

Identification

Multics Segment Library On-Line Information Base

Multics Segment List (MSL)

Edwin W. Meyer, Jr.

Purpose

The Multics Segment List (MSL) is a data format for an on-line segment containing information about a segment library. The MSL is designed for on-line user interrogation, for conversion to ascii for printing as a hard-copy library listing, and for use with automatic library update procedures. It is of sufficient scope as to be useful under a number of different maintenance philosophies.

Overview

An MSL is a set of segment information entries referenced in two ways:

(a) via a threaded list alphabetized by segment name; (b) via a hash-coded list keyed by segment name. Each entry contains information concerning one segment or other type of name.

The MSL uses LSM list structure format (MSPM BY.22) for speed and efficiency in entry look-up and modification. It is not an ascii segment, although it does contain ascii blocks. Thus it can not be directly printed.

MSL Entry Format

Each MSL entry consists of a 14 element node array plus various subsidiary LSM data blocks. (See MSPM BY.22.01 for LSM data organization.) In the description below, all items are character string blocks unless otherwise indicated.

<u>MSM array index</u>	<u>Item Identification</u>	<u>Description</u>
0	name	segment or other name
1	type_code	(binary) name type (see BV.9.02 for type_code list)
2	source_instal	installation date of source for this segment
3	object_instal	installation data of object for this segment
4	system_id	id of system of installation
5	who_auth	initials of author of segment
6	who_mod	initials of latest modifier of segment
7	area_use	basic area of use for this segment
8	document	MSPM BS abstract section
9	superior_list	node address of top of threaded list of superior MSL entries (see below)

<u>LSM array index</u>	<u>Item Identification</u>	<u>Description</u>
10	inferior_list	node address of top of threaded list of inferior MSL entries (see below)
11	path_list	node address of list of source and object pathnames. (see below)
12	proc_state	(binary)-used during update processing
13	nxt_entry	(node address) pointer to next entry in alphabetized list.

A superior/inferior list is a set of doubly threaded associative blocks (one block per name combination) that link an entry to superior or inferior entries. Each associative block is a 4 element node array of the following format:

0	sup_entry	(node address) pointer to the superior entry of the combination
1	inf_entry	(node address) pointer to the inferior entry of the combination
2	nxt_sup_blk	(node address) pointer to the next block in the superior list
3	next_inf_blk	(node address) pointer to the next block in the inferior list

