SERIES 60 (LEVEL 68)

MULTICS LIBRARY MAINTENANCE PROGRAM LOGIC MANUAL PRELIMINARY EDITION

RESTRICTED DISTRIBUTION

SUBJECT

Preliminary Description of the Organization of the Multics System Libraries, and of the Procedures and Tools Used to Maintain These Libraries. The Organizational Information and Procedures Can Be Applied to Subsystem Libraries Developed under Multics, as well as to the System Libraries

SPECIAL INSTRUCTIONS

This preliminary Program Logic Manual (PLM) describes certain internal modules constituting the Multics System. It is intended as a reference for only those who are thoroughly familiar with the implementation details of the Multics operating system; interfaces described herein should not be used by application programmers or subsystem writers; such programmers and writers are concerned with the external interfaces only. The external interfaces are described in the *Multics Programmers' Manual, Commands and Active Functions* (Order No. AG92), *Subroutines* (Order No. AG93), and *Subsystem Writers' Guide* (Order No. AK92).

As Multics evolves, Honeywell will add, delete, and modify module descriptions in subsequent PLM updates (refer to Appendix A for an outline of planned additions to each of the missing sections of this document). Honeywell does not ensure that the internal functions and internal module interfaces will remain compatible with previous versions.

The information contained in this document is the exclusive property of Honeywell Information Systems. Distribution is limited to Honeywell employees and certain users authorized to receive copies. This document shall not be reproduced or its contents disclosed to others in whole or in part.

ORDER NUMBER

AN80-00

May 1979

Honeywell

Preface

Multics PLMs are intended for use by Multics system maintenance personnel, development personnel, and others who are thoroughly familiar with Multics internal system operation. They are not intended for application programmers or subsystem writers.

The PLMs contain descriptions of modules that serve as internal interfaces and perform special system functions. These documents do not describe external interfaces, which are used by application and system programmers.

Since internal interfaces are added, deleted, and modified as design improvements are introduced, Honeywell does not ensure that the internal functions and internal module interfaces will remain compatible with previous versions.

© Honeywell Information Systems Inc., 1979
File No.: 2L13 AN80

Throughout this manual, references are frequently made to the six manuals that are collectively referred to as the Multics Programmers' Manual (MPM). For convenience, these references are as follows:

Document	Referred To In Text As
Reference Guide (Order No. AG91)	MPM Reference Guide
Commands and Active Functions (Order No. AG92)	MPM Commands
Subroutines (Order No. AG93)	MPM Subroutines
Subsystem Writers' Guide (Order No. AK92)	MPM Subsystem Writers' Guide
Peripheral Input/Output (Order No. AX49)	MPM Peripheral I/O
Communication Input/Output (Order No. CC92)	MPM Communications I/O

CONTENTS

ILLUSTRATIONS

TABLES

SECTION I

INTRODUCTION TO LIBRARY MAINTENANCE

(to be supplied)

SECTION II

LIBRARY ORGANIZATION

(to be supplied)

SECTION III

THE MULTICS SYSTEM LIBRARIES

(to be supplied)

SECTION IV

THE LIBRARY DESCRIPTOR COMMANDS

Section II introduced the concept of a library descriptor data base and its accompanying library search procedures. The descriptor and search procedures provide information about the organization and contents of a library, and they provide a mechanism for finding particular library entries and for obtaining entry status information. This section describes how various library maintenance commands use library descriptors to help perform their maintenance function.

COMMANDS WHICH USE A LIBRARY DESCRIPTOR

Currently, five library maintenance commands use the information in library descriptors to perform their maintenance functions on the Multics System Libraries. These commands are listed in Table 4-1 below. Because they all use library descriptors, the commands are collectively called the library descriptor commands. Detailed descriptions of the commands may be found in Section XIII.

While the commands listed in Table 4-1 perform widely divergent maintenance functions, they all share a common interface to the library descriptor and this leads to similarities in their user interfaces and modes of internal operation. The discussion in this section highlights these similarities.

Table 4-1. Library Descriptor Commands

library_fetch	fetches entries from the library into the user's working directory.
library_info	prints information about library entries on the user's terminal.
library_map	generates a map of the library, giving selected status information about each library entry.
library_print	generates a file containing the contents of selected, printable library entries preceded by information about their current status. Page footings and an index are supplied to make it easy to find entries.
library_cleanup	deletes selected entries from the library. Selection is based upon the names of entries, and the time which has passed since their directory entry was last modified.

INPUT TO THE SEARCH MECHANISM

The lib_descriptor_ subroutine is the interface procedure between the library descriptor commands and the information and search procedures defined in a library descriptor. Each library descriptor command calls a separate entry point in the lib_descriptor_ subroutine to get information about entries in the library. The calling sequences for each of these entry points share the following set of arguments:

1. the name of the library descriptor to be used.
2. an array of library names identifying the libraries to be searched.
3. an array of search names identifying the library entries being searched for.

The user can specify values for these input arguments when invoking each library descriptor command by using the -descriptor, -library (-lb), and -search_name control arguments, respectively. In addition, each of the commands allows search names to be specified without using the -search_name control argument.

The lib_descriptor_ subroutine uses the descriptor name to obtain a pointer to the library descriptor data base. This data base contains the names of all libraries defined by the descriptor. The array of library names provided as an input argument is compared with the defined library names to determine which libraries are to be searched.

Associated with each library name is the pathname of the physical directory or archive which contains the library, and a procedure which can be called to search for entries in the library. The pathname of each identified library directory or archive is passed to its search procedure, along with the array of search names. The search procedure then returns a tree of status information describing the library entries which are found. This status information is sufficient to allow the library descriptor commands to perform their function on the found library entries.

Default Input Values

When the user invokes one of the library descriptor commands without giving library names, search names, or a -descriptor control argument, then the command calls the lib_descriptor_ subroutine with an empty name array or a blank descriptor name in place of the missing data. The lib_descriptor_ subroutine then uses default values to fill in the missing information.

The name of the default library descriptor is stored as an internal static variable by the lib_descriptor_ subroutine. Each of the library descriptor commands uses this default library descriptor when the user has not given the -descriptor control argument. The initial default library descriptor defines the Multics System Libraries, and has the name multics_libraries_. However, the default library descriptor can be changed as described under "Changing the Default Values" below.

Default library names and search names are stored in the library descriptor. Different defaults are defined for each library descriptor command when the descriptor is created. These defaults are used by the commands when the user has not given any library names or search names as command arguments.

The default library and search names must be stored in the library descriptor because each descriptor defines a unique set of libraries containing different types of entries stored under differing naming conventions. One set of default library names and search names cannot be appropriate for all possible library definitions.

Similarly, different default library and search names must be stored for each of the library descriptor commands because the functions performed by the commands are often more logically applied to some of the libraries defined by a descriptor than to others, and to some types of library entries than to others. For

example, the default values for the library_print command might cause printing all of the info segments in the info library; whereas, those for library_map might cause mapping the status of all entries in all of the libraries.

The particular default values which are used for a given command invocation are returned as output arguments in the blank library descriptor name string, and in the empty library and search name arrays. This allows the commands to use these default values in error messages and warnings which may be printed.

LISTING THE DEFAULT VALUES

The library_descriptor (lds) command prints information about library descriptors. It can be used to print the name of the default library descriptor; to print the default library names and search names for a given library descriptor and a given descriptor command; or to print information about the libraries which are defined by a given descriptor.

For example, the command:

```
library_descriptor name
```

prints the name of the default library descriptor on the user's terminal.

```
library_descriptor default library_map
```

prints the default library names and search names for the library_map command, as defined by the default library descriptor.

```
library_descriptor default -descriptor rdms_libraries_
```

prints the default library and search names for all library descriptor commands which are defined by the rdms_libraries_ library descriptor.

See the description of the library_descriptor command in Section XIII for complete details on its usage.

CHANGING THE DEFAULT VALUES

The library_descriptor command can also be used to change the name of the default library descriptor in a given user process. The command:

```
library_descriptor set rdms_libraries_
```

makes the rdms_libraries_ the default library descriptor for the process in which the command is issued.

The default library names and search names for any of the library descriptor commands can be changed by redefining these values in the library descriptor source segment and recompiling the descriptor. These operations are described in Section V.

SECTION V

MAINTAINING USER LIBRARIES WITH THE LIBRARY TOOLS

It may have occurred to you to ask, "Why use library descriptors to define the structure of the Multics System Libraries?" Library descriptors were introduced in Section II as a means of defining the logical structure of a library. However, this structure information could just as well have been built into the library maintenance commands, rather than using a separate data base. So far, the justification for having library descriptors has been implied, but not stated.

This section tries to answer the question above, and in so doing, it points out how the tools and procedures used to maintain the Multics System Libraries can be used for other libraries as well.

THE RATIONALE FOR LIBRARY DESCRIPTORS

As suggested in the opening paragraph of this section, library structure information could have been built into each library maintenance tool, rather than defining library structure in a library descriptor. In fact, this was done in the earliest versions of the maintenance tools. However, the pitfalls of this scheme were quickly discovered as the Multics System Libraries expanded and were reorganized to meet changing system needs. The following pitfalls were found.

1. Code to define the library structure and to search for library entries had to be duplicated in each library command. Since the commands were programmed by different people at different times, different mechanisms were usually used to define the structure and to search for entries, leading to differing user interfaces for the commands, duplication of design effort, and increased likelihood of bugs in the code.

2. All of the library commands had to be modified whenever a new library was added to the Multics System Libraries. During a period of rapid library growth, this led to modifications of all of the commands every few months.

3. When a new library organization was created (thankfully, an infrequent occurrence), mechanisms for defining its structure and searching for its entries had to be added to each of the library commands; this required an integration of the new mechanism with all of the different mechanisms which existed in these commands.

4. Although the commands performed generally useful library maintenance functions, they could only be used for the Multics System Libraries, much to the displeasure of subsystem writers.

To avoid these pitfalls, the early library commands have been rewritten to use a centralized, external subroutine to find library entries. The subroutine gets library structure information from a separate, easily modified, user-identified data base associated with each group of libraries. The subroutine is the lib_descriptor_ subroutine, and the data base is, of course, the library descriptor.

The basic operation of the lib_descriptor_ subroutine has already been discussed in Section IV. The next few paragraphs describe what library structure information is stored in a library descriptor, and how a Multics system programmer or a subsystem writer can define or change a library descriptor.

CONTENTS OF A LIBRARY DESCRIPTOR

In Section II under "The Logical Structure of Libraries", it was pointed out that most program libraries are logically structured like a tree with root directories or archives containing the different types of library entries (source segments, object segments, bind segments, listings, bound and unbound executable segments and data segments, etc). Figure 5-1 shows such a tree-structured library.

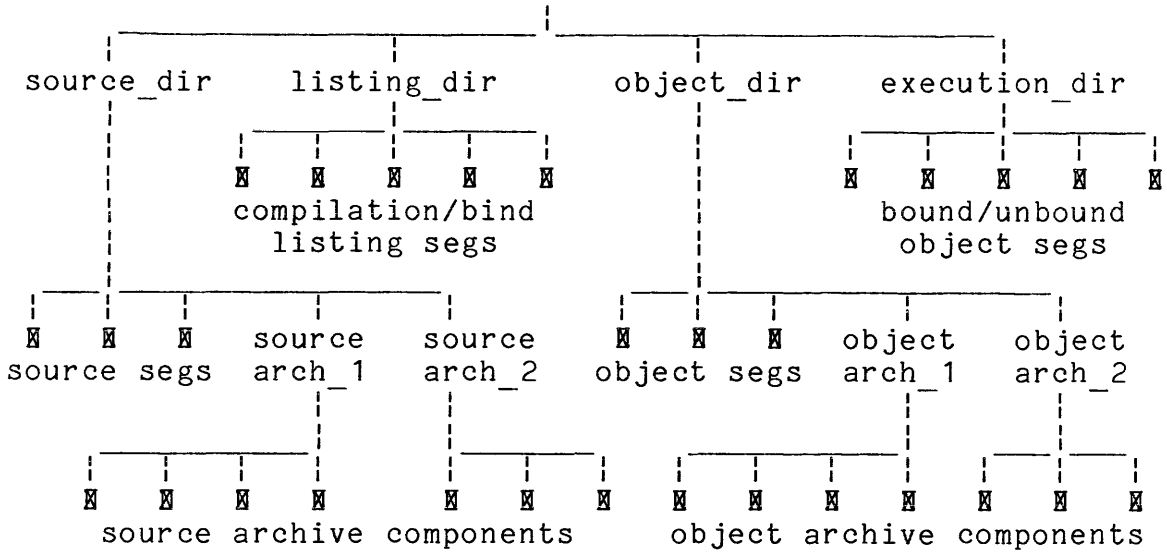


Figure 5-1. A Typical Library Tree

The library descriptor contains information about the roots of a library. lib_descriptor_ uses this information to find library entries. For each library root, the library descriptor defines the following set of information.

1. The logical library names by which the library root can be referenced.

2. The type of library root (either directory or archive).

3. The Multics storage system pathname of the physical directory or archive which is the library root.

4. The name of a program which knows how to search for library entries in the subtree below each library root. This program is called the search procedure for the library root.

In addition to the definitions of the library roots, a library descriptor defines which library descriptor commands can be used on the library, and what default library names and search names are to be used with each of these commands. Recalling from Section IV, when the user invokes a library descriptor command without giving library names or search names, then lib_descriptor_ uses default values defined in the library descriptor. The default library names and search names depend upon the structure of the libraries which the command is searching. Since this structure is defined in the library descriptor, the default library names and search names are also defined there.

With the library structure information and the command default information defined in a central data base which is used by all library descriptor commands, the system programmer can easily add new libraries by modifying the descriptor, and the subsystem writer can define a new library structure and use a ready-made set of commands to maintain the library.

THE LIBRARY DESCRIPTION LANGUAGE

Library descriptors are created by the library_descriptor_compiler command, which is described in Section XIII. This compiler accepts a source language description of a library structure as input, and produces an ALM data segment as output. When assembled, the ALM data segment becomes the library descriptor.

The source language accepted by the library_descriptor_compiler is called the library description language. It contains statements for naming a library descriptor, for defining the roots of a library structure, and for defining library descriptor command default values.

General Syntax

The statements in the library description language have the general syntax:

```
keyword: parameters;
```

where:

1. keyword is an identifier which names the statement.

2. parameters are one or more values associated with the statement. Parameters are separated from one another by one or more spacing characters (see Table 5-1 below).

Description Delimiters

The delimiters shown in Table 5-1 below are used in the library description language. These delimiters separate the statements in a library description source segment, and they separate the statement components (keywords and parameters) within each statement.

Table 5-1. Library Description Language Delimiters

:	keyword delimiter. It follows the keyword which names the statement, and separates this keyword from the parameter value.
;	statement delimiter. It ends each statement.
"	quoting character. It begins and ends each quoted string. A quoted string is treated as a single unit in the language, even though it may contain other delimiters. The PL/I quoting convention is followed: a pair of quoting characters ("") appearing together in a quoted string represents a single quoting character in that string.
(	begins a group of root name components in a compound root name appearing in a Root statement.
)	ends a group of root name components in a compound root name appearing in a Root statement.
/*	begins a comment.
*/	ends a comment.
space, horizontal tab, new line, new page	spacing characters. These characters may appear before or after any statement component or delimiter. The separate parameters from one another and space statements across the page for improved readability of the source.

A Complete Library Description

A complete library description:

1. begins with a Descriptor statement
2. contains a Define statement with several substatements to define attributes associated with the entire descriptor
3. has one or more Root statements with substatements which describe the roots of a library structure
4. ends with an End statement

The structure is:

```
Descriptor: ... ;
Define: ... ;
  .
  .
Root: ... ;
  .
  .
End: ... ;
```

Each of the statements listed above is a major statement in the library description language and each defines a unit of data in the descriptor. Major statements have a keyword identifier which begins with a capital letter.

Define and Root statements may be followed by minor statements which add information to the definition or root description. Minor statements have a keyword identifier which begins with a lower case letter.

The major statements and their minor statements are described below in detail. A detailed example which shows how to use each of the major and minor statements is included under "A Sample Library Description" below.

Descriptor Statement

A Descriptor statement begins a library description and defines the name of the library descriptor. It must be the first statement of a library descriptor definition.

A Descriptor statement has the syntax shown below.

```
Descriptor:  descriptor_name;
```

where descriptor_name is the name of the descriptor. It must begin with an alphabetic character, and may contain 1 to 32 alphanumeric characters or underscores (_).

Define Statements

A Define statement and its minor statements define attributes associated with the library descriptor.

Currently, only one kind of Define statement is implemented by the library description language: the Define commands statement.

Define Commands Statement

A Define commands statement adds no information to the library description, but serves mainly as a delimiting statement. It identifies the minor statements which follow as statements defining which library descriptor commands are supported for use on the libraries defined by the descriptor, and what their default library and search name command arguments are. As a delimiting statement, it has a fixed parameter value as shown below.

```
Define:  commands;
```

Two kinds of minor statements may follow a Define commands statement. A command statement defines a library descriptor command which is supported for use on the libraries described by the descriptor. An unsupported command statement defines a library descriptor command which is not supported for use on these libraries. These two minor statements are described in the next few paragraphs.

