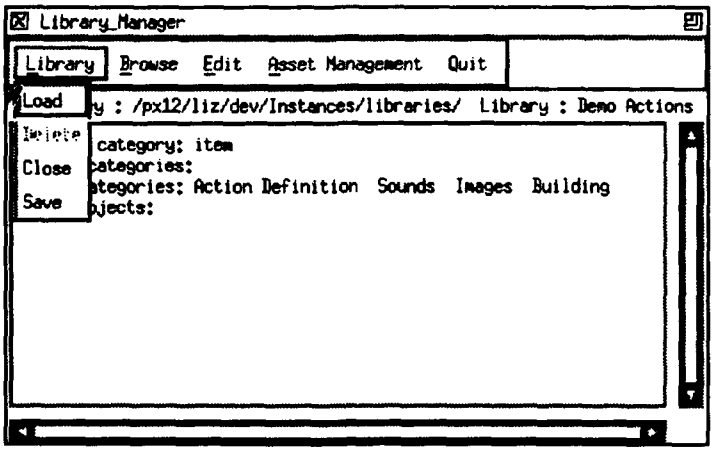
Slide 46



The available menu choices change depending on what choices the user has made while interacting with **Library_Manager**. For example, the next slide shows how the **Delete** button is no longer active after a library has been loaded.

Slide 47

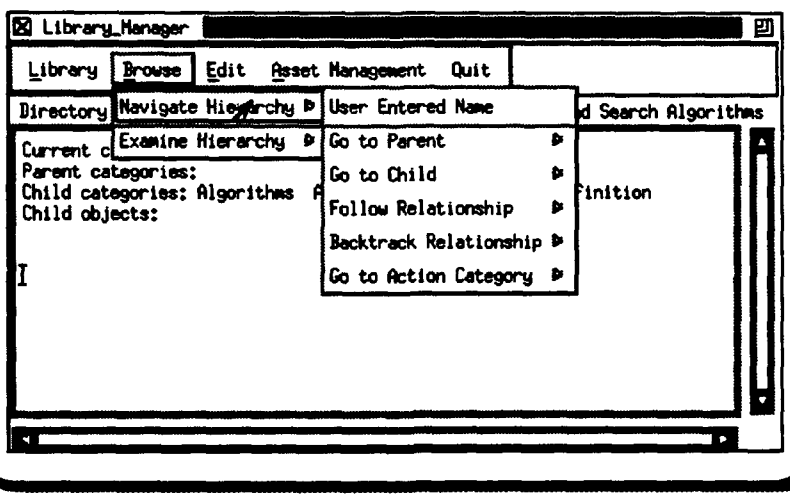
Changes in sensitivity in Library Pulldown after loading a library. Delete is not allowed on an open library.



There are two main choices in the **Browse** menu: **Navigate Hierarchy** and **Examine Hierarchy**. The collection of navigation choices are shown on the next slide. By selecting from this submenu, the user can textually navigate the library. Most of these are self-explanatory. The **Backtrack Relationship** choice produces a submenu of all the relationships which have the current category as a filler type. The owner of each relationship is also shown. Choosing one will make the relationship's owner category the current category. The **Go to Action Category** choice produces a submenu of all the action categories associated with the actions at the current category. Choosing one will make it the current category. The **Library_Manager** is the only RLF tool that permits a user to browse an action submodel.

Slide 48

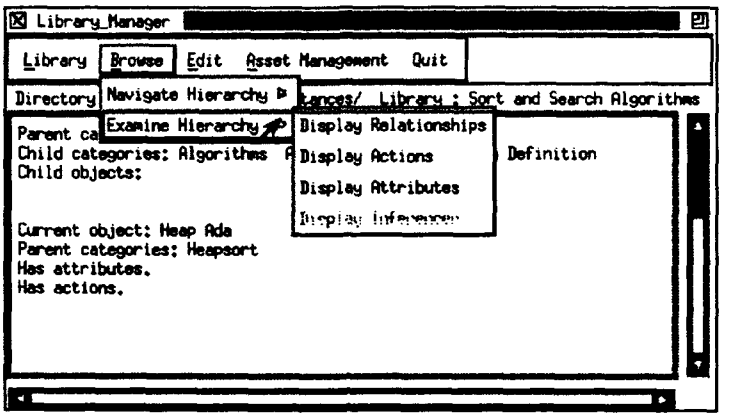
Navigate Hierarchy Pulldown Menu



The next slide shows the submenu choices for the **Examine Hierarchy** menu. These choices do what they say; if there are no actions, attributes or inferencers available at the current node, the submenu is greyed out.

