To: Distribution

From: Gary C. Dixon

Date: November 6, 1974

Subject: New Library Tools: Design Principles

For the past several years, efforts have been underway to rewrite and generally clean up the code in the tools which are used to maintain the Multics System Libraries. A major phase of the effort ended with the installation of the update_seg command, which installs segments in the Online Libraries. Now a second major phase is coming to fruition. This MTB summarizes the work which was done as part of this second phase.

Since the work was begun before MTBs or the MCR board came into being, it has been proceeding without having an approved MCR. It is my intention now to hold a Design Review of the basic designs summarized below, and then to submit several MCRs requesting the installation of the new or modified library tools.

Goals of Phase Two

Phase Two of the clean up campaign addresses (at least) 7 different tools which are used in library maintenance. These are: msl_info; msl_global_format; msl_short_format; get_library_source; cleanup; icref; and cross_reference. Respectively, these tools: printed brief information about entries in the library on the user's terminal; generated detailed library status information in a segment; generated brief library status information in a segment; extracted source segments from the library; actually delete from the Online Libraries those segments which were replaced as part of an installation, but which could not be deleted at installation time because they might have been in use in someone's process; cross-reference the use of include segments by library entries; cross-reference the use of library entries by other library entries. (1)

(1) Past tense is used in some of the descriptions above to indicate a change in the operation of certain tools. For example, we no longer have MSLs (Multics Segment Lists), so the msl programs have been replaced by three new programs which perform the same function in a different way. These programs are described in a forthcoming MTB. When the Online Libraries were

Each of these tools has one or both of the following design flaws:

- either the tool used a special library data base which was invariably out-of-sync with the actual library contents (the MSL data base); or
- both the logical and physical organization of the Multics System Libraries is coded into the tool and therefore the tool has to be modified whenever the logical or physical organization changes in any way.

Therefore, the goals of the clean up campaign are to:

++ eliminate the information from the MSLs which is duplicated elsewhere in the libraries (e.g., status information for library entries, name of bound segment containing a component, language type of a component) and to store the remaining information in a data base which is simpler to maintain, easier to check for consistency, and which does not interact with the library installation tools; and

++ store the organization of the Multics System Libraries (the directory structure, naming conventions, and knowledge of the types of segments in particular directories) in a single data base which can be used by each tool, and which can be centrally updated when reorganizations occur.

Replacing the MSL

It has been fairly easy to meet the first goal stated above, because the only MSL information not contained elsewhere in the libraries (either as segment status information or as archive component header information) is the ID of the particular system in which a Hardcore or Salvager Library entry was last modified. However, there is a direct relationship between the date on which a library entry was last modified, and the date on which a particular system was installed in the libraries. Therefore, we can replace the MSL data bases with a much simpler data base consisting of a list of system IDs for Hardcore and Salvager systems, and the date on which those systems were installed in the libraries. Then, by comparing the date modified of each Hardcore or Salvager Library entry with this list, we can determine in which system the entry was last modified.

The list of system IDs is implemented as an array of system ID - date pairs, sorted by date (and therefore by system ID too). New

re-organized, get_library_source was extended to allow extraction of object segments from the Online Libraries and was therefore renamed get_library_segment.

commands add an entry to the bottom of the list each time a Hardcore or Salvager system is updated into the libraries, and replace or delete entries which are in error. When given a date last modified for a library entry, a new subroutine returns the appropriate system ID.

Note that the list is easy to maintain and to check for consistency, and that it does not interact with the Hardcore updater, but is updated instead (via command) by the installer at the end of the Hardcore or Salvager installation process.

Having replaced the MSL with the system ID list, it has also been necessary to replace the msl tools which reported on the information stored in the MSLs. msl_info will be replaced by library_info (coding is in progress), and msl_short_format and msl_global_format have been replaced by library_map. These new tools will be described in a forthcoming MTB.

Problem With Library Organization

One of the biggest problems confronting the library maintenance tools is the organization of the libraries themselves. For various reasons, the system is divided into different logical libraries, and these libraries are in turn divided into sub-libraries (or directories). Thus, we have the standard library, unbundled library, tools library, author-maintained library, installation-maintained library, network library, ... And we have, within each library, source directories, object directories, bind list directories, execution directories (those seen by the user), bound component directories, info directories, include directories, ...

Even more of a problem than the ever proliferating number of logical libraries is the mapping of these logical entities onto the physical directories of the Multics Storage System. Add to these the different naming conventions used in different libraries, the differing search procedures, the restrictions on the types of entries placed in libraries, etc and you have an almost unmanageable set of rules for maintaining and accessing entries in the libraries. Implementing reasonably efficient search procedures which can treat all of the libraries in a fairly uniform manner is an extremely difficult task. Implementing such procedures in <u>each</u> of the many library maintenance tools would be impossible.

The evidence in the paragraph above led directly to the conclusions: that the libraries must have the simplest organization possible while providing reasonable storage and access efficiency; that all libraries should have the same organization, if possible; and that the procedures for maintaining and accessing entries in the libraries should be common to all library maintenance tools, and should be centrally located in a single external module which can be easily modified.

Acting on these conclusions, in 1971 we began the process of reorganizing the libraries, starting with the Online Libraries (the largest). The new library organization was chosen for its efficient storage of entries, its ease and efficiency of access to entries, and its simplicity. (2)

It is our goal (though a distant one) to promulgate this new organization throughout all of the Multics System Libraries. The biggest barrier to a uniform library organization are the Hardcore and Salvager Libraries, which are currently organized in a manner to optimize the installation of large groups of modifications (new systems) at one time, rather than to promote ease and efficiency of access to entries and simplicity of organization.

Thus, there are currently two different organizations used in the Multics System Libraries, and we are likely to retain these two organizations for the foreseeable future.