A Define commands statement and its minor statements have the syntax shown below.

```
Define:  commands;
  command:  command_name;
    library names:  star_names;
    search names:   star_names;
  unsupported command:  command_name;
```

One or more minor statements must follow the Define commands statement. At least one of these must be a command statement.

COMMAND STATEMENT

A command statement is a minor statement. It defines a library descriptor command which is supported for use on the libraries described by the descriptor.

A command statement has the syntax shown below.

```
command: command_name;
```

where command_name is the full name or abbreviated name of any of the library descriptor commands listed under "Commands Which Use a Library Descriptor" in Section IV.

A command statement may be followed by one or both of the following minor statements: a library names statement, and a search names statement. A library names statement defines the default library names to be used with the command when the user omits library names from the command line. A search names statement defines the default search names to be used with the command when the user omits search names from the command line.

These two minor statements have the syntax shown below.

```
library names: star_names;

search names: star_names;
```

where star_names are one or more entrynames in which the Multics star convention may be used to identify several libraries or library entries with a given entryname. If several star_names are given, they are separated from one another by spacing characters (see Table 5-1 above).

UNSUPPORTED COMMAND STATEMENT

An unsupported command statement is a minor statement. It defines a library descriptor command as not supported for use on the libraries described by the descriptor. Any attempt to use the command on these libraries fails with an appropriate error message.

An unsupported command statement has the syntax shown below.

```
unsupported command: command_name;
```

where command_name is the full name or abbreviated name of any of the library descriptor commands listed under "Commands Which Use a Library Descriptor" in Section IV.

An unsupported command statement might be used when it is undesirable or inappropriate to allow a particular library descriptor command to be used on a set of libraries, or when the search procedure for the libraries is not programmed to search for library entries according to the requirements of the command.

NOTES

If no command or unsupported command statement appears for a given library descriptor command, then that command is not supported for use on the library structure described by the descriptor. Making the commands unsupported by default gives the library maintainer a chance to evaluate new library descriptor commands before allowing them to be used on the libraries.

Since library descriptors are used solely by the library descriptor commands, it follows that a Define commands statement followed by at least one command minor statement must appear in every library description. If this were not the case, then no library descriptor commands would be supported for use on the libraries described in the descriptor, and the descriptor would be useless.

If no library names statement or search names statement follows the command statement for a particular library descriptor command, then no corresponding default values are defined for that command. The user is required to give the library names or search names each time that command is invoked.

The same library descriptor command should not be given in more than one command statement or unsupported command statement. However if this should occur by error, the last such definition is compiled into the library descriptor. Note that the user is not informed of such duplication.

Root Definitions

The main purpose of a library descriptor is to describe a library structure. Each root of this library structure is described by a Root statement followed by several minor statements: an optional type statement; a path statement; and a search procedure statement. These statements are described in the next few paragraphs.

A complete root definition has the syntax shown below.

```
Root:  root_names;
  type:  root_type;
  path:  root_pathname;
  search procedure:  search_entry_point;
```

A Root statement begins the description of a library root. It defines the logical names by which the library root is referenced in library descriptor commands. All minor statements following the Root statement (until the next major statement is encountered) further describe the root.

A Root statement has the syntax shown below.

```
Root: root_names;
```

where root_names are the logical names of the library root. The root_names may be given in two forms: single root_names and compound root_names.

A single root_name is a name consisting of 1 to 32 ASCII characters except the characters: < > ( ) * ? = %. Single root_names are separated from one another by spacing characters (see Table 5-1 above). Examples of single root_names are:

```
online standard.source languages.execution lib1.exp.source
```

A compound root_name is a collection of names represented as the cross-product of several groups of name components. Each group of components is enclosed in parentheses and separated from the group which follows by a period. An example is:

```
(online standard).(source s)
```

This example is equivalent to the single root_names:

```
online.source online.s standard.source standard.s
```

The root names formed by the cross-product must meet the requirements of single root_names. They must consist of 1 to 32 ASCII characters except the characters: < > ( ) * ? = %. A compound root_name has the syntax shown below.

```
(root_name_components){.(root_name_components)}...
```

where the root_name_components are ASCII characters (except: < > ( ) * ? = %) separated from one another by spacing characters.

While null character strings ("") cannot be used as single root_names, they can be used as root_name_components in a compound root_name. If a null string component is found while performing the cross-product operation for a compound root_name, then the null component is omitted from that step of the cross-product operation. For example:

```
(online standard "").(source s "")
```

is equivalent to the root_names:

```
online.source online.s online standard.source
standard.s standard source s
```

Note that, when the cross-product operation selects a null string from all groups of components, the resulting root_name is a null string. Since null strings are illegal root_names, the null string is ignored by the cross-product operation.

TYPE STATEMENT

A type statement is a minor statement which defines the type of library root being described. Directories and archives may be defined as library roots.

A type statement has the syntax shown below.

```
type: root_type;
```

where root_type may be "directory" or "archive".

A type statement is optional. If it is omitted from a root description, then the root is assumed to be a directory.

PATH STATEMENT

A path statement is a minor statement which defines the library root's pathname in the Multics storage system.

A path statement has the syntax shown below.

```
path: root_pathname;
```

where root_pathname is the absolute pathname of the library root.

A path statement is required in each root description. It must appear after the Root statement which names the library root, and before the next major statement.

SEARCH PROCEDURE STATEMENT

A search procedure statement is a minor statement which defines the procedure entry point which finds entries in the library root.

A search procedure statement has the syntax shown below.

```
search procedure: search_entry_point;
```

where search_entry_point is the name of a procedure entry point. Either of the following forms may be used for the

search_entry_point.

```
ref_name
ref_name$entry_point_name
```

Refer to "Reference Names" and "Entry Point Names" in Section III of the MPM Reference Guide for more information about the terms, reference_name and entry_point_name.

A search procedure statement is required in each root description. It must appear after the Root statement which names the library root, and before the next major statement.

End Statement

An End statement ends a library description. It must be the last statement of a library descriptor definition.

An End statement has the syntax shown below.

```
End: descriptor_name;
```

where descriptor_name is the name of the library descriptor which was given in the Descriptor statement.

PREPARING A LIBRARY DESCRIPTION

The paragraphs above define the syntax and semantics of the library description language. The next few paragraphs provide practical hints on how to use the various statements in the library description language, and they show an example of a library description.

Descriptor Names

There should be a direct mapping between the descriptor_name used in the Descriptor statement of a library description and the entryname of the source segment which contains that description. The entryname should be the descriptor_name followed by an ld suffix. For example, a descriptor named multics_libraries would be defined in a source segment called multics_libraries.ld.

The mapping must be maintained to avoid user confusion. Confusion can occur if the names are different. The entryname on the source segment is used to name the compiled library descriptor segment. However, the descriptor_name compiled into the library descriptor is reported by the library_descriptor command as the name of the current descriptor. A user might be confused if the library_descriptor command reported the name of the current descriptor as descriptor_1, but there was not segment called descriptor_1 in the user's search directories.

Multics system naming conventions for system subroutines and data bases require that the descriptor_name of system library descriptors end with an underscore (_). These conventions should be followed when selecting a descriptor_name.

Defining Commands

When a library descriptor command is given in an unsupported command minor statement of the Define commands statement, then that library command is prevented from operating on the library structure defined by the descriptor. Any attempt to use the command with this library descriptor causes an error message to be printed stating that the library descriptor does not support the command.

An unsupported command statement should be used when the function performed by a particular library descriptor command is not appropriate to the libraries defined by the descriptor, or when the search procedures used for these libraries do not support a particular library descriptor command.

When default library names or search names are defined for use with a supported library command on a given library structure, the user can determine these default values before using the command by way of the library_descriptor command. Also, any default values which are used by the command are printed when errors occur to ensure that user knows what default values were being used when the error occurred.

Each library descriptor command prints an error message if it is invoked without a search name or library name when no default search names or library names were given after its command statement in the library descriptor.

Root Names

When more than one library is described by a library description, it is common to give the roots multicomponent names. The first component identifies the library, and the second component identifies the type of entries stored in that root of the library. The example in Figure 5-2 below demonstrates this usage.

A Sample Library Description

Figure 5-2 below shows a sample description of a library structure.

```
Descriptor:  sample_libraries;

Define:  commands;
  unsupported command:  library_cleanup;
  command:  library_fetch;
  command:  library_info;
    library names: **;
  command:  library_map;
    library names:  source object;
    search names: **;
  command:  library_print;
    library names:  source include;
    search names:   *.pl1  *.alm  *.incl.*  *.ec;

/* Define the standard library */

Root:  (standard std "").(source s "") both;
  type:  directory;
  path:  >ldd>standard>source;
  search procedure:  standard_search$source;
Root:  (standard std "").(object o "") both;
  type:  archive;
  path:  >ldd>object>standard.archive;
  search procedure:  standard_search$object;

/* Define the experimental library */

Root:  (experimental x "").(source s "") both;
                          /* defaults to type: directory;  */
  path:  >ldd>experimental>source;
  search procedure:  experimental_search_procedure;
Root:  (experimental x "").(object o "") both;
  type:  archive;         /* type statement required here. */
  path:  >ldd>object>experimental.archive;
  search procedure:  standard_search$object;

/* Define include directory shared by both libraries. */

Root:  (standard std experimental x "").(include incl "")
       both;
  path:  >ldd>both>include;
  search procedure:  experimental_search_include;

End:  sample_libraries;
```

Figure 5-2. A Sample Library Description

The example in Figure 5-2 describes the following two libraries.

LIBRARY ID	LIBRARY CONTENTS
standard, std	the library containing standard, fully-tested programs and data.
experimental, x	the library containing experimental programs and data.

Each of these libraries contains the following library roots.

ROOT ID	ROOT CONTENTS
source, s	source segments for the programs and data in the library.
object, o	object segments for the programs and data in the library.
include, incl	include segments required to compile the source programs in the library.

The following library naming conventions have been applied in the library description above.

1. A library identifier from the table above can be used as a library name to reference all of the roots of that library.

2. A root identifier from the table above can be used as a library name to reference all library roots of the same type (e.g., source roots, object roots, include root).

3. A two-component library name of the form:

   library_identifier.root_identifier

   can be used to reference a particular root within a given library. For example, standard.source or experimental.include are such two-component library names.

4. The roots of both libraries can be referenced by the library name "both".

In addition, the following attributes of library descriptor commands are defined by the description in the example in Figure 5-2.

COMMAND	DEFAULT LIBRARY NAMES	DEFAULT SEARCH NAMES
library_cleanup	(unsupported)	
library_fetch	(none)	(none)
library_info	**	(none)
library_map	source, object	**
library_print	source, include	*.pl1, *.alm, *.incl.*, *.ec

CODING A LIBRARY SEARCH PROCEDURE

The techniques for coding a library search procedure will be described sometime in the future. However, the search procedure used for the Multics System Libraries, multics_library_search_, can be used for other libraries as well, as long as they are organized like the Multics System Libraries. Refer to the description of multics_library_search_ in Section XIV for more information about this search procedure.

SECTION VI

CROSS REFERENCING TOOLS

(to be supplied)

SECTION VII

ONLINE LIBRARY MODIFICATION

(to be supplied)

SECTION VIII

SUPERVISOR LIBRARY MODIFICATION

(to be supplied)

SECTION IX

COMMUNICATIONS LIBRARY MODIFICATION

This section describes the procedures used to modify the Multics Communications System.

THE MULTICS COMMUNICATION SYSTEM

The Multics Communication System is a series of programs written to operate the Multics Front-End Network Processor (FNP). This FNP handles the communication functions between Multics and user terminals and other remote devices. The FNP is a minicomputer with an 18-bit word and an instruction set similar to (though more limited than) that of the Honeywell 68/80 computer.

Communication programs are written in a special FNP assembler language called map355. An assembler for the map355 language is available on Multics under the GCOS Environment Simulator. The map355 command described in Section XIII of this manual provides a convenient, compiler-like interface to this GCOS assembler.

Communication programs employ several assembler macros to perform macro-operations. These macros are defined in a single segment, macros.map355, which must be compiled by a GCOS job using the GCOS Environment Simulator. For information about this simulator, refer to the gcos module description in the MPM Commands, and to the GCOS Environment Simulator manual, Order No. AN05.

The object segments generated by map355 for the communication programs are bound together by the bind_fnp command to form a core image which can be loaded into the FNP by Multics. Some programs must be kept unbound from the majority of communication programs. Core images for these programs are created by the coreload command. Refer to the bind_fnp and coreload module descriptions in Section XIII for information about these commands.

For more information about the communication programs, and the data structures they use, refer to the Multics Communications

System PLM, Order No. AN85.

THE COMMUNICATIONS LIBRARY

The source, object, and core images of the communications system reside in the Communications Library. This library has the structure shown in Figure 9-1.

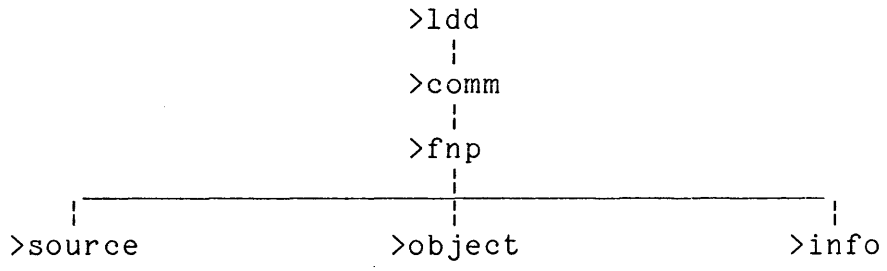


Figure 9-1. Structure of the Communications Library

The source segments for communication programs have a suffix of map355, and reside in the source directory. The object segments have a suffix of objdk, and reside in the object directory. The core image segments have no suffix, and also reside in the object directory. The source and object for the macros, the GCOS job used to assemble the macros, and the bindfile segment controlling the binding of communication programs by bind_fnp all reside in the info directory.

The Multics library descriptor identifies this library as the communications_library (com or comm), with directories: source (s), object (o), and info. The communications_library is a member of the offline_libraries (offline or off) group of libraries.

STRATEGY FOR COMMUNICATION SYSTEMS MODIFICATION

Modifying the communication system involves five steps.

1. Create a modification directory and a listings directory. The modification will be prepared in these directories.

2. Prepare the modification. This involves one or more of the following operations.

   A. Reassemble the macro file, if changes were made to the macros.
   B. Compile modified communication source programs to produce object decks.
   C. Bind or core load object decks to produce core images which can be loaded into the FNP.

3. Place links from the modification directory of the new Supervisor system which is to contain the modification to the core image segments in the modification directory.

4. Generate a new Multics Supervisor system tape containing the modification.

5. Install the modification in the Communications Library whenever the Supervisor modification is installed in the Supervisor Library.

These steps are described in more detail below.

The core images produced by step 2C above are listed in the Multics Supervisor header segment, and are included in the Multics Supervisor when a new Supervisor system tape is generated. During Supervisor tape generation, the core image segments are referenced by way of the links created in step 3 above. Refer to Section VIII for information about generating a new Multics Supervisor system tape.

CREATE DIRECTORIES TO HOLD THE MODIFICATION

The first step in modifying is to create a modification directory, and listing directory. The following command does this.

```
ec >LDD>EC>init_m N
```

where N is the version number of the new communication system.

The modification directory is created at

```
>ldd>comm>fnp>mcs.N
```

Copy into this directory the new or modified source segments (NAME.map355, etc), macro source (macros.map355), and bindfiles (mcs.bind_fnp, etc). These segments will be prepared in the modification directory for installation.

The listings directory is created at

```
>ldd>listings>mcs.N
```

As the modification is prepared, all listings which are generated are placed in the communication listings directory to separate the listings from the actual segments of the modification.

PREPARING THE MODIFICATION FOR INSTALLATION

Perform the following steps to prepare the modification for use and for installation into the Communications Library.

Generating a New Macro File

If the modification includes a new set of macros, then the source for these macros must be compiled. Otherwise, skip to "Compiling New or Modified Communication Programs" below.

The source for the macros is named macros.map355. To compile this source, issue the following command.

```
ec >LDD>EC>compile_m N macros.map355
```

where N is the version number of the new communication system, and macros.map355 names the macro source segment. The new macros.map355 source segment must be in the modification directory.

compile_m.ec issues a gcos command to compile the macros. This command runs the GCOS job,

```
>ldd>comm>fnp>info>macros_asm
```

under the GCOS Environment Simulator.

compile_m.ec uses or generates the following segments as part of the compilation process.

macros.map355
: The macro source segment which is compiled. It must be in the modification directory.

355_macros
: The new set of macros which is created in the modification directory, and which will be used when compiling other map355 segments.

355_macros.list
: A listing of the compilation results, which is placed in the listings directory.

macros_asm.sysprint.list
macros_asm.sysprint
: Extraneous segments created by the GCOS job. They are normally deleted by compile_m.ec, but if the exec_com is interrupted, these segments may be present in the modification directory. Run compile_m.ec to completion, and it deletes them.

