Published: 07/09/69

Identification

List Structure Manipulator (LSM)
Edwin W. Meyer, Jr.

Purpose

The list structure manipulator is a utility package that enables a procedure to easily and efficiently create, reference, and manipulate a data segment containing LSM standard list structure whose nodes consist of list, hash list, ascii, bit, and binary data types.

Overview

All list structure operations are performed within a working segment in the process directory. In order to read or alter a list structure segment, a procedure first issues a call to "pull" into the working segment the structure hanging off the root node of the LSM-type segment. The address of the root node of this new structure is returned to the caller. Similarly, in order to write a permanent LSM segment, a procedure issues a call to "push" the list structure hanging off a specified node into the specified segment and set the root node.

The lists of a LSM segment can be moved into the working segment without overwriting lists previously brought in. In this manner two or more LSM segments can be coalesced. On the other hand, the action of writing a list into a permanent LSM segment destroys all data previously in the segment and sets the root cell of that segment to point to the inserted list.

Each procedure that uses the list structure manipulator can create its own working segment. Thus there is no danger of unrelated procedures performing conflicting operations on the same working segment.

LSM Data Types

There are six data item types defined in version 1 of the list structure manipulator. (See Fig. 1) These are:

1. indirect - internal to LSM. Invisible to user.

2. fixed array - an array of fixed binary numbers of precision 35. The limiting case of 1 array element is used to represent a single number. The maximum number of array elements allocatable is currently 4094.

3. bit string - the largest allocatable string is 4094*36 bits in length.

4. character string - Multics standard ascii code. The largest allocatable string is 4094*4 characters in length.

5. node list - an array of node addresses. The maximum number of node elements allocatable is currently 4094.

6. hash list - consists of a node array of bucket lists. Each bucket is a forward threaded set of 3-element node array buckets. Node 0 points to the next bucket. Node 1 points to a reference character string data type and node 2 is a related node.

Data items are allocated in contiguous blocks of storage within a segment. The first word for all data item types is a specifier; the actual data follows in succeeding words.

A specifier word has the following format:

bits 0-5 "type"	type code - determines data item type
bits 6-17 "allo"	allocated block length - contains the number of words in this block (including specifier)
bits 18-35 "curl"	current data length - in units relevant to data type

A node address is a single word item with the following format:

bits 0-17 - =0	- reserved for process-independent segment index
bits 18-35	offset address within the LSM segment

0 is defined as the null address.

LSM Data Segment Format

The first three words of an LSM data segment are reserved as follows (see also Fig. 3):

word 0 "version"

=1 - version number of the LSM which created the segment

word 1 "free"

offset address of the first word of the free block. (The free block is a contiguous block of words lying between the allocated part of the data segment and its upper bound.) Its initial value is 3.

word 2 "root"

contains a node address pointing to the root node of the list structure

words 3 to 64K-1 This section is divided into two parts: the allocated portion and the free block. Initially the free block totally occupies this section, but as new data blocks are created, the necessary number of words are snipped off the low address end of the free block and allocated to the data blocks.

LSM Procedure Calls

```
call lsm1$init (pr);
     dcl pr ptr;
```

If a null pointer is supplied, lsm1$init finds or makes an empty working lsm segment in the process directory and returns its base pointer in `pr'. If `pr' is not null, lsm1$init assumes it to be a base pointer to a working segment to be truncated and re-initialized.

call lsm1$free (pr);

The segment pointed to by `pr´ is truncated to zero and added to the free working segment list for later re-use. This call should be made whenever an lsm working segment is about to be abandoned in order to prevent the buildup of many blocks of garbage in secondary storage. A null pointer is returned in `pr´.

call lsm1$gc (pr, count, active_node);
dcl count fixed bin (17),
active_node fixed bin (34);

If more than `count´ - 3 words have been allocated to data items in the working segment (base pointer `pr´), then it will be garbage-collected. This is done by transferring all of the list structure inferior to `active_node´ to a new working segment. Only items traceable from `active_node´ get transferred. The base pointer to the new segment is returned in `pr´ and the node equivalent to `active_node´ in the new segment is returned.

call lsm1$pull (pr, dir_path, entry, root_node);
dcl dir_path char(*),
entry char(*),
root_node fixed bin (34);

A copy of the list structure contained in `dir_path´ >`entry´ is moved into the working segment, and the address of the root_node of the working copy of the moved lists is returned in `root_node.´ If there is any error such that the list structure can not be pulled in, `root_node´ is returned as -1.

call lsm1$push (pr, dir_path, entry, root_node);

A copy of the list structure inferior to `root_node´ in the working segment is "pushed" into the segment `dir_path´>`entry´ and its root node cell is set to point to the homolog of `root_node.´ Any previous contents of the segment are overwritten. If the list structure can not be moved for any reason, `root_node´ is returned as -1. The original list structure inferior to `root_node´ is destroyed.

```
call lsm1$get_blk (pr, node, type, curl, bpr);

     dcl node fixed bin (34), /*address to data item*/
     type fixed bin (17),/*data type*/
     curl fixed bin (17),/*current length*/
     bpr ptr; /*pointer to start of data*/
```