Centralizing Library Organization Information

Having decided to centralize the knowledge of library organization into a single module, we first had to decide what knowledge was needed. The list below outlines the information which is currently being stored, or is known to be needed in the near future for proposed extensions to library maintenance commands.

A. the logical structure of the libraries, including library names, directory names, and the relationship between the various directories of a given library.
B. the mapping of this logical structure onto the physical directories of the Multics Storage System.
C. the conventions for separating the various types of library entries among the directories of a given library (e.g., source segments go in the source directory, info segments go in the info directory of a library, etc).
D. the conventions for storing the various types of library entries in the library directories, and for naming those stored entries (e.g., the source for bound segments is stored in a source archive, the archive is named bound_seg_name_.s.archive, and has additional

---

(2) The new library organization is described in MSB-87, "Plan for Multics System Library Conversion and for Shifting Library Maintenance to the 6180".

names for each of the source components it contains).

E. the conventions for accessing library entries in libraries with differing organizations.

F. the attributes of new entries placed in a library (e.g., ACL, ring brackets, AIM controls, etc).

G. the type of information which should be returned, by default, for the entries of various libraries (e.g., in the Online Libraries, ring brackets are important; they are not in the Hardcore Libraries).

H. the conventions for modifying and deleting library entries as part of the normal installation process.

The next step was to decide in what form to store this highly varied set of information. While some of the information is simple in nature and can easily be tabularized in some data structure, much of the information is too complex to be described by any data base generation language, or even to be stored in a general data base structure. Therefore, the information was split into two parts: that which could be tabularized in a data base; and that which had to be encoded into a program. A new data base and program were then created, along with a simple compiler for the data base. The data base is known as the library descriptor, and the program is called the library search program.

The Library Descriptor

Currently, the library descriptor contains:

1. a definition of the roots of the library, the parts of the library which remain constant across modifications made to the library, and from which a search can begin for library entries.
2. the names by which each library root can be referenced.
3. the relationship between a library and its sub-libraries, as expressed by common name components (e.g., the libraries standard.source, standard.object, and standard.lists share a name component, and are therefore related; similarly, standard.source, unbundled.source, tools.source, and auth_maint.source share a name component and are related).
4. the path name of the physical directory (3) which is the realization of the logical library root in the Multics Storage System.

---

(3) An archive may also be a library root, with its components being the library entries. For example, the bind_maps.archive of the Hardcore and Salvager libraries is a library root which contains, as archive components, the bind listings for the Hardcore Library bound segments.

5. an entry variable which defines the entry point in the library search program to be called to search for entries in the library root.

Future plans call for associating the following additional information with each library root:

6. the ACL, ring brackets, and AIM controls which are used by default when installing new entries in the library root.
7. a list of suffixes which define, through naming conventions, the types of entries which may be installed in the library root (e.g., a source library root can contain only **.s.archive, *.pl1, *.alm, *.fortran, *.bcpl, *.ec, ...).
8. an entry variable which defines the entry point in a library installation program to be called to install an entry in the library root.

In addition, the library descriptor defines the default library names and search names which are to be used with each of the library maintenance commands. These default values must be specified in the library descriptor, because they depend upon the names of the libraries defined in the descriptor, and on the naming conventions used for entries in the library. For each library maintenance command which uses the library descriptor, the following information is stored:

9. a switch indicating whether or not the command is supported by the library descriptor and library search program.
10. an array of default library names (which may be empty).
11. an array of default search names (names used to search for library entries; this array may also be empty).

A simple data base language was developed to define the contents of a library descriptor. Definitions written in this language are stored in library descriptor source segments, which have a name suffix of .ld; they are compiled into an ALM data segment by the library_descriptor_compiler (ldc), a reduction_compiler-generated compiler.

All references to library descriptors are made through a subroutine called lib_descriptor_, which is responsible for maintaining a constant user interface to the information across changes in the internal structure of the data.

The Library Search Program

The library search program contains one entry point for each class of library root. Library roots are classified according to the following criteria:

a. the kind of entries stored in the library root (e.g., source entries, or info entries, or executable entries, etc).
b. the type of entries stored in the library root (e.g., links, segments, directories, archives, MSFs).
c. the naming convention used in the library root, and the associated procedure for searching for library entries.
d. the way in which modifications are installed into the root, and the mechanism for flagging obsolete entries awaiting deletion.
e. the type of status information which should be returned by default for the various types of library entries in the root.
f. the depth in the library hierarchy (of directories, archives and MSFs) at which searching for a library entry below the root should be discontinued.

Each entry point in the library search program performs the searching functions for the various library maintenance commands according to the criteria appropriate to one library root class. The searching criteria are coded in normal PL/I code.

The result of the search is an information tree containing the status of all found library entries, plus the status of the parent, grandparent, ... of each found library entry up to and including status for the library root containing the found entry. The tree represents the physical (as opposed to logical) library structure containing the found library entries. The status information delineates each node of the tree as a link, segment, directory, archive, archive component, MSF, or MSF component, and includes enough other status information to perform the appropriate library maintenance function on found entries without further information.

Entry points are provided in the lib_descriptor_ subroutine to perform the type of searching appropriate to the particular library maintenance function being performed. This maintenance function information is passed to the library search program, which must tailor its searching criteria according to the library maintenance function.

General Library Maintenance Tools

By using the library descriptor and library search program, we have not only centralized the library organization into a single module, but have also enabled a sub-system maintainer to replace this module with one describing his sub-system libraries. He then has a complete set of library maintenance tools which will operate on his sub-system library in the same way as on the Multics System Libraries. This generalization of the library tools beyond the Multics System Libraries is a pleasant side effect of centralizing the library-dependent information.