Compiling New or Modified Communication Programs

The next step in preparing a modification is to compile any new or modified source programs. Segments with a map355 suffix contain the source for the various communication programs. For example, a segment called NAME.map355 would contain the source for the NAME program. To compile new or modified programs, issue the following command.

```
ec >LDD>EC>compile_m N SOURCE_SEG_NAMES.map355
```

where N is the version number of the new communication system, and SOURCE_SEG_NAMES.map355 are the segment names of one or more source segments to be compiled, including the map355 suffix. The source segments must be in the modification directory. For example, the following command compiles the init and utilities programs.

```
ec >LDD>EC>compile_m 2.01 init.map355 utilities.map355
```

The star convention may <u>not</u> be used in the SOURCE_SEG_NAMES.map355, but the segs active function can be. For example,

```
ec >LDD>EC>compile_m 2.01 [segs *.map355]
```

compile_m.ec issues a map355 command to compile each source segment. map355 provides a compiler-like interface to the compiler of the map355 language which exists under the GCOS Environment Simulator.

compile_m.ec invokes map355 with control arguments which produce a compiled object segment and a compilation listing for each source program which is compiled. The segments used or generated by compile_m.ec are summarized below.

NAME.map355 The communication source program which is compiled. It must be in the modification directory.

NAME.objdk The communication object segment generated from the source segment in the modification directory.

NAME.list The compilation listing which documents the compilation. It is moved to the listings directory.

If a new copy of 355_macros exists in the modification directory, then compile_m.ec uses this copy in compiling the source programs. For this reason, it is important to compile the new macros BEFORE compiling any other source programs.

NOTE: If map355 is interrupted while compiling a segment (by a system crash or fatal process error, etc.), the links which it creates to reference segments in the process directory will still exist in the new process. When map355 is run in the new process, it will attempt to reuse these links to the old process directory (which no longer exists) and will encounter errors. In such cases, map355 prints a suitable error message, unlinks the bad links, and returns. Therefore, subsequent invocations of map355 in the new process will work correctly.

Generating Core Images

The last step in preparing a modification is to convert the object segments into core images which can be loaded directly into the FNP. This conversion is performed for different object segments in one of two ways:

1. Many object segments are bound together into a single core image, using the bind_fnp command under the control of a bindfile.

2. Single object segments are converted into core images by using the coreload command.

The mcs and site_mcs core images are created by bind_fnp; the gicb core image is created by coreload.

BINDING OBJECT SEGMENTS TOGETHER

To bind several object segments into a single core image, issue the following command.

```
ec >LDD>EC>bind_m N BINDFILE.bind_fnp
```

where N is the version number of the new communication system, and BINDFILE.bind_fnp is the name of the bindfile used by bind_fnp to control which segments are bound. The name should end with a bind_fnp suffix.

bind_m.ec invokes the bind_fnp command to create a core image. bind_fnp searches for the object segments named in the bindfile first in the modification directory, and then in the Communications Library object directory, >ldd>comm>fnp>object.

bind_m.ec invokes bind_fnp with control arguments which produce a core image segment and a bind listing. bind_m.ec uses or generates the following segments.

NAME.bind_fnp — The bindfile which controls which communication object segments are bound together, etc. If this bindfile is being modified as part of the modification, then the modified copy must be in the modification directory. Otherwise, bind_m.ec use the installed copy of the bindfile in the Communications Library info directory, >ldd>comm>fnp>info.

NAME — The core image segment which bind_fnp creates in the modification directory.

NAME.list — The bind listings which documents the binding process. It is moved to the listings directory.

As an example, if the Multics communication system bindfile (mcs.bind_fnp) is being changed as part of Version 2.00, then the new bindfile should be in the modification directory. The following command creates a new communication system core image.

```
ec >LDD>EC>bind_m 2.00 mcs.bind_fnp
```

If Version 2.01 does not change the bindfile, then the bindfile installed in the Communications Library info directory, >ldd>comm>fnp>info, is automatically used as shown in the following command:

```
ec >LDD>EC>bind_m 2.01 mcs.bind_fnp
```

bind_m.ec prints out the names of the object programs which were bound, and the bind map entries for those programs found in the modification directory. This information can be used to check that all new or modified programs which should be included in the core image actually were included.

CONVERTING SINGLE OBJECT SEGMENTS

To convert a single object segment into a core image, issue the following command, while in the modification directory.

```
coreload NAME.objdk
```

where NAME.objdk is the object segment to be converted. The coreload command then creates NAME in the modification directory.

LINKING TO AND USING THE NEW CORE IMAGES

Generation of core image segments by bind_m.ec or by the coreload command completes the preparation of a modification to the communication system. All that remains is to use the modified core image segments in a new Multics Supervisor system.

To use the new core images, place a link from the new Supervisor modification directory to each new core image segment in the modification directory. Then generate the new Supervisor system, as described in Section VIII.

If a totally new core image segment has been created as part of the modification to the communication system, then the Supervisor header segment must be modified to name this new core image. Modification of the header segment is also described in Section VIII.

INSTALLING THE MODIFICATION

(to be supplied)

TESTING THE MODIFICATION

(to be supplied)

DE-INSTALLING THE MODIFICATION

(to be supplied)

DOCUMENTING THE MODIFICATION

(to be supplied)

MODIFICATION TOOLS

The following is a summary of the tools used to modify the communication system. All exec_com segments reside in >LDD>EC. Other tools reside in >tools, or are in the one of the other Multics System Libraries.

```
ec >LDD>EC>init_m N
```

creates a communication modification directory and a listings directory (>ldd>comm>fnp>mcs.N) (>ldd>listings>mcs.N).

```
ec >LDD>EC>compile_m N SOURCE_SEG_NAMES.map355
```

compiles communication source programs and a macro source segment. Segments to be compiled must reside in the modification directory. Compiled object segments are placed in this directory, and listings are placed in the communication listings directory.

```
ec >LDD>EC>bind_m N BINDFILE.bind_fnp
```

binds several communication object segments together into a single core image segment. The bindfile must reside in the modification directory if it has been modified, or in the Communications Library info directory (>ldd>comm>fnp>info) otherwise. Object segments reside either in the communication modification directory or in the Communications Library object directory (>ldd>comm>fnp>object). The core image segment is created in the modification directory, and a bind listing is moved to the listings directory.

```
bind_fnp BINDFILE>bind_fnp -list
```

used by bind_m.ec to bind several object segments into a core image segment.

```
coreload NAME.objdk
```

converts a single communication object segment into a core image segment.

```
gcos  >ldd>comm>fnp>info>macros_asm -list -lower_case -brief
```

used by compile_m.ec to compile a new macro source segment.

```
map355 NAME.map355 -list -macro_file 355_macros
```

used by compile_m.ec to compile a communication system source program, optionally with a modified set of macros.

SECTION X

BOOTLOAD LIBRARY MODIFICATION

(to be supplied)

SECTION XI

WHEN THE SYSTEM LIBRARIES SELF-DESTRUCT

(to be supplied)

SECTION XII

LIBRARY MAINTENANCE EXEC_COM'S

(to be supplied)

SECTION XIII

LIBRARY TOOLS

This section contains command descriptions for the tools used in library maintenance. The use of many of these tools have been discussed in Section IV, Section VI, Section VII, Section VIII, Section IX, and Section X. In addition, many of these commands are used by the exec_com segments described in Section XII.

Name: bind_fnp

This command produces a core image segment that can be loaded into the FNP. It uses two control segments: a bindfile which specifies the configuration that the FNP will support, the names and ordering of the object segments included in the core image, and the size of certain software tables; and an optional search rules segment which specifies which directories are searched to find the object segments.

This command is fully described in the MAM Communications manual, Order No. CC75.

Names: lfree_name, lfn

This command is part of the Multics Installation System (MIS) which is used to install modifications in the Multics Online Libraries. The command interfaces with the installation subroutine which frees names on one directory entry so that those names can be used on a replacement entry.

An entryname is freed according to the following algorithm:

1. If the name ends with an integer suffix, then the name is freed by incrementing the suffix by 1. For example, qedx.1 becomes qedx.2.

2. If the name does not end with an integer suffix, then the name is freed by adding a 1 suffix. For example, edm becomes edm.1.

3. If the freed name is longer than 32 characters, then the portion of the name preceding the integer suffix is truncated before the integer suffix is added or incremented. For example,
   bound_misc_translatrs_.s.archive
   becomes
   bound_misc_translatrs_.s.archi.1

4. If another entry which has the freed name already exists in the directory, then that entryname is freed. This means that, if teco and teco.1 are names on two segments in the same directory, freeing the name teco produces teco.1; this causes teco.1 to be freed, producing teco.2.

Usage

lfree_name pathname

where pathname is the relative or absolute pathname which identifies the entry whose name is to be freed. Only the final entryname of the pathname is freed. All other names on the entry remain intact. The Multics star convention may not be used.

Entries: lfree_name$restore, lfn$restore

This entry point in the command unfrees (or restores) a freed entryname by reversing the algorithm described above.

Usage

```
lfree_name$restore  pathname
```

where pathname is the relative or absolute pathname which identifies the restored entryname. A freed name is constructed from this entryname, the directory entry having the freed name is found, and its name is restored to entryname.

Note

lfree_name calls an installation subroutine which is part of the Multics Installation System to free entry names. This installation subroutine, in turn, calls the installation_tools_ gate into ring 1 to allow the names on ring 1 library segments to be freed. However, maintainers of outer ring libraries do not have access to this privileged gate. They can use lfree_name to free and restore entrynames by initiating the hcs_ gate with the reference name installations_tools_ once per process before using lfree_name. The following command will perform this function:

```
initiate [get_pathname hcs_] installation_tools_
```

Examples

If a bound segment in the working directory has the names bound_qedx_, qedx, and qx, then the command

```
lfree_name (bound_qedx_ qedx qx)
```

frees those names. If qedx.1 already exists in the working directory, that name is freed to qedx.2.

```
lfree_name$restore (bound_qedx_ qedx qx)
```

restores all of these names to their original values. Note that the arguments given to lfree_name and lfree_name$restore are the same, the unfreed entrynames. lfree_name frees these entrynames, while lfree_name$restore constructs freed names from these entrynames, and restores entrynames which match those freed names.

Names: library_cleanup, lcln

This command deletes library entries which are no longer needed. Segments, links, and multisegment files may be deleted in this manner.

Library entries matching one or more search name arguments are selected as candidates for possible deletion. If they have not been modified within a given grace period, then they are eligible for deletion.

By default, library_cleanup only lists the entries eligible for deletion. The -delete control argument must be given to cause deletion of these entries.

This command uses a library descriptor and library search procedures, as described in Section IV.

Usage

```
library_cleanup {search_names} {-control_args}
```

where:

1. search_names are entrynames which identify the library entries which are candidates for deletion. The Multics star convention may be used to identify a group of entries with a single search name. Up to 30 search names may be given in the command. If none are given, then any default search names specified in the library descriptor are used.

2. control_args are selected from the following list of control arguments and can appear anywhere in the command:

   -delete, -dl causes the library entries which are eligible for deletion to be deleted.

   -list, -ls causes the library entries which are eligible for deletion to be printed on the user's terminal. This is the default if neither -delete, -list, nor -long is given.

-long, -lg causes all library entries which match the search names to be printed on the user's terminal, even if they are not eligible for deletion according to their date/time entry modified. Entries which are eligible for deletion are flagged with an asterisk (*).

-time days
-tm days gives a grace period in days. Matching library entries whose date/time entry modified falls within this grace period are not eligible for deletion. The default grace period is seven days.

-library library_name,
-lb library_name
identifies a library which is to be searched for entries to be deleted. The Multics star convention may be used to identify a group of libraries with a single library name. Up to 30 -library control arguments may be given in each command. If none are given, then any default library names specified in the library descriptor are used.

-search_name search_name
identifies a search name which begins with a minus (-) to distinguish the search name from a control argument. There are no other differences between the search names described above and those given with the -search_name control argument. One or more -search_name control arguments may be given in the command.

-descriptor desc_name
gives a pathname or reference name which identifies the library descriptor describing the libraries to be searched. If no -descriptor control argument is given, then the default library descriptor is used.

Notes

If the -delete and -list control arguments are used together, then the library entries being deleted are printed on the user's terminal.

If an entry which is eligible for deletion resides in an inner ring, library_cleanup must call the restricted installation_tools_ gate to change its ring brackets prior to deleting it. If the user does not have access to this gate, then the entry in not deleted, and a linkage error occurs.

Names: library_descriptor, lds

A library descriptor is a data base which associates directories or archives in the Multics storage system with the roots of a logical library structure. Library descriptors are discussed in detail in Section II.

This command prints information about library descriptors on the user's terminal, and controls the use of library descriptors by the other library descriptor commands. It can print the pathname of the directory or archive associated with a library root; can print detailed information about one or more library roots; can set and print the name of the default library descriptor used by the other library descriptor commands; and it can print the default library and search names associated with each library descriptor command. The relationship between library_descriptor and the other library descriptor commands is discussed further in Section IV.

Usage

library_descriptor key {arguments}

where the keys and their arguments are described in the paragraphs which follow.

Key: name, nm

The name key returns the name of the default library descriptor which is currently being used. library_descriptor may be invoked as an active function when the name key is used.

Usage

library_descriptor name

Key: set

The set key sets the name of the default library descriptor.

Usage

library_descriptor set desc_name

1. desc_name is the pathname or reference name of the new default library descriptor. If a reference name is given, the descriptor is searched for according to th search rules, which are documented in Section III, see Reference Names, of the MPM Reference Guide.

Key: pathname, pn

The pathname key returns the pathname of the library root(s) which are identified by one or more library names. library_descriptor may be invoked as an active function when the pathname key is used.

Usage

library_descriptor pathname library_names {-control_args}

where:

1. library_names are the names of the libraries whose pathnames are to be returned. The Multics star convention may be used to identify a group of libraries. Up to 30 library names may be given.

2. control_args are selected from the following list of control arguments and can appear anywhere after the key in the command:

-descriptor desc_name
gives the pathname or reference name of the library descriptor defining the library roots whose pathnames are to be returned. If the -descriptor control argument is not specified, then the default library descriptor is used.

-library library_name
-lb library_name
identifies a library name which begins with a minus (-) to distinguish the library name from a control argument. There are no other differences between the library names described above and those given with the -library control argument. One or more -library control arguments may be given in the command.

Key: default, dft

The default key prints the default library name(s) and search name(s) associated with one or more of the library descriptor commands.

Usage

library_descriptor default {command_names} {-control_arg}

where:

1. command_names are the names of the library descriptor commands whose default library and search names are to be printed. If no command names are given, the defaults for all of the library descriptor commands are printed.

2. control_arg may be the -descriptor control argument as described above. It may appear anywhere after the key in the command.

Key: root, rt

The root key prints detailed information about library roots on the user's terminal. The information includes the names on each library root, its pathname, and its type.

Usage

library_descriptor root library_names {-control_args}

where:

1. library_names identify the library roots about which information is to be printed. The Multics star convention may be used to identify a group of libraries. Up to 30 library names may be given.

2. control_args are selected from the following list of control arguments and can appear anywhere after the key in the command:

   -name, -nm causes all of the names on each library root to be printed.

   -primary, -pri causes the primary name on each library root to be printed.

   -match causes all library root names which match any of the library names to be printed. This is the default.

   -descriptor desc_name
   is as above.

   -library library_name
   -lb library_name
   is as above.

Names: library_descriptor_compiler, ldc

This command compiles a library description to produce a library descriptor data segment.

Refer to "Library Description Language" in Section V for a discussion of the syntax and semantics of the library description language.

Usage

library_descriptor_compiler desc_name {-control_arg}

where:

1. desc_name — is the relative pathname of the segment containing the library description to be compiled. If this pathname does not end with an ld suffix, then one is assumed.

2. control_arg — may be either of the following control arguments:

   -brief, -bf — indicates that the brief form of error messages is to be used for all errors diagnosed during the compilation. (See "Notes" below.)

   -long, -lg — indicates that the long form of error messages is to be used for all errors diagnosed during the compilation. (See "Notes" below.)

Notes

If the segment being compiled is called descriptor_name.ld, then the compilation generates a segment called descriptor_name.alm in the working directory. This segment can be assembled by the alm command to produce the library descriptor data segment.

If neither the -brief nor -long control argument is used, then the long form of error messages is used for the first occurrence of an error, and the brief form is used for subsequent occurrences of that error.

Names: library_fetch, lf

This command copies entries from a library into the user's working directory. Control arguments allow copying the entries into another directory or renaming them as they are copied; select which library entrynames are placed on the copy; allow copying the library entry which contains a matching entry instead of the matching entry itself (e.g., copy the archive which contains a matching archive component); or copying all of the components of the containing entry. A documentation facility is provided for recording in a file the status of each entry which is copied.

This command uses a library descriptor and library search procedures, as described in Section IV.