The data block at address `node' in the working segment (base pointer is `pr') is investigated and its `type,' `curl,' and `bpr' to the first word of the data are returned. If `node' is not valid, `type' is returned as 0.

```
call lsm1$make_blk (pr, node, type, curl, bpr);
```

A new data block is created in the working segment of type `type,' current length `curl,' and the minimum allocated length sufficient to encompass the current length. The node address of the created data block is returned in `node' and a pointer to the first word of the data area is returned in `bpr.' If a new block can not be created, `node' is returned as -1.

```
call lsm1$hash (pr, node, op_code, keyp, key1, s_node, r_node);

     dcl op_code fixed bin(17), /*operation and key type code*/
     keyp ptr, /*pointer to base of key character string*/
     key1 fixed bin(17), /* character length of key*/
     s_node fixed bin(34), /*node of key string supplied or
         returned
     r_node fixed bin(34), /*related node associated with
         key string - supplied or returned*/
```

This call checks the hash list at `node' in the working segment for the existence of the key character string defined by `keyp' and `key1' or `s_node' (depending on `op_code') and adds, deletes, reads, or writes the key-string and its related node.

op_code		operation	
key defined by: keyp, key1	s_node	if key found	if key not found
0	4	reads into \`s_node' and \`r_node'	\`r_node' = -1
1	5	same as above	adds key to list - returns \`s_node' and \`r_node' = 0
2	6	writes \`r_node' into related node cell - returns \`s_node'	adds key to list - writes \`r_node' - returns \`s_node'
3	7	deletes key and related node from hash list	returns \`r_node' = -1