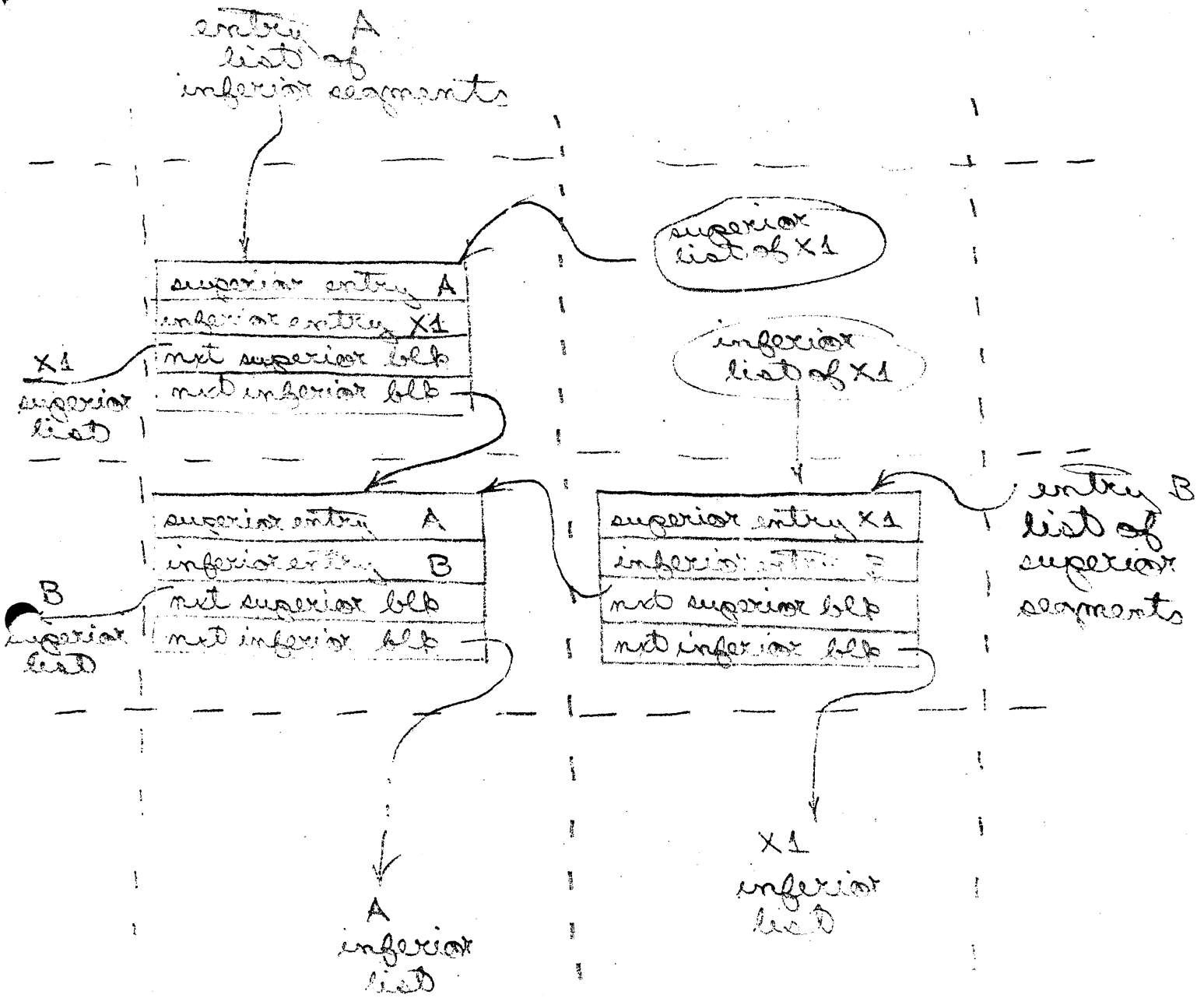


Figure 1. Example of Doubly Threaded Superior / Inferior Lists

When properly threaded, for each entry (j) in the superior list of an entry (A), that entry (A) is part of the inferior list of entry (j), and vice-versa. An example is illustrated in Figure 1.

The path list of an entry is a four element node array consisting of the following paths:

0	source_path	path of source segment
1	object_path	path of object segment
2	old_dir	path of directory containing previous source and object
3	info_dir	path of info segment (currently used for locating bound segment bind map)

The entry type\_code determines the interpretation of each of these paths in one of the following ways:

- (a) not used
- (b) free segment - pathname of containing directory
- (c) archived segment - pathname of archive

#### MSL List Structure

The root of the MSL list structure is a four node "root\_list":

<u>Item</u>	<u>Identification</u>	<u>Description</u>
0	alpha_list	(node address) pointer to the top of the alphabetized list of entries
1	hash_list	(node address) pointer to the hash list of entry names.

<u>Item</u>	<u>Identification</u>	<u>Description</u>
2	char_hash	(node address) pointer to a hash list of character strings other than entry names. (Ensures that only one physical character block is created no matter how many times it is used.)
3	type_list	(node address) pointer to a list of items defining the various type_codes.

"type\_list" is a node array whose 'j'th node points to an "item\_list" defining type\_code j.

"item\_list" is a 3-node array containing the following items:

0	type	2-char type code
1	source_suffix	suffix of source segment
2	path_code	4-element fixed binary array specifying the interpretation to be given the paths in the corresponding array positions of "path_list"

The following path codes are currently defined:

0	this position not used
1	free segment - pathname of containing directory
2	archived segment - pathname of containing archive