Usage

library_fetch {search_names} {-control_args}

where:

1. search_names are entrynames which identify the library entries to be copied. The Multics star convention may be used to identify a group of entries with a single search name. Up to 100 search names may be given in the command. If none are given, then any default search names specified in the library descriptor are used.

2. control_args are selected from the following list of control arguments and can appear anywhere in the command:

-library library_name,
-lb library_name
identifies a library which is to be searched for entries matching the search names. The Multics star convention may be used to identify a group of libraries to be searched. Up to 100 -library control arguments may be given in each command. If none are given, then any default library names specified in the library descriptor are used.

-name, -nm indicates that all of the names on each matching library entry are to be placed on the copy. See the discussion of naming considerations under "Notes" below.

-primary, -pri indicates that the first name of each matching library entry is to be placed on the copy. See the discussion of naming considerations under "Notes" below.

-match indicates that, for each matching library entry, the entrynames which match any of the search names are to be placed on the copy. See the discussion of naming considerations under "Notes" below. This is the default.

-into path identifies the directory into which library entries are copied and indicates how they are renamed. An absolute or relative pathname may be given. The directory portion of the pathname identifies the directory into which each library entry is copied. The final entryname of the pathname is used to rename each library entryname being placed on the copy, under control of the Multics equal convention. Only one -into control argument may appear in a command line. If -into is not given, matching entries are copied into the user's working directory and no renaming occurs.

-chase indicates that the target of a matching library link is to be copied.

-no_chase indicates that a warning message is to be printed when a matching link is found in the library, and that no copying is to occur. This is the default.

-long, -lg causes the pathname of each matching entry to be printed on the user's terminal as the entry is copied.

-brief, -bf suppresses printing the pathname of matching entries. This is the default.

-container causes the library entry which contains each matching entry to be copied, instead of the matching entry itself. See the discussion under "Notes" below.

-components causes all of the component library entries of a matching library entry to be copied, rather than just the matching entry itself. It also causes all components of a library entry containing a matching component to be copied. See the discussion under "Notes" below.

-entry, -et causes each matching library entry itself to be copied. This is the default.

-search_name search_name,
identifies a search name which begins with a minus (-) to distinguish the search name from a control argument. There are no other differences between the search names described above and those given with the -search_name control argument. One or more -search_name control arguments may be given in the command.

-descriptor desc_name
gives a pathname or reference name which identifies the library descriptor describing the libraries to be searched. If no -descriptor control argument is given, then the default library descriptor is used.

-retain, -ret indicates that library entries which are awaiting deletion from the library (as determined by the library search program) are to be copied.

-omit indicates that library entries awaiting deletion from the library are to be omitted from the search, and are not to be copied. This is the default.

-output_file file,
-of file indicates that status information for each copied library entry is to be appended to a file. A relative or absolute pathname of the file may be given. If it does not have a suffix of fetch, then one is assumed.

-all, -a indicates that all available status information for copied library entries is to be recorded in the output file.

-default, -dft indicates that only default status information is to be recorded in the output file. This is the default.

Notes

Any combination of the control arguments governing naming (-name, -primary, and -match) may be given in the command. However, the following groups of control arguments are mutually exclusive, and only one argument from each group may be given in the command: -chase and -no_chase; -long and -brief; -container, -components, and -entry; -retain and -omit; and -all and -default.

An -all or -default control argument may only be specified when the -output_file control argument is also given. The particular status information recorded in the output file for the -default control argument is under the control of the library search program. It includes the information deemed most important for the type of entry contained in the library.

If the file given in the -output_file control argument does not exist, it is created by library_fetch. If it does exist, new status information is appended to the end of the file preserving any previously recorded status. This feature allows the user to build a history of the entries copied out of a library.

When using the -into control argument, care must be taken to ensure that the equal name included in the -into pathname can be applied to all names to be placed on each of the copied entries. Name duplications can easily result when more than one library entry matches the search names.

The -container and -components control arguments are provided to facilitate copying all of the library entries included in a given bound segment or related to a given subsystem. For example, by identifying a component of the source archive for a bound segment and using the -container control argument, the entire source archive is copied into the user's directory. Similarly, by identifying a directory in the library containing all of the component entries of a subsystem and using the -components control argument, each component is copied into the user's directory.

When the -container, -components, or -chase control arguments are used, it may happen that none of the entrynames on a copied library entry matches any of the search names. Because the user may have requested that only matching names be placed on the copies, the library search program causes the first entryname

to be placed on the copy when one of these three control arguments is used, in addition to any names requested by the user.

The user is automatically given re access to object segments which are copied, r access to peruse_text object segments, and rw access to all other segments.

Examples

```
library_fetch abbrev.pl1 -into >udd>Multics>user>new_=.=
```

copies the source segment abbrev.pl1 into the directory >udd>Multics>user, renaming the copy new_abbrev.pl1.

```
library_fetch bound_runoff_.** -library online
```

copies all of the segments in the online libraries whose names begin with bound_runoff_ into the user's working directory. This might include the source archive, bindable object archive, bound object segment, and bind listing.

```
lf bound_runoff_.** -library online.source -components
```

copies all of the source components from the source archive for bound_runoff_ into the user's working directory.

```
lf qedx.pl1 -components
```

copies all of the source components in the archive containing qedx.pl1 into the user's working directory.

```
library_fetch *.alm -lb network.source -into new_=.alm
```

copies all ALM source segments from the network source library into the user's working directory, and adds a new_ prefix to the names placed on each segment.

```
library_fetch pl1_status.info -nm -lb info
```

copies the pl1_status.info segment from the info segment libraries into the user's working directory, copying all entrynames from the library entry onto the copy.

```
library_fetch **.ec -library online.??????
```

copies all exec_com segments from the online source and object libraries into the user's working directory.

```
library_fetch -lb supervisor.bc bound_sss_wired_.*
```

copies the bind segment from the bindable object archive called bound_sss_wired_.archive. Note that although the object archive itself matches the search name which was given, only the matching archive component is copied because the -container control argument was not given.

```
library_fetch -lb include stack_frame.incl.*
```

copies the stack frame declaration include segments for all source languages from the include library into the user's working directory.

Names: library_info, li

This command selects entries from a library, and prints the status of these entries on the user's terminal. The entries are printed in alphabetical order by primary name.

A full range of status information can be included in the output by using one or more of the output arguments. Besides information returned by the status command, the output can include access information, object segment attributes and other segment contents information, quota information, etc.

This command uses a library descriptor and library search procedures, as described in Section IV. When no output arguments are given, the information included by default is controlled by the search program for the particular library being searched. The default output includes the information most appropriate for library maintenance.

Usage

library_info {search_names} {-control_args} {lm_output_args}

where:

1. search_names are entrynames which identify the library entries to be output. The Multics star convention may be used to identify a group of entries with a single search name. Up to 100 search names may be given in the command. If none are given, then any default search names specified in the library descriptor are used.

2. control_args are selected from the following list of control arguments and can appear anywhere in the command:

-library library_name,
-lb library_name
identifies a library which is to be searched for entries matching the search names. The Multics star convention may be used to identify a group of libraries with a single library name. Up to 100 -library control arguments may be given in each command. If none are given, then any default library names specified in the library descriptor are used.

-components causes status information for all the components of a matching library entry, in addition to the output for the matching entry. It also causes status information for all components of a library entry containing a matching entry. See the discussion under "Notes" below.

-container causes status information for the library entry which contains each matching entry, in addition to the output for the matching entry. See the discussion under "Notes" below.

-entry, -et causes status information to be printed for only the library entries which match one of the search names. This is the default.

-chase suppresses status information for any intermediate links which exist between a library link and its eventual target.

-no_chase causes status information for the intermediate links. This is the default.

-retain, -ret causes status information for library entries awaiting deletion from the libraries (as determined by the library search program).

-omit suppresses status information for library entries awaiting deletion from the libraries. This is the default.

-search_name search_name
identifies a search name which begins with a minus (-) to distinguish the search name from a control argument. There are no other differences between the search names described above and those given with the -search_name control argument. One or more -search_name control arguments may be given in the command.

-descriptor desc_name
gives a pathname or reference name which identifies the library descriptor describing the libraries to be searched. If no -descriptor control argument is given, then the default library descriptor is used.

3. lm_output_args control which status information is included in the output. Any of the output arguments accepted by the library_map command (described later in this section) may be used for library_info as well. The output arguments can appear anywhere in the command.

Notes

Any combination of output arguments may be used in a command since the use of several output arguments merely causes more information to be included in the output. However, the following groups of control arguments are mutually exclusive, and only one argument from each group may be given in a command: -chase and -no_chase; -retain and -omit.

The -container and -components control arguments are provided to facilitate information gathering on all library entries related to a given bound segment. When only one component of a bound segment archive is matched, -entry causes status information to be printed for only the matching library entry; -container and -components cause status for related library entries as well. -container and -components may be used singly or together, but neither can be used with -entry.

The following example illustrates the effect of using -container and -components. If a search name is given which matches a component in a source archive, giving -entry would produce status for only that component. Giving -container instead would produce status for the source archive, as well as for the matching component. Giving -components would produce status for all of the components of the source archive containing the matching component. Giving both -container and -components would produce status for the source archive and all of its components.

Examples

```
library_info abbrev.* -lb source
```

returns information about the source segment for the abbrev procedure.

```
library_info  bound_apl_.**.archive -lb unb.s -container
-components
```

returns status for both of the APL source archives (bound_apl_.1.s.archive and bound_apl_.2.s.archive), and for all of their components.

```
library_info listen_ -lb supervisor.bndc -contents
```

returns information about the compilation and object attributes of the listen_ procedure. Refer to the description of output arguments in the library_map command for information about the -contents control argument.

Names: library_map, lm

This command selects entries from a library, and writes the status of these entries into a map file suitable for dprinting. The entries in the file are alphabetized by primary name.

A full range of status information can be included in the map items by using one or more of the output arguments. Besides information returned by the status command, the map items can include access information, object segment attributes and other segment contents information, quota information, etc.

This command uses a library descriptor and library search procedures, as described in Section IV. When no output arguments are given, the information included by default in the map items is controlled by the search program for the particular library being mapped. The default map item includes the information most appropriate for a library map.

Usage

library_map {search_names} {-control_args} {output_args}

where:

1. search_names are entrynames which identify the library entries to be output. The Multics star convention may be used to identify a group of entries with a single search name. Up to 100 search names may be given in the command. If none are given, then any default search names specified in the library descriptor are used.

2. control_args are selected from the following list of control arguments and can appear anywhere in the command:

-library library_name,
-lb library_name
identifies a library which is to be searched for entries matching the search names. The Multics star convention may be used to identify a group of libraries with a single library name. Up to 100 -library control arguments may be given in each command. If none are given, then any default library names specified in the library descriptor are used.

-output_file file,
-of file identifies the output file in which the library map is to be generated. A relative or absolute pathname may be given for the file. If it does not have a suffix of map, then one is assumed. If no -output_file control argument is given, then the map is generated in the library.map file which is created is the user's working directory.

-header heading,
-he heading gives a character string which is used as a heading line on the first page of the map to identify which libraries have been mapped. If the string contains blanks, then it must be enclosed in quotes. Only the first 120 characters of the string are used. If no -header control argument is given, then a default heading line is used. See the discussion under "Notes" below.

-footer footing
-fo footing gives a character string which is used in the footing line at the bottom of each page to identify the libraries being mapped. If the string contains blanks, then it must be enclosed in quotes. Only the first 45 characters of the string are used. If no -footer control argument is given, then a default character string is used in the footing line. See the discussion under "Notes" below.

-entry, -et causes map items to be included in the output only for library entries which match one of the search names.

-components causes map items for all the components of a matching library entry, in addition to the item for the matching entry. It also causes map items for all components of a library entry containing a matching entry. See the discussion under "Notes" below.

-container causes a map item for the library entry which contains each matching entry, in addition to the item for the matching entry. See the discussion under "Notes" below. This is the default.

-cross_reference, -cref
causes cross reference map items to be included in the output for the secondary names on library entries which are output. See the discussion under "Notes" below. This is the default.

-no_cross_reference, -ncref
suppresses cross reference map items.

-chase suppresses map items for any intermediate links which exist between a library link and its eventual target.

-no_chase causes map items for the intermediate links. This is the default.

-retain, -ret causes a map item for library entries awaiting deletion from the libraries (as determined by the library search program).

-omit suppresses the map item for library entries awaiting deletion from the libraries. This is the default.

-search_name search_name
identifies a search name which begins with a minus (-) to distinguish the search name from a control argument. There are no other differences between the search names described above and those given with the -search_name control argument. One or more -search_name control arguments may be given in the command.

-descriptor desc_name
gives a pathname or reference name which identifies the library descriptor describing the libraries to be searched. If no -descriptor control argument is given, then the default library descriptor is used.

3. output_args are selected from the following list of output arguments and can appear anywhere in the command:

-all, -a causes all available information to be output.

-default, -dft causes default information to be output, in addition to the information requested by other output arguments. This is the default when no other output arguments are given.

-status, -st causes all status information printed by the command "status -all" to be output, except for access control information.

-access causes all access control information to be output. This includes: the user's access mode to the library entry, its ring brackets, ACL, access class, AIM attributes, safety switch setting, and for directory entries the initial ACLs.

-contents causes information describing the contents of library entries to be output. This includes: compilation information, object attributes, and segment printability information.

The following output arguments are available, but are probably not of interest to every user. They provide more selective control over which status information is included in the output.

-name, -nm causes all names to be output.

-primary, -pri causes the primary name to be output.

-match causes all names which match any of the search names to be output.

-type, -tp causes the type of each library entry to be output. Types include: link, segment, archive, archive component, multisegment file, multisegment file component, and directory.

-pathname, -pn causes the pathname of the parent of each library entry to be output.

-link_target causes the pathname of the target of each library link to be output.

-date, -dt causes the date/time contents modified, date/time used, date/time entry modified, date/time dumped, and date/time compiled to be output.

-date_time_contents_modified, -dtcm
causes the date/time modified to be output.

-date_time_used, -dtu
causes the date/time used to be output.

-date_time_entry_modified, -dtem
causes the date/time entry modified to be output. For archive components, this corresponds to the date/time component updated into the archive.

-date_time_dumped, -dtd
causes the date/time dumped to be output.

-date_time_compiled, -dtc
causes the date/time compiled to be output.

-length, -ln causes the records used, current length (if different from the records used), maximum length (if different from sys_info$max_seg_size), bit count, archive component offset, and directory quota information to be output.

-records, -rec causes the records used to be output.

-current_length
causes the current length to be output (if different from records used).

-max_length causes the maximum length to be output (if different from sys_info$max_seg_size).

-bit_count causes the bit count to be output.

-offset, -ofs causes the word offset of an archive component within its archive to be output.

-quota causes directory quota information to be output for library directory entries. This includes: quota set on the directory, quota used, terminal quota switch setting (if on), a count of inferior directories with terminal quota (if nonzero), the time/record product for the directory, and the date/time time/record product updated. If a directory is a master directory, this information is also printed.

-author, -at causes the author and bit count author (if different from the author) to be output.

-unique_id causes the unique identifier to be output.

-device, -dv causes the name of the logical volume on which the entry resides to be output, for non-directory, non-MSF entries; causes the name of the son's logical volume to be output for directory and MSF entries. Also causes the setting of the transparent-to-paging-device switch to be output.

-copy, -cp causes the setting of the copy-on-write switch to be output (if on).

-safety causes the setting of the safety switch to be output (if on).

-mode, -md causes the user's mode of access to the library entry to be output.

-ring_brackets, -rb
causes the ring brackets to be output.

-acl causes the access control list to be output.

-access_class causes the access class to be output (if other than system low). Also, the setting of the security-out-of-service switch, the audit switch, and the multiple access class switch is output (if on).

-initial_acl causes the initial access control lists associated with library directory entries to be output.

-compiler_name causes the name of the compiler of an object segment to be output.

-compiler_version causes the version information for the compiler of an object segment to be output.

-compiler_options causes the compiler option information stored in an object segment to be output.

-object_info causes information about format of an object segment and its entry bound to be output.

-non_ascii causes an indication that a library entry contains non-ASCII characters to be output.

-error causes messages indicating errors which occurred while obtaining the status information to be output.

-level causes a level number to precede each output entry. This number indicates the relationship between a library entry and its components. Normally, this relationship is indicated only by indenting the component names beneath those of the library entry.

-new_line, -nl causes a line to be skipped between each level 1 entry in the output. Normally, no lines are skipped between entries.

Notes

Any combination of output arguments may be used in a command since the use of several output arguments merely causes more information to be included in each map entry. However, the following groups of control arguments are mutually exclusive, and only one argument from each group may be given in a command: -cross_reference and -no_cross_reference; -chase and -no_chase; -retain and -omit.

The -container and -components control arguments are provided to facilitate the mapping of library entries related to a given bound segment. When only one component of a bound segment archive matches one of the search names, -entry causes a map item for only the matching library entry; -container and -components cause map items for entries related to a matching entry as well. -container and -components may be used singly or

together, but neither can be used with -entry.