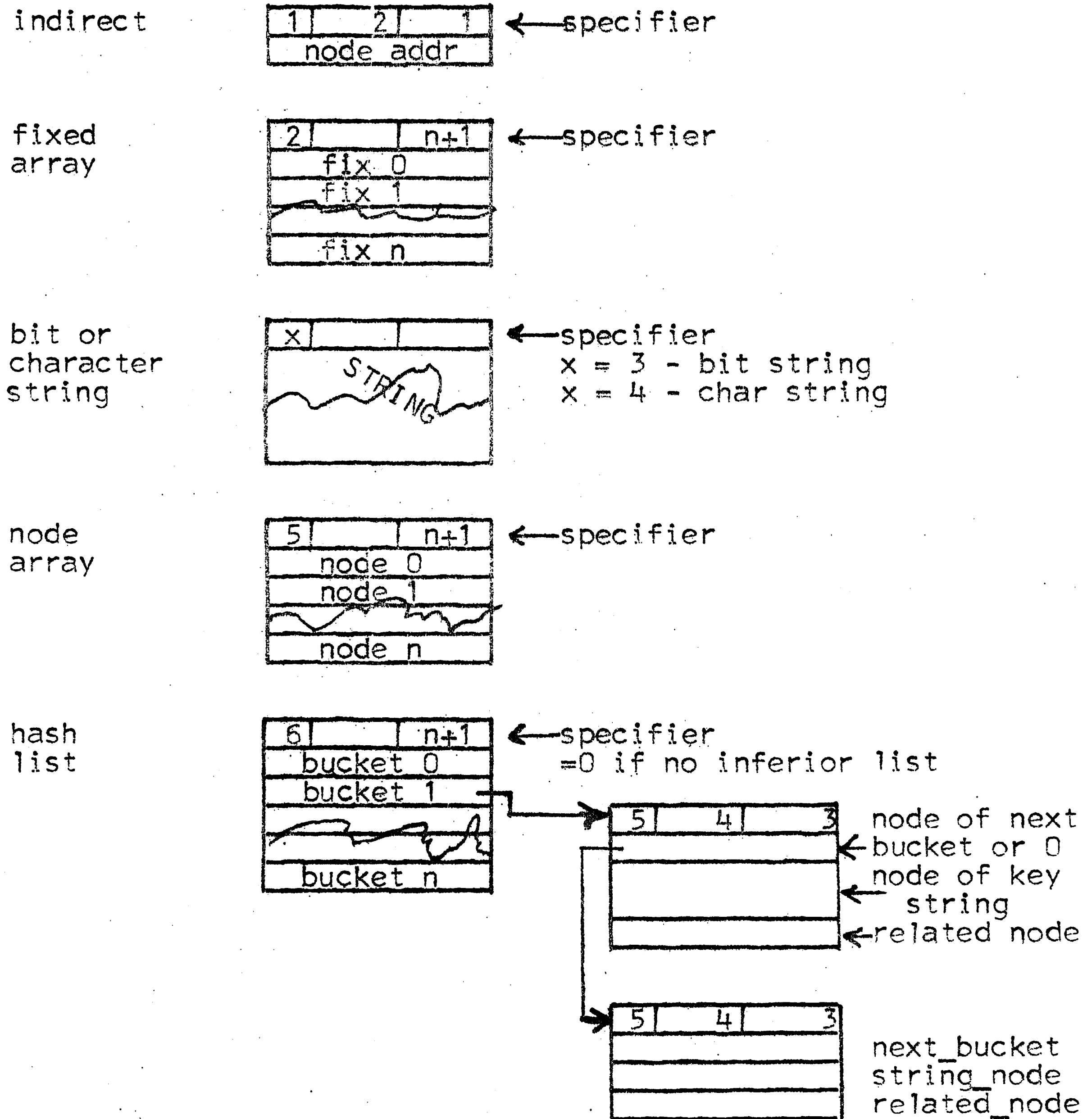

Figure 1. LSM Data Types

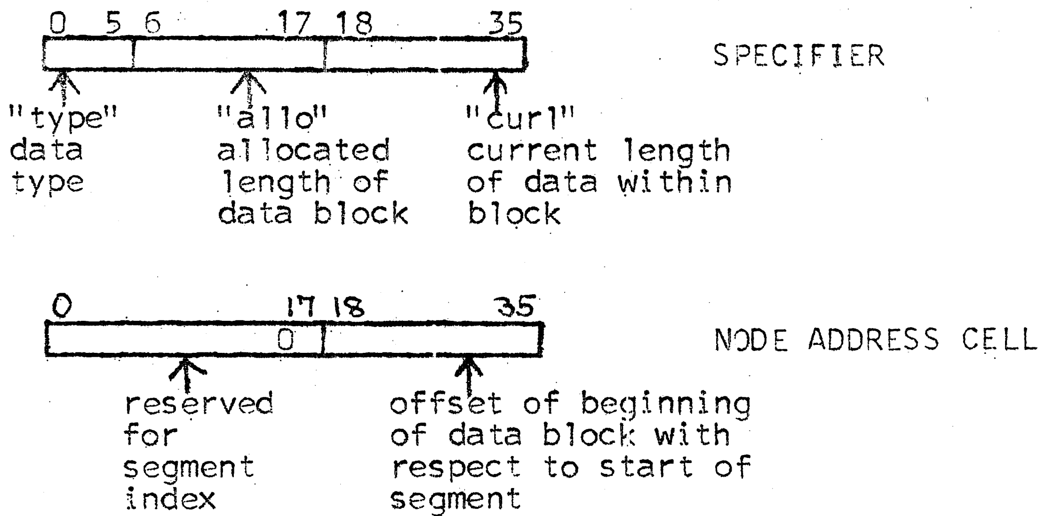

Figure 2. Specifier and Node Address Formats

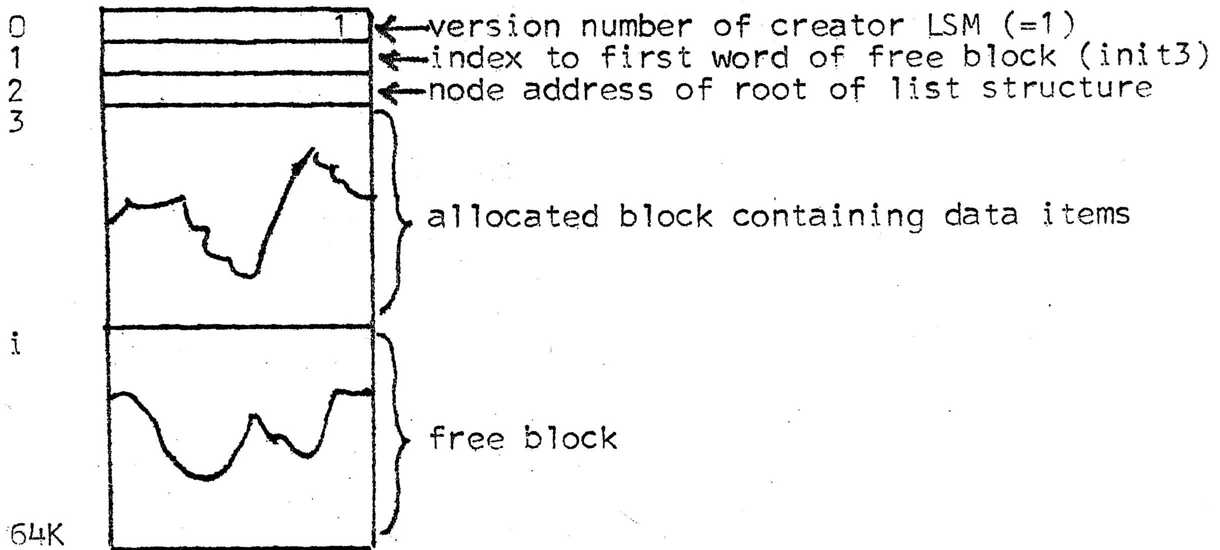

Figure 3. LSM Data Segment Structure