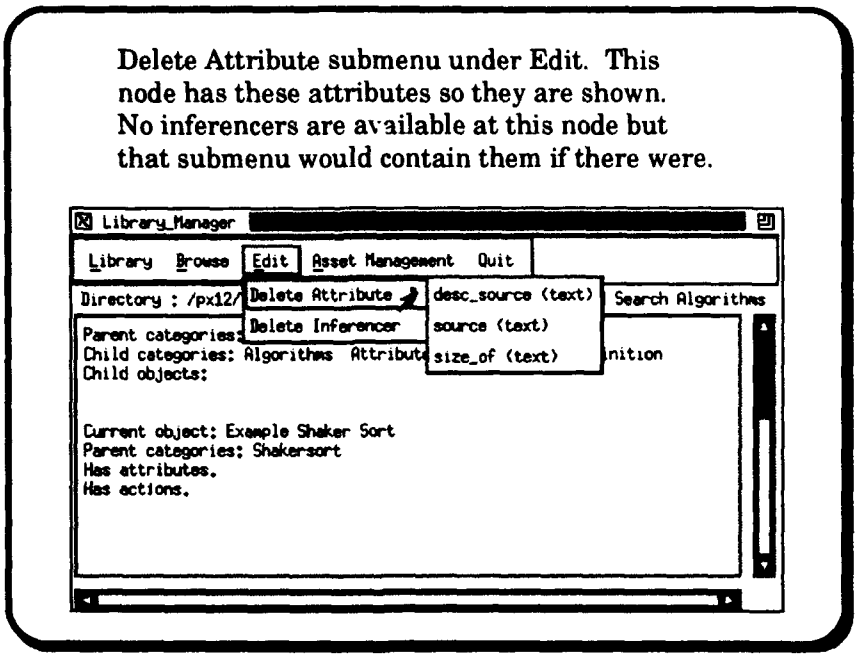
Slide 49

Examine Hierarchy submenu. Offers to display those items which exist at this node -- note that Display Relationships is always given as a choice.



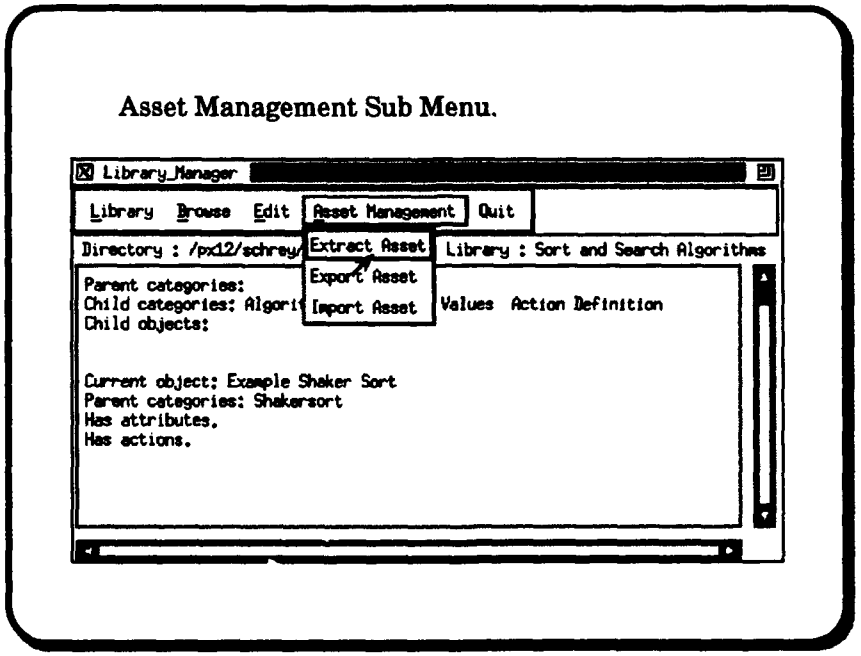
The current version of the **Library_Manager** provides two deletion options for node attributes and inferencers. A submenu of available attributes for the current node is shown on the next slide. If the user selects an available attribute or inferencer to delete, the item is deleted once the library has been saved with changes. There is no immediate confirmation that the deletion has been accomplished. Such deletions should be treated with caution as the corresponding LMDL spec for the library model has not been changed.

Slide 50



The current version of the **Library_Manager** is an initial version. Substantial improvements are planned for the near future. Currently, the **Asset Management** functions are limited to asset extraction to the user's working directory. **Export** and **Import** are not yet implemented – they are intended to support asset exchange between libraries.