The following example illustrates the effect of using -container and -components. If a search name is given which matches a component in a source archive, giving -entry would produce a map item for only that component. Giving -container instead would produce a map item for the source archive, as well as one for the matching component. Giving -components would produce map items for all of the components of the source archive containing the matching component. Giving both -container and -components would produce map items for the source archive and all of its components.

When the -cross_reference control argument is used, a cross reference map item is included in the map for each secondary name on a matching library entry. The cross reference item includes: the secondary name; the date/time modified for the library entry; and its pathname. The pathname ends with the primary name of the library entry, providing a reference to the map item which includes complete information about the entry.

The library map is generated in an output file identified by the -output_file control argument. If the -output_file control argument is not given, then a file called library.map is created in the user's working directory. If the output file already exists, it is truncated before the new map is created. Thus several library_map commands executed in the same working directory (in the same or different processes) without an -output_file control argument can produce unpredictable results. In such cases, the -output_file control argument should be used to create a different map file in each command.

If the -header control argument is given, then the heading line is centered on the first page of the map beneath the lines:

Map of the nn Entries

of the

The heading line should be worded with this in mind. For example:

Map of the 35 Entries

of the

Standard Library Bind Listing Directory

If -header is not given, a default heading line is constructed by concatenating the names of the libraries which were searched, as shown below:

Map of the 350 Entries

of the

Libraries

standard_library.list, unbundled_library.list,
tools_library.list, user_library.list, network_library.list

If the -footer control argument is given, then the footing line placed at the bottom of each page of the library map contains the footing character string given with the control argument, along with a page number, and the names of the first and last map items which appear on the page. If -footer is not given, then the concatenated library names used in the heading line are also used in the footing line.

Examples

library_map -lb info -lb mpm **.info **.runoff -of documentation

creates the documentation.map file in the working directory, which contains a map of the entries in the info and MPM libraries which match the search names **.info or **.runoff.

library_map -lb online.* ** -of online -dtd -dft

creates the online.map file which contains a map of all of the entries in the online.* libraries. Each map entry includes the date dumped, as well as whatever default information was specified by the library search program.

library_map

creates a map in the library.map file of the working directory which contains map items for those entries in the default library (or libraries) which match the default search name(s). These default values are specified in the default library descriptor data base.

Names: library_print, lpr

This command selects library entries whose contents is printable, and writes the contents of these entries into a file suitable for dprinting. Printable library entries are those which contain only ASCII characters. The ASCII portion of peruse text object segments is also printable. Thus, library_print can print source segments, listings, bind segments, info segments, peruse text object segments, exec_com and absentee control segments, printable multisegment files, etc.

The entries in the print file are alphabetized by the primary name on the library entry. Each entry is preceded by a header which lists the status of the entry. An index of all entry names appears at the end of the print file.

This command uses a library descriptor and library search procedures, as described in Section IV. When no output arguments are given, the status information included by default in each entry's heading is controlled by the search program for the particular library being printed. The default heading includes information most appropriate for library maintenance.

Usage

library_print {search_names} {-control_args} {lm_output_args}

where:

1. search_names are entrynames which identify the library entries whose contents is to be output. The Multics star convention may be used to identify a group of entries with a single search name. Up to 100 search names may be given in the command. If none are given, then any default search names specified in the library descriptor are used.

2. control_args are selected from the following list of control arguments and can appear anywhere in the command:

-library library_name,
-lb library_name
identifies a library which is to be searched for entries matching the search names. The Multics star convention may be used to identify a group of libraries with a single library name. Up to 100 -library control arguments may be given in each command. If none are given, then any default library names specified in the library descriptor are used.

-output_file file,
-of file
identifies the output file in which the printed contents is to be generated. A relative or absolute pathname may be given for the print file. If it does not have a suffix of print, then one is assumed. If no -output_file control argument is given, then the print file is generated in the library.print file which is created is the user's working directory.

-header heading,
-he heading
gives a character string which is used as a heading line on the first page of the print file to identify which libraries have been printed. If the string contains blanks, then it must be enclosed in quotes. Only the first 120 characters of the string are used. If no -header control argument is given, then a default heading line is used. See the discussion under "Notes" below.

-footer footing
-fo footing
gives a character string which is used in the footing line at the bottom of each page to identify the libraries being printed. If the string contains blanks, then it must be enclosed in quotes. Only the first 45 characters of the string are used. If no -footer control argument is given, then a default character string is used in the footing line. See the discussion under "Notes" below.

-components causes all the components of a matching library entry to be output, instead of the entry itself. It also causes all components of a library entry containing a matching entry to be output. See the discussion under "Notes" below.

-container causes the library entry which contains each matching entry to be output as a whole, rather than the matching entry. See the discussion under "Notes" below.

-entry, -et causes only the contents of library entries which match one of the search names to be output. This is the default.

-chase suppresses entry heading information for any intermediate links which exist between a library link and its eventual target whose contents is output.

-no_chase causes entry heading information for the intermediate links. This is the default.

-retain, -ret causes library entries awaiting deletion from the libraries (as determined by the library search program) to be output.

-omit suppresses library entries awaiting deletion from the libraries. This is the default.

-search_name search_name
identifies a search name which begins with a minus (-) to distinguish the search name from a control argument. There are no other differences between the search names described above and those given with the -search_name control argument. One or more -search_name control arguments may be given in the command.

-descriptor desc_name
gives a pathname or reference name which identifies the library descriptor describing the libraries to be searched. If no -descriptor control argument is given, then the default library descriptor is used.

3. lm_output_args control which information is included in the entry headings. Any of the output arguments accepted by the library_map command may be used for library_print as well. The output arguments can appear anywhere in the command.

Notes

Any combination of output arguments may be used in a command since the use of several output arguments merely causes more information to be included in the heading for each entry. However, the following groups are control arguments are mutually exclusive, and only one argument from each group may be given in a command: -components, -container, and -entry; -chase and -no_chase; -retain and -omit.

The -container and -components control arguments are provided to facilitate the printing of library entries related to a given bound segment. When only one component of an archive is matched, -entry causes only the matching library entry to be output; -container and -components cause the other components of the archive to be output as well. -container causes the entire archive to be output as a whole, rather than just the matching component. -components causes all of the archive components to be output, rather than just the matching component.

The print file is generated in an output file identified by the -output_file control argument. If the -output_file control argument is not given, then a file called library.print is created in the user's working directory. If the output file already exists, it is truncated before the new print file is created. Thus, several library_print commands executed in the same working directory (in the same or different processes) without an -output_file control argument can produce unpredictable results. In such cases, the -output_file control argument should be used to create a different print file in each command.

If the -header control argument is given, then the heading line is centered on the first page of the print file beneath the lines:

Print Out of the nn Entries

of the

The heading line should be worded with this in mind. For example:

Print Out of the 35 Entries

of the

Standard Library Bind Listing Directory

If -header is not given, a default heading line is constructed by concatenating the names of the libraries which were searched, as shown below:

Print Out of the 350 Entries

of the

Libraries

standard_library.list, unbundled_library.list,
tools_library.list, user_library.list, network_library.list

If the -footer control argument is given, then then footing line placed at the bottom of each page of the print file contains the footing character string given with the control argument, along with a page number and the name of the entry being output. If -footer is not given, then the concatenated library names used in the heading line are also used in the footing line.

Examples

library_print -lb info -lb mpm **.info **.runout -of documentation

creates the documentation.print file in the working directory, which contains a print out of the entries in the info and mpm libraries which match the search names **.info or **.runout.

```
library_print -lb online.object **.bind -of online -dtd -dft
```

creates the online.print file which contains a print out of all of the bind control segments in the online object libraries. Each entry includes a header with the date dumped, as well as whatever default status information was specified by the library search program.

```
library_print
```

creates a print out in the library.print file of the working directory which contains the contents of those entries in the default library (or libraries) which match the default search name(s). These default values are specified by the default library descriptor data base.

Name: map355

This command is used to assemble a program written in the FNP assembler language, map355. The command does not assemble the program directly. Instead, it prepares a GCOS job deck to perform the assembly and calls the GCOS Environment Simulator to do the work.

This command is fully described in the MAM Communications manual, Order No. CC75.

Names: update_seg, us

This command is used to define the contents of a modification, and to install or de-install the modification in one or more libraries.

A modification is a group of physically- or logically-related segments which must be installed in a library at the same time in order to maintain library consistency and integrity. For example, a source segment and its compiled object segment are physically-related segments which must be installed concurrently to ensure that library source segments correspond to library object segments. On the other hand, two object segments which interact with one another are logically-related segments which must be installed concurrently to ensure proper operation.

The update_seg command is the library maintainer's interface to the Multics Installation System (MIS). MIS installs the related segments of a modification into a library at the same time (or nearly so):

1. by dividing the installation of each segment into a series of steps (getting the unique id, names, and ACL of new and old segments, copying the target segment, adding to and deleting from the target segment's names, freeing names on the old segment, etc).
2. by performing one step for all segments of the modification before moving on to the next step.
3. by installing the segments which are used by library users (e.g., object segments) as a group after installing the other segments in the modification (e.g., source segments, archives, and info segments).

Using this strategy, the installation window (the period of library inconsistency) can be reduced to less than one minute per modification, and is usually about five seconds per modification.

MIS offers several benefits to the library maintainer. The MIS subroutines which perform each installation step are all restartable. If a system failure or a process failure occurs during an installation, the installation can be resumed from the point of interruption, as long as the Multics Storage System remains intact across the failure.

The MIS subroutines are also reversible. Each MIS subroutine performs a specific installation function when invoked in "installation" mode with a group of arguments. The same MIS subroutine will perform the logical inverse of its installation function (a de-installation function) when it is invoked in "de-installation" mode with the same group of arguments. If a

bad modification has been installed, it can be removed from the libraries by invoking MIS in "de-installation" mode, without the use of supplementary tools or special procedures.

MIS provides planned automatic error recovery. If MIS detects a fatal installation error, it can recover automatically from the error by invoking, in "de-installation" mode, the installation subroutines which completed before the fatal error occurred. Most common installation errors (name duplication, entry not found, record quota overflow, etc) are handled in this manner.

MIS allows a limited degree of rerunnability. All MIS subroutines are rerunnable after having been invoked in "de-installation" mode, as long as the segments in the modification have not been changed since the de-installation. The installer can correct many minor errors (e.g., name duplications) without having to start the installation from the very beginning.

Finally, MIS automatically documents an installation. An MIS subroutine creates a description of a modification, and appends this description to an ASCII installation log as a part of the installation. In addition, a paragraph summarizing the modification can be inserted at the top of an installations info segment to notify users of changes to the libraries.

update_seg stores the definition of a modification in an installation object (io) segment as a list of tasks. The task list consists of one or more task blocks, each representing a call to one of the MIS installation subroutines. The defined modification is installed by sorting these task blocks by type of installation step and calling the MIS subroutines associated with the order task blocks. The update_seg command interfaces with the MIS task list processor and installation subroutines to perform the definition and installation operations.

Usage

update_seg opname arguments

where:

1. opname — designates the operation to be performed.

2. arguments — may be one or more arguments, depending upon the particular operation to be performed.

The opnames permitted, followed by their alternate forms where applicable, are shown below in five functional groupings:

set_defaults, sd initiate, in print_defaults, pd	Creation operations
add delete, dl move, mv replace, rp	Definition operations
print, pr list, ls	Listing operations
install de_install	Installation operations
clear	Clearing operation

The creation operations create and initiate an installation object (io) segment in which a modification is defined. The definition operations define the segments of the modification, and the steps which must be performed to install those segments. The listing operations list the segments of the modification and the installation steps to be performed for those segments. The installation operations install and de-install the modification. Finally, the clearing operations reset an io segment when an installation has failed and the modification has been installed.

Usage is explained below under a separate heading for each designated operation. The explanations are arranged functionally, as shown above.

GENERIC CONTROL ARGUMENTS

The following control arguments are accepted by several update_seg operations. To avoid describing them with each of these operations, the control argument syntax is described here. The description of each operation includes a list of control arguments accepted by that operation, and it states how the operation is affected by each control argument.

1. -acl mode1 User_id1 ... moden -User_idn-
defines an access control list (ACL) by pairing each access mode with the access control name which follows,

where:

a. modei is a valid access mode for segments. It may be any or all of the letters rew to indicate read, execute, and write access respectively. Use null, "n", or "" to specify null access.

b. User_idi is an access control name that must be of the form Person_id.Project_id.tag. Missing components in the access control name are assumed to be "*". If the last modei has no User_idi following it, the library maintainer's Person_id and current Project_id are assumed.

2. -add_name names
-an names defines a list of names to be added to the target segment of a definition operation, where names are one or more entrynames.

3. -archive, -ac specifies that the segment being defined in a definition operation is an archive, and that the names of its archive components are to be added to the target segment of the definition operation. Normally the archive component names are not added to the target segment.

4. -defer, -df specifies that the installation subroutines which gather information about the segments in a definition operation should defer their information gathering until the installation operation is performed. Thus, changes made to the segment after the modification is defined will be reflected in the installed segment. Normally, name and ACL changes made between the definition and installation operations are not reflected in the installed target segment. Segment replacements during this period cause a fatal installation error.

5. -delete_acl User_ids
-da User_ids defines a list of ACL entries which are to be removed from the ACL of the target segment of a definition operation, where User_ids are as defined for -acl above.

6. -delete_name names
-dn names defines a list of names to be removed from the target segment of a definition operation, where names are one or more entrynames.

7. -max_length -N-
-ml -N- specifies that the maximum length attribute of the target segment of a definition operation is to be set as shown below. Normally, the maximum length is set to sys_info$default_max_length for regular segments, and to the current length of the segment being installed for special segments. See -special_seg below for information about special segments.

N > 0 the maximum length is set of N words.

N = 0 the maximum length is set to the current length of the segment being installed.

N omitted the maximum length is set to sys_info$max_seg_size.

8. -name names
   -nm names
   defines the list of names to be placed on the target segment of a definition operation, where names are one or more entrynames.

9. -ring_brackets r1 -r2- -r3-
   -rb r1 -r2- -r3-
   defines a set of ring brackets,

where:

a. r1 is the write bracket.

b. r2 is the read bracket. If omitted, it is set to the maximum of the following values: the write bracket, the current validation level, or 5.

c. r3 is the gate bracket. It may not be specified unless r2 is also specified. If omitted, it is set to the maximum of the following values: the read bracket, the current validation level, or 5.

10. -set_acl mode1 User_id1 ... moden -User_idn-
    -sa mode1 User_id1 ... moden -User_idn-
    defines a list of ACL entries which are to be added to the ACL of the target segment of a definition, where modei and User_idi are as defined for -acl above.

11. -set_log_dir path
    -sld path
    defines the directory identified by path as the directory containing the installation log and installation info segments. These segments are described further under "Automatic Documentation" below.

12. -special_seg
    -ss
    defines the target segment of a definition operation as a special segment. The properties of special segments are described below under "Special Segments".

Ring Brackets

The ring brackets given in a -ring_brackets control argument control the intraprocess use of the segments being installed. A description of ring brackets and intraprocess access control can be found in the MPM Reference Guide.

Automatic Documentation

The directory defined in a -set_log_dir control argument is called the documentation directory. Two types of information about a modification are logged in segments contained in this directory.

1. A summary of the modification is inserted at the beginning of Installations.info. This is a segment designed to inform users of recent changes to the libraries.

2. Detailed information about which segments and bound segment components are changed by the modification is appended to Installations.log, along with the summary described above. This log contains a permanent record of all installations.

These documentation segments are multiplexed among several different update_seg installers by using a lock word in the segment Installations.lock.

Special Segments

The -special_seg control argument is used to reduce the installation window for the user-visible segments of a modification. For example, if a modification contains two bound segments, one of which calls the other, then it is important to reduce the time between the installation of the first segment and the installation of the second. Otherwise, users of the first segment could receive errors when it tried to reference the second.

To reduce the length of user-observable installation windows, the segments of a modification being installed in user search directories can be defined as special segments which have the following properties:

1. The final installation of all special segments (adding names to these segments) is deferred until all regular segments have been installed.

2. The de-installation of a modification causes the regular segments which were installed to be deleted from the library. Special segments are renamed instead of being deleted.

3. The default setting for the maximum length attribute of segments differs for special segments from that used for regular segments. Special segments use the current length of the segment being installed as the default maximum length. Regular segments use sys_info$default_max_length.

Deferring the final installation of special segments until the last possible moment provides several desirable advantages. If a fatal error occurs while installing a regular segment, no special segments will have been installed and the user-visible portions of the libraries will remain in a consistent state. In addition, the installation window for special segments is shortened by grouping them together at the end of the installation, because there are fewer segments going through the final installation step (adding names) at the same time. This further reduces the user's expose to library inconsistencies.

Special segments cannot be deleted by a de-installation operation because some users may being using them. However, renaming the special segments prevents more users from using them after they have been de-installed.

Operation: initiate

This operation is the first operation required to install a modification. It creates a new installation object (io) segment and initiates it for use by update_seg.

Only one io segment can be initiated in a process at any given time. This restriction allows the library maintainer to omit the io segment name from most update_seg operations. When the io segment name is omitted, then the operation refers to the io segment which is currently initiated. This is usually the io segment segment named in the last initiate operation.

Besides creating new io segments, the initiate operation can be used to switch to and initiate another existing io segment, or to change the attributes of an existing io segment.

Usage

update_seg initiate {io_seg} {-control_args}

where:

1. io_seg is the pathname of the io segment to be initiated. If the final entryname does not have an io suffix, then one is assumed. If io_seg is omitted, then the attributes of the currently-initiated io segment are changed.

2. control_args are selected from the following list of optional control arguments:

   -restart, -rt indicates that the io segment identifed by io_seg exists and is to be reinitiated. Normally a new io segment is created when io_seg is given.

   -acl mode1 User_id1 ... moden -User_idn-
   defines the default ACL used by the initiated io segment. This ACL is placed on new segments being added to a library when no -acl control argument is given in an add definition operation. Normally, the global default ACL is used as the default ACL on a new io segment.

-ring_brackets r1 -r2- -r3-
-rb r1 -r2- -r3-
defines the default ring brackets used by the initiated io segment. These ring brackets are placed on new segments being added to a library when no -ring_brackets control argument is given in an add definition operation. Normally, the global default ring brackets are used as the default ring brackets on a new io segment.

-set_log_dir path
-sld path
gives the pathname of the documentation directory to be used by the io segment. Normally the global documentation directory is used.

-log
indicates that a summary of the modification is to be typed in as part of the initiate operation. This summary is placed in one or more documentation segments, as described under "Automatic Documentation" above. Normally, no summary is associated with a new io segment.

Notes

The global default ACL, ring brackets, and documentation directory have the values shown in Table 13-1 below.

Table 13-1. Initial Values for update_seg Global Defaults

ACL:	re *.*.*
ring brackets:	1,5,5
documentation directory:	working directory

These values may be changed for the life of the library maintainer's process by using the set_defaults operation. The current global defaults may be printed by using the print_defaults operation.

When the -log control argument is given, the initiate operation responds by printing "Input". All subsequent lines typed by the library maintainer are used as a summary of the modification being defined in the io segment. Input of the summary ends when the library maintainer types a line containing only a period (.). The summary is placed in both of the documentation segments when the modification is installed.

The summary lines are truncated or filled out to 65 characters to improve the readability of the documentation segments. A completely blank line or a line beginning with a space or horizontal tab (HT) character will force a break in the filling of the previous line.

The summary of a modification can be changed at any point before the modification is installed (before an installation operation). Reinitiating the io segment with the -log control argument causes any previously-defined summary to be replaced by a new summary.

The summary associated with any io segment can be printed as described below under the print operation.

Operation: print_defaults

This operation prints the global default ACL, ring brackets, and documentation directory. It also prints the default values associated with an io segment. The default documentation directory is printed only if different from the working directory.

Usage

update_seg print_defaults {io_seg}

where io_seg is an optional argument which specifies the pathname of an existing io segment whose defaults are to be printed. If the final entryname does not have an io suffix, then one is assumed.

Notes

If an io_seg argument is given with the print_defaults operation, the named io segment is reinitiated, and remains initiated after the defaults have been printed. Thus, all further update_seg operations will refer to this initiated io segment.

If no io_seg argument is given, then the defaults of the initiated io segment are printed if one is initiated.

Operation: set_defaults

This operation sets the global default ACL, ring brackets, and documentation directory.

Usage

```
update_seg set_defaults  {-control_args}
```

where:

1. control_args are selected from the following list of control arguments:

   -acl mode1 User_id1 ... moden -User_idn-
   defines a new global default ACL.

   -ring_brackets r1 -r2- -r3-
   -rb r1 -r2- -r3-
   defines a new set of global default ring brackets.

   -set_log_dir path
   -sld path
   defines a new global default documentation directory.

Notes

If none of the control arguments listed above are specified, then the corresponding global default value remains unchanged.

Operation: add

This operation defines a segment which is to be added to a library as part of a modification. The definition is appended to the currently-initiated io segment. The following installation steps are required for the most common case of the add operation.

1. Get the unique id of the new segment.
2. Get the names on the new segment.
3. Gather detailed information about the new segment for documentation of the installation.
4. Create a uniquely-named target segment in the library, and copy the contents of the new segment into the target segment.
5. Set the ring brackets on the target segment.
6. Set the ACL on the target segment.
7. Add the new segment's names to the uniquely-named target segment.
8. Remove the unique name from the target segment.
9. Document the addition of the new segment to the library.

Usage

update_seg add new_seg target_seg {-control_args}

where:

1. new_seg is the pathname of the new segment to be added to the library. A relative or absolute pathname may be given.

2. target_seg is the pathname of the target segment which is to be created in the library directory. A relative or absolute pathname may be given, and the Multics equal convention may be used to equate components in the final entrynames in the new_seg and target_seg pathnames. Note that an error will occur if the final entryname of the target_seg pathname is not one of the names placed on the target segment as it is installed.

3. control_args are selected from the following list of optional control arguments:

-acl mode1 User_id1 ... moden -User_idn-
defines the ACL to be placed on the target segment. Normally the default ACL is used.

-add_name names
-an names
defines a set of names to be added to the target segment.

-archive, -ac
specifies that the new segment is an archive whose components names are to be added to the target segment.

-defer, -df
specifies that the information gathering in steps 1-3 above is to be deferred until the installation operation.

-delete_acl User_ids
-da User_ids
defines ACL entries to be removed from the target segment.

-delete_name names
-dn names
defines a set of names to be removed from the target segment.

-log
specifies that detailed information about the installation of the new segment is to be logged in Installations.log.

-max_length -N-
-ml -N-
defines the maximum length attribute setting for the target segment. Normally the default setting is used.

-name names
-nm names
defines the names to be placed on the target segment. Normally, the names on the new segment are placed on the target segment.

-ring_brackets r1 -r2- -r3-
-rb r1 -r2- -r3-
defines the ring brackets to be placed on the target segment. Normally, the default ring brackets are placed on the target segment.

-set_acl mode1 User_id1 ... moden -User_idn-
-sa mode1 User_id1 ... moden -User_idn-
defines ACL entries to be added to the ACL on the target segment.

-special_seg
-ss
defines the target segment to be a special segment.

Operation: delete

This operation defines a segment which is to be deleted from a library as part of a modification. The definition is appended to the currently-initiated io segment. The following steps are required for the most common case of the delete operation.

1. Get the unique id of the segment to be deleted (the target segment).
2. Get the names on the target segment.
3. Gather detailed information about the segment being deleted for documentation of the installation.
4. Add a unique name to the target segment.
5. Free the names on the target segment.
6. Document the deletion of the target segment from the library.

At the time of the installation operation, the segment is not actually deleted from the library. Instead, the segment's names are freed and a unique name is added to the segment to mark it as a candidate for deletion by the library_cleanup command at some later date. The segment cannot be deleted because it cannot be terminated in the process of any library user who might be using it.

The segment's names are freed: by adding an integer suffix to the primary segment name, as described in the lfree_name command description in Section XIII; and by deleting any other names on the segment. The renamed primary name is retained to identify the segment. The remaining names are deleted to prevent library users from referencing the segment.

Usage

update_seg delete target_seg {-control_args}

where:

1. target_seg — is the pathname of the segment to be deleted from the library. A relative or absolute pathname may be given.

2. control_args — are selected from the following list of optional control arguments:

   -defer, -df — specifies that the information gathering in steps 1-3 above is to be deferred until the installation operation.

-log specifies that detailed information about the deletion of the segment is to be logged in Installations.log.

-special_seg
-ss defines the target segment to be a special segment.

Operation: replace

This operation defines a segment which is to replace another segment in a library as part of a modification. The definition is appended to the currently-initiated io segment. The following steps are required for the most common case of the replace operation.

1. Get the unique id of the segment to be replaced (the old segment).
2. Get the names on the old segment.
3. Get the ACL on the old segment.
4. Get the ring brackets on the old segment.
5. Get the unique id of the segment to replace the old segment (the new segment).
6. Get the names on the new segment.
7. Gather detailed information about the new segment for documentation of the installation.
8. Create a uniquely-named target segment in the library, and copy the contents of the new segment into this target segment.
9. Set the ring brackets on the target segment to those on the old segment.
10. Set the ACL on the target segment to that on the old segment.
11. Add a unique name to the old segment.
12. Free the names on the old segment.
13. Add the new segment's names to the target segment.
14. Remove the unique name from the target segment.
15. Document the replacement of the library segment.

Usage

update_seg replace new_seg old_seg {target_seg} {-control_args}

where:

1. new_seg — is the pathname of the new segment. A relative or absolute pathname can be given.

2. old_seg — is the pathname of the library segment which is to be replaced. A relative or absolute pathname may be given, and the Multics equal convention may be used to equate components in the final entrynames of the new_seg and old_seg pathnames. Note that, if the target_seg argument is omitted, an error will occur if the final entryname of the old_seg

pathname is not one of the names placed on the target segment as it is installed.

3. target_seg is the optional pathname of the target segment, if this differs from the pathname of the old segment. A relative or absolute pathname may be given, and the Multics equal convention may be used to equate components in the final entrynames of the target_seg and old_seg pathnames. Normally the pathname of the old segment is formed by using the directory portion of old_seg and the final entryname portion of new_seg (i.e., [directory old_seg]>[entry new_seg]). Note that an error will occur if the final entryname of the target_seg pathname is not one of the names placed on the target segment as it is installed.

4. control_args are selected from the following list of optional control arguments:

-acl mode1 User_id1 ... moden -User_idn-
defines the ACL to be placed on the target segment. Normally, the ACL on the old segment is placed on the target segment.

-add_name names
-an names
defines a set of names to be added to the target segment.

-archive, -ac specifies that the new segment is an archive whose component names are to be added to the target segment.

-defer, -df specifies that the information gathering in steps 1-7 above is to be deferred until the installation operation.

-delete_acl User_ids
-da User_ids
defines ACL entries to be removed from the target segment.

-delete_name names
-dn names
defines a set of names to be removed from the target segment.

-log specifies that detailed information about the installation of the replacement library segment is to be documented in

Installations.log.

-max_length -N-
-ml -N-
defines the maximum length attribute setting for the target segment. Normally the default setting is used.

-name names
-nm names
defines the names to be placed on the target segment. Normally, the names on the new segment are placed on the target segment.

-old_name
-onm
specifies that the names on the old segment are to be placed on the target segment. Normally, the names on the new segment are placed on the target segment.

-ring_brackets r1 -r2- -r3-
-rb r1 -r2- -r3-
defines the ring brackets to be placed on the target segment. Normally, the ring brackets on the old segment are placed on the target segment.

-set_acl mode1 User_id1 ... moden -User_idn-
-sa mode1 User_id1 ... moden -User_idn-
defines ACL entries to be added to the ACL on the target segment.

-special_seg
-ss
defines the target segment to be a special segment.

Notes

The -name and -old_name control arguments are mutually exclusive. If both are given in a replace operation, then the last one given is used.

Just as in a delete operation, the old segment in a replace operation in not deleted at the time of the installation operation. Instead, the old segment's names are freed, and a unique name is placed on the segment to mark it as a candidate for deletion by the library_cleanup command at a later date.

Operation: move

This operation defines a segment which is a library segment to be moved to another library directory as part of a modification. The definition is appended to the currently-initiated io segment. The following steps are required for the most common case of the move operation.

1. Get the unique id of the segment to be moved (the old segment).
2. Get the names on the old segment.
3. Get the ACL on the old segment.
4. Get the ring brackets on the old segment.
5. Gather detailed information about the old segment for documentation of the installation.
6. Create a uniquely-named target segment in the other library, and copy the contents of the old segment into this target segment.
7. Set the ring brackets on the target segment to those on the old segment.
8. Set the ACL on the target segment to that on the old segment.
9. Add a unique name to the old segment.
10. Free the names on the old segment.
11. Add the old segment's names to the target segment.
12. Remove the unique name from the target segment.
13. Document the movement of the library segment.

Usage

```
update_seg move  old_seg target_seg  {-control_args}
```

where:

1. old_seg — is the pathname of the segment to be moved. A relative or absolute pathname may be given.

2. target_seg — is the pathname of the segment into which the old segment is moved. A relative or absolute pathname may be given, and the Multics equal convention may be used to equate components in the final entrynames of the old_seg and target_seg pathnames. Note that an error will occur if the final entryname of the target_seg pathname is not one of the names placed on the target segment as it is installed.

3. control_args are selected from the following list of optional control arguments:

-acl mode1 User_id1 ... moden -User_idn-
defines the ACL to be placed on the target segment. Normally, the ACL on the old segment is placed on the target segment.

-add_name names
-an names
defines a set of names to be added to the target segment.

-archive, -ac
specifies that the old segment is an archive whose component names are to be added to the target segment.

-defer, -df
specifies that the information gathering in steps 1-5 above is to be deferred until the installation operation.

-delete_acl User_ids
-da User_ids
defines ACL entries to be removed from the target segment.

-delete_name names
-dn names
defines a set of names to be removed from the target segment.

-log
specifies that detailed information about the movement of the library segment is to be documented in Installations.log.

-max_length -N-
-ml -N-
defines the maximum length attribute setting for the target segment. Normally the default setting is used.

-name names
-nm names
defines the names to be placed on the target segment. Normally, the names on the old segment are placed on the target segment.

-ring_brackets r1 -r2- -r3-
-rb r1 -r2- -r3-
defines the ring brackets to be placed on the target segment. Normally, the ring brackets on the old segment are placed on the target segment.

-set_acl mode1 User_id1 ... moden -User_idn-
-sa mode1 User_id1 ... moden -User_idn-
defines ACL entries to be added to the ACL on the target segment.

-special_seg
-ss
defines the target segment to be a special segment.

Note

Just as in a delete operation, the old segment in a move operation is not deleted at the time of the installation operation. Instead, the old segment's names are freed, and a unique name is placed on the segment to mark it as a candidate for deletion by the library_cleanup command at a later date.

Operation: print

This operation prints information on the terminal about the modification defined in an io segment. One set of information is included for each segment of the modification. This information normally includes the following items.

1. The type of definition operation (add, delete, replace, or move), and the pathnames of the target segment, old segment, and/or new segment given in the definition of each modification segment.

2. A list of the control arguments given in the definition operation.

3. The names, ACL, and ring brackets to be placed on the target segment.

4. The detailed information about the modification segment to be included in Installations.log when the -log control argument was given in the definition operation.

Usage

update_seg print {io_seg} {-control_args}

where:

1. io_seg — is an optional argument which specifies the pathname of an existing io segment whose modification is to be printed. If the final entryname does not have an io suffix, then one is assumed. See "Notes" below for a discussion of this argument.

2. control_args — are selected from the following list of optional control arguments:

   -log — specifies that only the summary of the modification provided when the io segment was initiated is to be printed.

   -brief, -bf — suppresses information items 2-4 given above from the printed output.

   -long, -lg — adds a list of the installation steps required to install each modification segment to the printed information.

-error, -er suppresses information about all modification segments except those which encountered an error during the most recent attempt to install or de-install the modification. The printed information includes the installation step in which the error occurred, and an error message describing the error.

Notes

If an io_seg argument is given with the print operation, the named io segment is reinitiated, and it remains initiated after the modification information has been printed. Thus, all further update_seg operations will refer to this initiated io segment.

If no io_seg argument is given, then the modification information for the initiated io segment is printed, if one is initiated.

The -brief and -long control arguments can be used together to suppress information items 2-4 given in the list above while including a list of installation steps. Similarly, the -long and -error control arguments can be used together to print all of the installation steps, rather than just those in which an error occurred.

Operation: list

This operation creates an installation listing segment containing information about an io segment. The listing segment is created in the working directory. If the listed io segment is named io_seg.io, then its listing segment is named io_seg.il. The installation listing normally contains the following information items.

1. The pathname of the io segment.
2. The date and time at which the installation listing was created.
3. The access identifier of the process which created the io segment.
4. The version of update_seg used to create the io segment.
5. The date and time at which the last creation, definition, or listing operation was performed on the io segment.
6. If the modification has been installed, the access identifier of the process which installed the modification, and the date and time of installation.
7. If the modification has been de-installed, the access identifier of the process which de-installed the modification, and the date and time of de-installation.
8. If the io segment has been cleared (see the clear operation below), the access identifier of the process which cleared the io segment, and the date and time of clearing.
9. The summary of the modification, if one was defined when the io segment was initiated.
10. A list of the definition operations performed on the io segment. This list includes the pathnames of the target segment, old segment, and/or new segment for each definition operation.
11. If the modification has been installed, a list of any errors which occurred during the installation.
12. A description of the modification which includes the following information for each modification segment:
    a. The type of definition operation (add, delete, replace, or move), and the pathnames of the target segment, old segment, and/or new segment given in the definition of each modification segment.
    b. A list of control arguments given in the definition operation.
    c. The names, ACL, and ring brackets to be placed on the target segment.
    d. The detailed information about each modification segment used to document the installation.