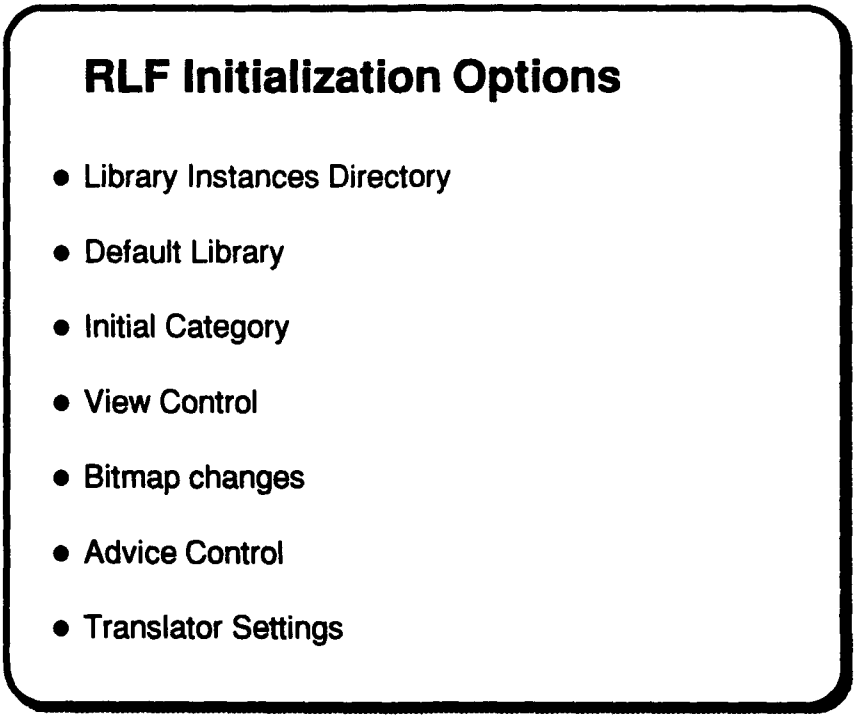
Slide 51



7 RLF Preferences File

RLF users and, in particular, RLF Administrators can personalize their interaction with the family of RLF applications by the installation and use of the `.rlfrc` preferences file. For example, SLIDE 52 summarizes some of tl e options which can be set with this file.

Slide 52



In general, following UNIX conventions, conflicting values established on an RLF command line override those set earlier, either as environment variables or as `.rlfrc` values, while any surviving environment variables override any values set in the `.rlfrc` file. Thus, if the `RLF_LIBRARIES` environment variable is set, it will override any `library directory` entry in the `.rlfrc` file.

A default library setting will cause the browser to open that library immediately. Within a library, a specific node can be chosen as the first focal point of the browser. View control can be used to select whether the topology view is on or off at start-up, and whether the first graphical view drawn is a specialization view or relationship view. The depth of the view can also be set.

If a user does not specify any advice options, the user must answer some start-up questions to select preferences during the first inferencing session. If a user wishes to establish alternate bitmaps for model icons, these can be set-up through the `.rlfrc` as well. The user can also select some options to be observed when running the RLF language translators `Lmdl` and `Rbdl`.

The complete syntax for the `.rlfrc` file is given in the Administrator's manual. A sample specification file is shown on the next multi-part slide.

Slide 53

Example .rlfrc File

```
--|
--| Sample startup file for the Reuse Library Framework version 4.1
--|

--|
--| Library directory or name specifications
--|

--library directory : /path/Libraries
--library : "Sort and Search Algorithms"

--|
--| Parameters for the RLF Graphical Browser
--|

topology : off
cardinality : off
layout offset : x : 20
layout offset : y : 5
history length : 50
view type : specialization
view depth : relationship : 2

--|
--| AdaTau inferencing settings
```

Slide 54

```
--|

advice : explanations : all
advice : automatic move : false

--|
--| Bitmaps for nodes
--|

--node bitmap : category : /path/box_m.xbm
--node bitmap : category : inferencer : /path/box_I_m.xbm
--node bitmap : category : actions : /path/box_A_m.xbm
--node bitmap : category : inferencer actions : /path/box_AI_m.xbm
--node bitmap : object : /path/cube_m.xbm
--node bitmap : object : inferencer : /path/cube_I_m.xbm
--node bitmap : object : actions : /path/cube_A_m.xbm
--node bitmap : object : inferencer actions : /path/cube_AI_m.xbm

--|
--| Specification translator settings
--|

translator: Lmdl: quiet: no
translator: Rbdl: quiet: no
```