Usage

```
update_seg list  {io_seg}  {-control_args}
```

where:

1. io_seg — is an optional argument which specifies the pathname of an existing io segment which is to be listed. If the final entryname does not have an io suffix, then one is assumed. See "Notes" below for a discussion of this argument.

2. control_args — are selected from the following list of optional control arguments:

   -brief, -bf — suppresses item 12 above from the listing.

   -long, -lg — appends to the listing a detailed description of the modification which includes the installation steps required to install each modification segment.

Notes

If an io_seg argument is given with the list operation, the named io segment is reinitiated, and it remains initiated after the io segment has been listed. Thus, all further update_seg operations will refer to this initiated io segment.

If no io_seg argument is given, then the currently-initiated io segment is listed, if one is initiated.

The -brief and -long control arguments can be used together to provide a detailed description of the modification without the regular description outlined in item 12 above. The detailed description includes all of the information in the regular description. However because of the length of the detailed description, it is often useful to have both the shorter regular description as a quick reference, and the longer detailed description for an installation step reference.

Operation: install

This operation installs the modification defined in an io segment.

Usage

update_seg install {io_seg} {-control_args}

where:

1. io_seg — is an optional argument which specifies the pathname of an existing io segment defining the modification to be installed. If the final entryname does not have an io suffix, then one is assumed. See "Notes" below for a discussion of this argument.

2. control_args — are selected from the following list of optional control arguments:

   -severity N
   -sv N — defines the severity level of fatal installation errors. All errors whose severity is equal to or greater than N are treated as fatal errors. N must be an integer from 1 to 5 inclusive. The default severity is 1, making all installation errors fatal. Refer to "Controlling the Fatality of Installation Errors" below for a description of the severity levels associated with the various kinds of installation errors.

   -stop — disables the automatic error recovery mechanism, causing update_seg to stop when a fatal installation error occurs. Refer to "Installation Errors" below for more information.

Notes

If an io_seg argument is given with the install operation, the named io segment is reinitiated, and it remains initiated after the modification has been installed. Thus, all further update_seg operations refer to this initiated io segment.

If no io_seg argument is given, then the modification defined in the currently-initiated io segment is installed, if one is initiated.

Any error codes which were set during a prior installation operation are automatically cleared before beginning the installation. This ensures that all errors which may be reported pertain to the current installation operation.

The Multics Installation System calls entries in the installation_tools_ gate in order to install Multics System Library segments into ring 1. Maintainers of outer ring libraries do not have access to this privileged gate. They can use update_seg to install segments by initiating the hcs_ gate with the reference name installation_tools_ once per process before using the install operation. The following command will perform this function:

```
initiate [get_pathname hcs_] installation_tools_
```

Installation Errors

If an error occurs during the installation of a modification, a message is printed to diagnose the error. Two types of errors may occur: nonfatal errors, and fatal errors. The diagnostic message for a nonfatal error begins with a Warning caption, while that for a fatal error begins with an Error caption.

Nonfatal errors do not stop the installation, but are merely diagnosed as they occur so that the library maintainer can take corrective action after the installation is complete.

Fatal errors have a more serious impact on the installation. Normally, the occurrence of a fatal error stops the installation of the modification, and automatically de-installs all portions of the modification which were installed prior to the error. When the -stop control argument is given, the occurrence of an error merely stops the installation.

The -stop control argument should be used with care because stopping the installation of a modification will probably leave the target library in an inconsistent state. When -stop is used, the library maintainer must recover from any installation errors as quickly as possible to reduce this period of inconsistency. Because the normal error recovery procedures minimize the period of library inconsistency and are so fast and easy to use, the -stop control argument is not recommended for general use.

Controlling the Fatality of Installation Errors

A given installation error may be nonfatal or fatal, depending upon the severity level associated with that error, and upon the fatal severity level given in the -severity control argument.

The Multics Installation System defines four severity levels, numbered 1 through 4. One of these severity levels is assigned to each possible installation error, depending upon how severely that error impacts the installation. Errors with a high severity level impact an installation more severely than those with a lower severity. The errors which fall into each severity level are described in Table 13-2 below.

Table 13-2. Severity of update_seg Installation Errors

SEVERITY	TYPES OF ERRORS
1	Errors incurred while: adding a name which is already on the target segment; deleting a name or ACL entry which is not on the target segment; or processing an archive with more than 100 components.
2	Errors incurred while: adding an invalid ACL entry to the target segment; adding a name which is already on another entry in the target directory; setting the target segment's bit count, maximum length attribute, or safety switch; deleting the final name from the target segment; and freeing the names on the old segment.
3	All other errors except for record quota overflows.
4	Record quota overflow errors.

The library maintainer must determine which severity levels shown in Table 13-2 above contain errors fatal to the installation being performed. The maintainer should then set the fatal severity level for the installation by using the -severity control argument.

Determination of the fatal severity level greatly depends on the type of modification being installed. A severity 1 error occurring during the modification of a heavily-used user search directory could have severe consequences for the library users,

and would probably warrant a fatal severity level of 1. On the other hand, a severity 1 error occurring during the installation of new information segments into a documentation directory would have less impact on library users, since no users would be depending upon the new segments. A fatal severity level of 2 might be appropriate in this case.

Low fatal severity levels are recommended for general use. The automatic error recovery for fatal errors is very fast, and subsequent reinstallation of the modification is easy do to. High fatal severity levels should be used only in unusual circumstances and with extreme caution.

Correcting Fatal Installation Errors

If a fatal installation error occurs, the normal error recovery procedure automatically de-installs all portions of the modification installed before the error occurred. The library maintainer can follow one of the two strategies below to correct the error:

1. If the error did not involve the definition of the modification (the contents, attributes or pathnames of any of the segments in the modification), then the library maintainer can correct the cause of the error and reinstall the modification using the install operation. An example of such an error is a record quota overflow in the target directory, or a name duplication between the target segment and an existing library entry.

2. If the error did involve the definition of the modification, then the library maintainer must redefine the modification correctly in a new io segment and reinstall the modification using the install operation. An example of such an error is an attempt to place the wrong name on a target segment causing a name duplication, or a -delete_name control argument attempting to delete the final name from a target segment.

If the -stop control argument was given with the install operation, then the installation is stopped if a fatal error occurs without de-installing whatever portions of the modification were installed prior to the error. The library maintainer can follow one of three strategies to correct the error:

1. If the error did not involve the modification definition, the library maintainer can correct the error and continue the installation by using the install operation.

2. If the error did not involve the modification definition, the library maintainer can use the de_install operation to de-install the modification until the error is corrected, and then use the install operation to reinstall the modification.

3. If the error did involve the modification definition, the library maintainer must use the de_install operation to de-install the modification, must then redefine the modification correctly in a new io segment and install that segment using the install operation.

Recovering From a Crash

If the system should crash during an installation, or a fatal process error occur during an installation, then the installation of the modification can be continued by using the install operation. Alternately, parts of the modification installed prior to the crash can be de-installed by using the de_install operation.

While it is usually safe to attempt to de-install a modification after a system crash, the de-installation will probably fail if the crash has affected any Multics storage system directory referenced as part of the modification. If such a failure occurs, it is necessary to complete the de-installation manually by using the lfree_name, lset_ring_brackets, lsetacl, ldelete, ldeletename, and lrename commands.

Operation: de_install

This operation de-installs the modification defined in an io segment. The modification must have been previously installed, either completely or partially.

Usage

update_seg de_install {io_seg} {-control_args}

where:

1. io_seg — is an optional argument which specifies the pathname of an existing io segment defining the modification to be de-installed. If the final entryname does not have an io suffix, then one is assumed. See "Notes" below for a discussion of this argument.

2. control_args — are selected from the following list of optional control arguments:

-severity N
-sv N — defines the severity level of fatal de-installation errors. All errors whose severity is equal to or greater than N are treated as fatal errors. N must be an integer from 1 to 5 inclusive. The default severity is 1, making all de-installation errors fatal. Refer to "Controlling the Fatality of De-Installation Errors" below for a description of the severity levels associated with the various kinds of de-installation errors.

-stop — disables the automatic error recovery mechanism, causing update_seg to stop when a fatal de-installation error occurs. Refer to "De-Installation Errors" below for more information.

Notes

If an io_seg argument is given with the de_install operation, the named io segment is reinitiated, and it remains initiated after the modification has been de-installed. Thus, all further update_seg operations refer to this initiated io segment.

If no io_seg argument is given, then the modification defined in the currently-initiated io segment is de-installed, if one is initiated.

As with the install operation, the de_install operation uses the installation_tools_ gate to de-install segments from ring 1. Maintainers of outer ring libraries should issue the command:

```
initiate [get_pathname hcs_] installation_tools_
```

once per process before using the de_install operation.

De-Installation Errors

If an error occurs during the de-installation of a modification, a message is printed to diagnose the error. As with installation errors, the message for a de-installation error has a Warning caption for a nonfatal error or an Error caption for a fatal error.

A nonfatal error does not stop the de-installation. A fatal error stops the de-installation and automatically reinstalls the modification. When the -stop control argument is given, a fatal error stops the de-installation without reinstalling it.

Controlling the Fatality of De-Installation Errors

A given de-installation error may be nonfatal or fatal, depending upon the severity level associated with that error, and upon the fatal severity level given by the library maintainer in the -severity control argument. The errors which fall into each severity level are described in Table 13-3 below.

Table 13-3. Severity of update_seg De-Installation Errors

SEVERITY	TYPES OF ERRORS
1	Errors incurred while: restoring a name which is already on an old segment; removing a name which is not on the target segment; or deleting the target segment.
2	Errors incurred while: restoring a name on an old segment which is already on another entry in the old segment's directory; removing the final name from the target segment; or resetting the ACL or ring brackets on the target segment.
3	All other errors.

Correcting Fatal De-Installation Errors

Fatal de-installation errors usually occur because the segments in the target directory (or their attributes) have been changed since the modification was installed. Such modifications could occur: if a subsequent modification affected one or more of the segments of the modification; if the Multics storage system was reloaded, causing a new unique identifier to be assigned to each segment; if a system crash forced the target directory to be salvaged; etc.

The proper corrective action for most de-installation errors involves returning the segments in the modification to their state just after installation. In some cases, this may be as simple as de-installing a subsequent modification. In other cases, returning to the installation state may be undesirable. For example, the de-installation of a subsequent modification could restore a module with a serious bug. It might be better to replace all bad modules with fixed versions if these are available, rather than restoring to modules with worse bugs.

In some cases, returning to the installation state may be impossible. It would be very difficult to restore the unique identifiers for segments in a reloaded directory. The update_seg clear -uid operation is provided to disable unique identifier checking by update_seg in such cases. However, it must be used with extreme caution. Without this checking, other segments besides those in the modification may be affected by the de-installation.

Finally, it may not be possible to restore segments in a salvaged directory to their original state. In such cases, it may be necessary to use -severity 4 in the de-install operation to de-install other portions of the installation, and then to de-install portions in the salvaged directory manually. Care must be taken in such operations, because the library will be inconsistent until both the automatic and manual de-installation operations are complete. Having such a large de-installation window may necessitate performing the de-installation during a special session of Multics when users are not allowed to log on.

Recovering from a Crash

As with an install operation, a de-installation interrupted by a system crash can be restarted by using the de_install operation, or can be reversed by using the install operation.

Operation: clear

This operation clears all error codes set during a prior installation or de-installation operation. This allows the io segment to be printed or listed prior to a reinstallation of the modification without having prior error messages appear in the output.

The clear operation also clears segment unique identifiers stored in the modification definition. These unique identifiers are stored to ensure that the segments defined in modification definition operations are those which are actually installed or de-installed. Clearing these identifiers disables such checking, and allows the de-installation of a modification whose segment unique identifiers have been reset by a Multics storage system reload.

NOTE: Extreme care should be taken when clearing segment unique identifiers to ensure that the correct segments will be de-installed by the subsequent de_install operation.

Usage

update_seg clear {io_seg} {-control_args}

where:

1. io_seg — is an optional argument which specifies the pathname of an existing io segment which is to be cleared. If the final entryname does not have an io suffix, then one is assumed. See "Notes" below for a discussion of this argument.

2. control_args — must be one or both of the control arguments listed below:

   -error, -er — specifies that error codes stored in the modification definition are to be reset.

   -uid — specifies that unique identifiers for the segments in the modification are to be reset, thus disabling unique identifier checking during subsequent installation and de-installation operations.

Notes

If an io_seg argument is given with the clear operation, the named io segment is reinitiated, and it remains initiated after the io segment has been cleared. Thus, all further update_seg operations refer to this initiated io segment.

If no io_seg argument is given, then the currently-initiated io segment is cleared, if one is initiated.

Examples

The following are typical examples of terminal sessions using update_seg to modify segments. Brief explanations of each command line typed by the user are given below each example.

Example 1

```
1  update_seg initiate >udd>Multics>example1
   r 1521  .613  11.729  153

2  update_seg add >udd>Multics>seg1 >sss>seg1
   r 1521  .188  4.161  61

3  update_seg replace >udd>Multics>seg2 >sss>seg2
   r 1521  .220  4.188  37

4  update_seg delete >sss>seg3
   r 1521  .109  3.038  61

5  update_seg move >sss>seg4 >unbundled>seg4
   r 1521  .199  2.394  31

6  update_seg install
   Installation beginning.
   Installation complete.
   r 1521  2.009  19.592  304

7  update_seg list
   r 1522  .610  9.418  98

8  dprint example1.il
   1 request signalled, 0 already queued.
   r 1522  .088  1.142  35
```

line 1: Create and initiate an io segment called example1.io in the directory >udd>Multics. Use global default values for the default ACL and ring brackets.

line 2: Define segment >udd>Multics>seg1 as a modification segment to be added to the >sss directory. Put the default ACL and ring brackets on this segment.

line 3: Define segment >udd>Multics>seg2 as a modification segment which is to replace segment >sss>seg2. Put the old segment's ACL and ring brackets on the target segment.

line 4: Define segment >sss>seg3 as a modification segment which is to be deleted.

line 5: Define segment >sss>seg4 as a modification segment which is to be moved to the directory >unbundled.

line 6: Install the modification defined in the initiated io segment (>udd>Multics>example1.io).

line 7: Create a listing segment which describes the modification, and includes any installation errors. The segment is called example1.il, and is created in the working directory.

line 8: Dprint the listing.

Example 2

```
1  us initiate example2 -acl re User.Multics -rb 4 5 5
   r 1523  .536  5.210  104

2  us add >udd>Multics>seg5 >sss>seg5
   r 1523  .185  3.554  62

3  us add seg6 >sss>seg6 -acl re *.Multics -rb 1 5 5
   r 1523  .126  1.106  25

4  us replace seg7 >sss>== -acl re User.Multics n * -ss -log
   r 1523  .375  4.834  88

5  us install
   Installation beginning.
   Copying special target segments.
   Adding names to special target segments.
   Installation complete.
   r 1523  2.098  19.673  344
```

line 1: Initiate example2.io, setting the default ACL to re User.Multics and the default ring brackets to 4,5,5.

line 2: Define >udd>Multics>seg5 as a modification segment to be added to the directory >sss. The default ACL and ring brackets will be placed on this segment.

line 3: Define segment seg6 in the working directory as a modification segment to be added to the directory >sss. Put ACL of re *.Multics.* and ring brackets 1,5,5 on the target segment.

line 4: Define seg7 in the working directory as a modification segment which is to replace >sss>seg7. Put an ACL of re User.Multics.* and null *.*.* on the target segment. Put the ring brackets of >sss>seg7 on the target segment. Also treat the target segment as a special segment and log the modification of this segment in Installations.log.

line 5: Install the modification defined in example2.io.

Example 3

```
1  update_seg initiate >udd>Multics>example1 -restart
   r 1536  .486  6.066  110

2  update_seg de_install
   De-installation beginning.
   Non-special target segments deleted.
   De-installation complete.
   r 1536  1.504  9.525  174

3  update_seg de_install example2
   De-installation beginning.
   Names removed from special target segments.
   Non-special target segments deleted.
   De-installation complete.
   r 1537  1.092  11.523  169

4  update_seg list -long
   r 1538  .610  9.418  98
```

line 1: Reinitiate >udd>Multics>example1.io, the io segment created in Example 1 above.
line 2: De-install the modification defined in this io segment.
line 3: Reinitiate example2.io, an io segment created in the working directory as part of Example 2. De-install the modification defined in this io segment.
line 4: Create a listing segment, example2.il, in the working directory which describes the modification defined in example2.io.

Example 4

```
1  us initiate library -log
   Input.
2  MCR 128:  Fix bug in the delete command (bound_fscom1_)
3  which prevented segments whose copy switch is on from
4  being deleted.
5  .
   r 1547  1.600  8.856  169

6  us rp bound_fscom1_.s.archive >ldd>sss>s== -archive
   r 1547  1.750  11.898  222

7  us rp bound_fscom1_.archive >ldd>sss>o>== -archive
   r 1547  .310  6.534  55

8  us rp bound_fscom1_.list >ldd>sss>lists>bound_fscom1_.list
   r 1547  .239  3.822  38

9  us rp bound_fscom1_ >sss>== -ss -log -rb 1 5 5 -acl re *
   r 1548  .729  9.802  165

10 us install
   Installation beginning.
   Copying special target segments.
   Adding names to special target segments.
   Installation complete.
   r 1548  5.759  32.104  566
```

line 1: Create and initiate a new io segment in the working directory, library.io. Use the -log control argument to add a summary of the modification to the io segment. update_seg responds by typing "Input".

line 2: Lines 2 through 5 typed by the library maintainer are the summary of the modification. This summary is inserted at the top of Installations.info, and appended to the end of Installations.log when the modification is installed.

line 6: Define bound_fscom1_.s.archive in the working directory as a modification segment which is to replace >ldd>sss>s>bound_fscom1_.s.archive. Add its archive component names to the target segment. Put the old segment's ACL and ring brackets on the target segment.

line 7: Define bound_fscom1_.archive in the working directory as a modification segment which is to replace >ldd>sss>o>bound_fscom1_.archive, and add its components names to the target segment.

line 8: Define bound_fscom1_.list as a replacement for >ldd>sss>lists>bound_fscom1_.list.

line 9: Define bound_fscom1_ as a replacement for >sss>bound_fscom1_, logging the replacement of this segment in Installations.log. Put ring brackets of 1,5,5 on the target segment, an ACL of re *.*.*, and treat the target segment as a special segment. The names on bound_fscom1_ are placed on the copy installed in >sss by default.
line 10: Install the modification defined above.

SECTION XIV

LIBRARY SUBROUTINES AND DATA BASES

This section contains the descriptions of important subroutines and data bases which are used to maintain the Multics Libraries.

Name: multics_libraries_

This data base is the library descriptor for the Multics System Libraries. Like all library descriptors, it defines: the roots of the Multics System Libraries; the names by which these roots can be referenced in library descriptor commands; and the default library names and search names used by each of the library descriptor commands when operating on the Multics System Libraries.

The general organization of libraries is described in Section II, "Library Organization", and the organization of the Multics System Libraries is described in Section III. The use of library descriptors is discussed further in Section IV, "The Library Descriptor Commands". The library description language which is used to define library descriptors is described in Section V, "Maintaining User Libraries with the Library Tools".

The Multics System Libraries

The Multics System is composed of the "logical libraries" listed in Table 14-1 below. Each of the libraries is, in turn, composed of several directories containing the different kinds of library entries (source and object segments; bind lists; info, include, and peruse_text segments; multisegment files) which are stored in the libraries. These directories are listed in Table 14-2 below. A library descriptor command can reference an entire logical library by name, or it can reference one or more of its directories.

Note that the logical library structure defined below does not map directly onto the physical library organization in the Multics storage system. However, the library descriptor tools can reference all of the physical libraries by logical library name.

Table 14-1. Logical Libraries of the Multics System

LIBRARY IDENTIFIER	LIBRARY CONTENTS
standard_library, std	most user commands and subroutines, and the system support routines for these commands and subroutines.
languages_library, lang	Multics PL/I and ALM translators and their support routines.
unbundled_library, unb	Honeywell program products and other unbundled software.
tools_library, tools	system administrative and maintenance commands and subroutines.
installation_library, inst	installation-maintained software.
user_library, user	user-provided software installed at the installation to facilitate sharing.
network_library, net	nonsupervisory software used in connecting the Multics System to the ARPA Network.
supervisor_library, sup, hardcore, hard, h	the supervisor (hardcore) software of the Multics System.
bootload_library, boot, bos	software for the Bootload Operating System (BOS).
communications_library, com, mcs	software for the Multics Communications System (MCS).

Each of the above logical libraries contains one or more of the following logical directories. A listing of which directories are part of which library is given in Table 14-4 below.

Table 14-2. Multics System Library Directories

DIRECTORY ID	DIRECTORY CONTENTS
source, s	the source language segments which can be translated into library object segments. The directory also contains exec_com segments which are used to create library object segments.
object, o	the object segments produced by translating the library source segments. The directory also contains exec_com and absentee input segments, and multisegment files intended for use by users.
lists, l	the listings produced by binding several library object segments together into bound segments.
execution, x	the bound and unbound object segments and data bases used by Multics users. The directory also contains exec_com segments and absentee input segments intended for use by users. The directory is generally included in the search rules of some or all users.
bound_comp, bndc, bc	the object archives which may be bound into bound segments.
info, i	information segments which can be printed on the user's terminal under control of the help command. These segments describe the commands and subroutines included in the library, and they outline library problems, command status, upcoming changes, etc.
include, incl	the source segments which are included as part of several other source segments, under control of a source language translator.

Library Names

One or more libraries or directories may be referenced in a library descriptor command by giving the appropriate combinations of library identifier and directory identifier as library names.

1. A particular library identifier from the table above can be used as a library name to reference all of the directories in that library.

2. A particular directory identifier from the table above can be used as a library name to reference all directories of a given type (e.g., all source directories, all object directories, etc).

3. A two-component library name of the form:

   library_identifier.directory_identifier

   can be used to reference a particular directory in a given library. For example, standard.source and lang.incl are such two-component library names.

4. A library name employing the star convention can be used to reference several libraries or directories. For example, *.?????? references all source and object directories, and ** references all library directories.

5. Two groups of libraries can be referenced by using the library identifiers shown in Table 14-3 below. These identifiers may be used separately or in combination with directory identifiers.

Table 14-3. Multics Library Groups

LIBRARY ID	GROUP OF LIBRARIES REFERENCED
online_libraries, online, on	standard_library, languages_library, unbundled_library, tools_library, installation_library, user_library, network_library.
offline_libraries, offline, off	supervisor_library, bootload_library, communications_library.

Not all of the libraries listed above contain each type of directory. Table 14-4 below shows which library/directory combinations are valid.

Table 14-4. Directories in Each Multics Library

std.source	lang.source	unb.source
std.object	lang.object	unb.object
std.lists	lang.lists	unb.lists
std.execution	lang.execution	unb.execution
std.info	lang.info	unb.info
std.include	lang.include	unb.include
	tools.source	
	tools.object	
	tools.lists	
	tools.execution	
	tools.info	
	tools.include	
inst.source	user.source	net.source
inst.object	user.object	net.object
inst.lists	user.lists	net.lists
inst.execution	user.execution	net.execution
inst.info	user.info	net.info
inst.include	user.include	net.include
sup.source	bos.source	com.source
sup.object	bos.object	com.object
sup.include	bos.include	
sup.bound_comp		

Some examples of library names are:

```
online_libraries
off.source
standard_library.info
include
user.x
network_library.lists
std.??????
```

Library Descriptor Command Defaults

Table 14-5 below shows the default library names and search names defined for each of the library descriptor commands.

Table 14-5. Library Descriptor Command Defaults for the Multics System Libraries

COMMAND	DEFAULT LIBRARY NAMES	DEFAULT SEARCH NAMES
library_cleanup	online	!??????????????
library_fetch	online.source, sup.source	(none)
library_info	online	(none)
library_map	online.source, online.object, online.lists, online.execution	**
library_print	info	*.**.info

Name: multics_library_search_

This subroutine is the library search procedure for the Multics System Libraries. Its entry points are referenced by multics_libraries_, the library descriptor for the Multics System Libraries. These entry points may also be used to search other libraries which have directories structured like those of the Multics System Libraries.

Section II discusses the general organization of libraries, and Section III describes the organization of the Multics System Libraries. Refer to the multics_libraries_ data base description in Section XIV for more information about this library descriptor.

The entry points described below conform to the calling sequence for library search procedures described in Section V under "Coding a Library Search Procedure". Therefore, the usage of these entry points is not repeated here.

Instead, each entry point description below discusses the types of directories which can be searched with the entry point. In addition, Table 14-6 compares the following attributes of each entry point.

1. The types of library entries supported by the entry point (links, segments, archives, and MSFs).

2. The kind of default output arguments used with each entry point (either online defaults or offline defaults). These default output arguments are defined in Table 14-7.

3. The kinds of checks made automatically. These can include checking for archives, checking for object segments, and checking the printability of a library entry. An entry is printable if it only contains ASCII characters, or if it is a peruse_text object segment.

4. The kinds of checks inhibited, even if requested by the user.

5. The special naming conventions which are applied. The search names given by the user may be mapped into different names which are actually used to search the directory. Alternately, some search entry points may require that the names of archive components be used as additional names on the archive to speed the search process. In some libraries, entries awaiting deletion from the library (obsolete entries) are marked with a

unique name (returned by the unique_chars_ subroutine). These entries are not found by a search unless the -retain control argument is given in a library descriptor command. Refer to the MPM Subroutines for a description of the unique_chars_ subroutine.

6. The use of a system identification data base by the search entry point.

Entry: multics_library_search_$online_source_dirs

This entry point searches directories organized like the Multics online source directories. These directories contain archived and unarchived source segments, and exec_com control segments which are used to create object segments. The names of all archive components must be placed as additional names on their respective archives.

Entry: multics_library_search_$online_object_dirs

This entry point searches directories organized like the Multics online object directories. These directories contain archived and unarchived object segments, backup copies of exec_com and absentee control segments intended for user usage, and backup copies of MSFs. The names of all archive components must be placed on their respective archives.

Entry: multics_library_search_$online_list_info_dirs

This entry point searches directories organized like the Multics online lists, info, and include directories. These directories contain printable segments.

Entry: multics_library_search_$online_execution_dirs

This entry point searches directories organized like the Multics online execution directories. These directories contain bound and unbound object segments, data bases, exec_com and absentee control segments, and MSFs used by users. Such directories are usually included in user search rules.

Entry: multics_library_search_$hardcore_source_dir

This entry point searches the Multics supervisor source directory. It is inappropriate for use on other libraries because it uses a specialized system identification data base.

Entry: multics_library_search_$hardcore_bc_dir

This entry point searches the Multics supervisor bound components directory. It is inappropriate for use on other libraries because it uses a specialized system identification data base.

Entry: multics_library_search_$hardcore_object_dir

This entry point searches the Multics supervisor object directory. It is inappropriate for use on other libraries because it uses a specialized system identification data base.

Entry: multics_library_search_$offline_source_dirs

This entry point searches the Multics BOS and MCS source directories. It is currently identical in function with the $online_source_dirs entry point.

Entry: multics_library_search_$offline_object_dirs

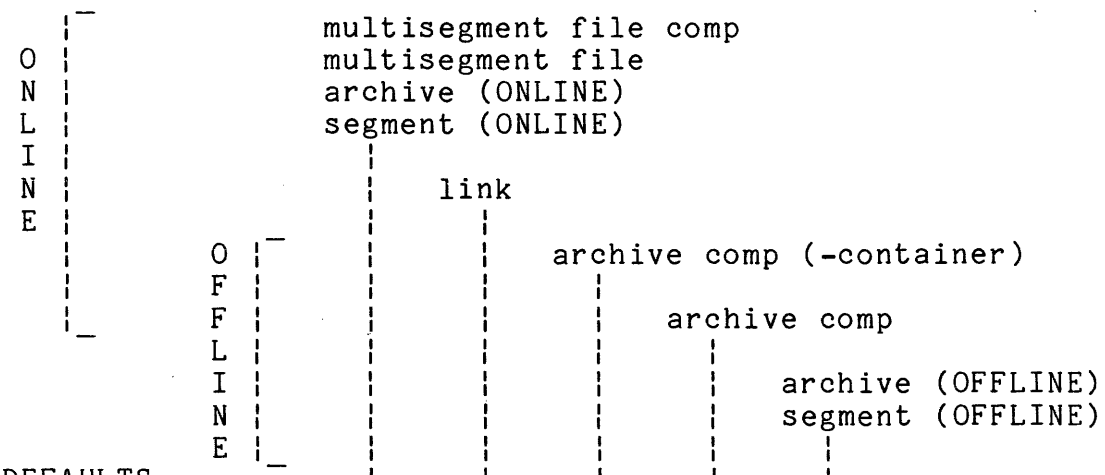
This entry point searches the Multics BOS and MCS object directories. It is currently identical in function with the $online_object_dirs entry point.

Table 14-6. Comparison of multics_library_search_ entry points

	online_source_dirs	online_object_dirs	online_list_info_dirs	online_execution_dirs	hardcore_source_dir	hardcore_bc_dir	hardcore_object_dir	offline_source_dirs	offline_object_dirs
ENTRIES SUPPORTED:									
links	X	X	X	X				X	X
segments	X	X	X	X			X	X	X
archives	X	X			X	X		X	X
multisegment files	X	X	X	X				X	X
OUTPUT DEFAULTS:									
online	X	X	X	X				X	X
offline					X	X	X		
AUTOMATIC CHECKS:									
archives	X	X			X	X		X	X
printability		2		2		2	2		2
object segments		1		1		1	1		1
INHIBITED CHECKS:									
archives			X	X			X		
printability	X		X	1	X		1	X	
object segments	X		X		X			X	
NAMING:									
search names					X				
archive names	X	X						X	X
obsolete entries	X	X	X	X				X	X
SYSTEM ID DATA:									
supervisor					X	X	X		

1. Checks made or inhibited only for the library_fetch command.
2. Automatic checks made only for the library_print command.

Table 14-7. Default Output Arguments for Online and Offline Search Procedures



OUTPUT DEFAULTS					
-name	MP	MP	IMP	FIMP	MP
-primary	CFIMP	CFIMP	FIMP	FIMP	FIMP
-match	CFIMP	CFIMP	FIMP	FIMP	FIMP
-type	CFIMP	CFIMP	F	FIMP	FIMP
-pathname	CFIMP	CFIMP	F	FIMP	FIMP
-link_target		CFIMP			
-dtcm	CFIMP		F		FIMP
-dtem	C MP	C IMP	FIMP	FIMP	MP
-dtd	M	M			
-dtc	F		F	F	F
-current_length	M				
-records	M				
-max_length	M				
-bit_count	F M		F	F	F
-copy	CF MP		F		F MP
-safety	C M				M
-ring_brackets	M				
-compiler_version	F		F	F	F
-compiler_options	F		F	F	F
-object_info	F		F	F	F
-level	C IMP	C IMP	IMP	IMP	IMP
-new_line	CFIMP	CFIMP	FIMP	FIMP	FIMP
-error	CFIMP	CFIMP	FIMP	FIMP	FIMP
-cross reference	MP		M		MP

C = library_cleanup F = library_fetch I = library_info
M = library_map P = library_print

APPENDIX A

PLANNED DOCUMENATION ADDITIONS

This section outlines the planned additions to this document (by major heading only) to provide the user with a general idea of what the completed PLM will consist of.

SECTION I — Introduction to Library Maintenance
- Maintenance and Modification Functions
- Survey of the Maintenance Tools

SECTION II — Library Organization
- The Logical Structure of Libraries
- Structure Definitions in the Library Descriptor
- Contents Definitions in the Search Procedure
- Using the Definitions in Library Tools
- Defining A Library Structure

SECTION III — The Multics System Libraries
- The Online and Offline Libraries
- The Online Libraries
- The Offline Libraries
- The Multics Library Descriptor

SECTION VI — Cross Reference Tools
- Cross Reference Functions
- Object Segment Cross Reference
- Include Segment Cross Reference

SECTION VII — Online Library Modification
- Overview of the Online Libraries
- Strategy For Library Modification
- Preparing a Modification
- Testing the Modification
- Installing the Modification
- Documenting the Modification
- Modification Tools

SECTION VIII Supervisor Library Modification

- Overview of the Supervisor
- Strategy For Library Modification
- Preparing a Modification
- Testing the Modification
- Installing the Modification
- Updating the Modification
- Documenting the Modification
- Modification Tools

SECTION X Bootload Library Modification

- Overview of the Bootload Operating System (BOS)
- Strategy for BOS Modification
- Preparing a Modification
- Testing the Modification
- Installing the Modification
- Updating the Modification
- Documenting the Modification
- Modification Tools

SECTION XI When The Systems Libraries Self-Destruct

- Problems Which Can Occur
- Special Tools for Correcting Problems

SECTION XII Library Maintenance Exec_Com's

- BOS exec_com's

SECTION XIII Library Tools

- check_mst, ckm
- compare_entry_names, cen
- cross_reference, cref
- edit_mst_header, emh
- gate macros
- generate_mst, gm
- include_cross_reference, icref
- laddname, lan
- ldelete, ldl
- ldeleteacl, lda
- ldeletename, ldn
- lnames
- lpatch
- lrename, lren
- lset_ring_brackets, lsrb
- lsetacl, lsa
- object_submission_test, ost
- setcopysw
- source_submission_test, sst
- sys_dates_
- translator_search_rules
- updater, upd

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE	SERIES 60(LEVEL 68) MULTICS LIBRARY MAINTENANCE PLM PRELIMINARY EDITION	**ORDER NO.**	AN80, REV. 0
		DATED	MAY 1979

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME _____ **DATE** _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

24014, 5C679, Printed in U.S.A.

AN80-00