TO: MSPM Distribution
FROM: E. W. Meyer
SUBJECT: BX.17.01
DATE: 03/11/68

This revision of GECØS Segment Loader for Multics reflects the following changes:

(1) a new control line, "lowload j" has been implemented.

(2) the calling sequence required to interface with the "call_out" control line has been altered to conform to GMAP and FØRTRAN standards.

(3) there is now a 64 word slave prefix occupying words 0-63 of the created segment.

(4) a new option, "list" has been added.

(5) gecos_seg may create SYMDEF's for its own purposes.

(6) error codes are now defined.

(7) many "gecos_seg" procedure segments have been renamed, combined, or added.

Published: 03/08/68
(Supersedes: BX.17.01, 08/24/67;
BX.17.01A, 12/15/67)

Identification

GECØS segment loader for Multics
gecos_seg
E. W. Meyer, Jr. and D. B. Wagner

Purpose

The gecos_seg command is the basic tool to be used in carrying 635 programs into Multics. Gecos_seg loads a collection of 645 object "decks" into an impure procedure segment and sets up linkage and call-translators so that the procedure can communicate with its environment.

The loader which operates as part of the gecos_seg command is a one pass loader which loads successive GMAP programs into a created Multics segment from the highest location downward and makes proper interprogram links. None of the GECOS standard options or debugging facilities are available, nor is a library search made. Only the preface and text cards are recognized, all other are ignored. On-line error messages are confined to errors within the loader program itself, such fatal errors as segment full or misplaced preface card, and a listing of all undefined SYMREFS following the loading of the program.

Usage

Gecos_seg is invoked through the command line,

gecos_seg alpha

where alpha.gecos_seg is an ascii "load list" file. This file is a series of "control lines" telling which 635 object decks to load, what call-translators to create in the segment, and what in-references to include in the linkage section. The procedure segment created will have the name alpha.

Basic Control Lines

The control line

segment_size n

where n is a decimal number, directs gecos_seg to create the text segment alpha of length n words and to load the 635 object decks from the highest location downward. The highload loading order is illustrated in Figure 1(a). If the unallocated region between blank common and the program region becomes used up during loading, an error will be signaled and loading discontinued.

The control line

    lowload j

where j is a decimal number, can be used instead of the "segment_size" control line to direct gecos_seg to load the 635 object checks from lowest location upward. j specifies the length of the blank common region. The lowload loading order is illustrated in Figure 1(b).

The control line

    object beta

directs gecos_seg to load the 635 object deck beta.635object into the segment. Gecos_seg relocates addresses properly and links all SYMREF's and BLOCK's exactly as GELOAD would. (Beta may be composed of a directory pathname and entry prefix).

The control line

    reference_in gamma delta

directs gecos_seg to create linkage so that the reference alpha$gamma (where alpha is the name of the procedure segment being created) refers to the SYMREF delta.

The control line

    call_in gamma delta

directs gecos_seg to set up a "call translator" in the text segment. This call translator does a standard Multics save, followed by a standard Multics return. Gecos_seg creates linkage as for the reference_in control line so that a standard Multics call to alpha$gamma reaches this call translator. No arguments are permitted on a call in.

The control line

call_out gamma delta

specifies to gecos_seg that all SYMREF's to the name delta (which presumably are referenced only in GECOS standard calls) should go to a call translator which makes a standard Multics argument list and calls the procedure gamma. (The case in which gamma consists of segment$entry is allowable). Only one argument is allowed in a call out.

The control lines must be ordered as follows:

(a) one "segment_size n" or "lowload j" control line.*

(b) one or more "object beta" control lines.

(c) any number or none of the following (in any order):

"reference_in gamma delta"
"call_in gamma delta"
"call_out gamma delta"

* it is possible to achieve overlays through the repeated use of "segment_size" and "lowload" control lines interspersed within a group of "object" control lines. A "lowload j" control line will direct the loading of the following 635 object decks upward from location j + 64, and a "segment_size n" control line will direct the loading of the following object decks downward from location n.

Slave Prefix

The slave prefix is 64 word block occupying words 0 - 63 of the text segment, corresponding to the slave prefix used by GECOS for inter-activity communication. (see CPB-1003D pp. 125-130).

The only cell that is defined by gecos_seg is absolute location 31, which contains program load limits. At the end of loading, the left half of address 31 contains the address of the first location above the blank common region, and the right half contains the address of the first location below the subprogram and labeled common region.

In a highload ("segment_size" control line used), this identifies the limits of the unallocated region. If multiple "lowload" and "segment_size" control lines are used, this information will be incorrect.

Options

Two options are defined:

<u>brief</u> curtails all on_line output except error messages. If "brief" is off, a load map is written out.

<u>list</u> directs gecos_seg to produce the listing segment <u>alpha</u>.list following completion of the loading. If "list" is off, no listing segment is produced.

Special SYMDEF's

The SYMDEF's ƶ00001 thru ƶ99999 may be generated within the gecos_seg command in order to process "call_in" and "call_out" control lines, and should not be used within the supplied object decks.

Undefined Symbols

If any subprograms have made external references (SYMREF's) which have not been defined within other subprograms, the on-line comment "undefined symbols" will appear, followed by (if "brief" is off) a listing of all undefined symbols. Text segment cells making references to such symbols will not be relocated. This is a non-fatal error.

Fatal Errors

Gecos_seg uses the standard error-handling mechanism as described in MSPM sections BY.11.00 - BY.11.04.

The following error codes are defined:

User errors

21 - control line not recognized.

22 - intersection of blank common and program region during highload.

23 - non-decimal character in control line numeral.

24 - SYMDEF supplied for "reference_in" or "call_in" control line not found.

25 - created segment can't be moved into the working directory.

26 - segment can not be created in process directory.

Possible object deck errors

31 - premature deck end during repeated preface card search.

32 - repeated preface card not found.

33 - attempt to load beyond segment limits.

34 - non-existant SYMREF index.

35 - attempt to retrieve beyond limits of card image.

System errors

41 - text_seg: out of bounds on retrieval

42 - ascii_out: non-existant line number

43 - ascii_out: target segment unspecified

44 - tbl: "def" out of bounds

45 - seg_control: segment can not be expanded

46 - seg_control: segment can not be retrieved

Implementation

The loading process operates by sequentially reading the control lines and performing the action designated by each.

The control line

"segment_size *n*" or "lowload j"

causes the creation in the process directory of the text and linkage segments <unique_name> and <unique_name>.link, where <unique_name> is a unique 15 character string created by calling "unique_chars". The loader is set to its initial highload or lowload state and the definitions, reference, and linkage tables are cleared (see below).

The control line

"object beta"

initiates the object deck segment beta.635object and starts the loader, reading preface and text card images from the object segment and processing them.

The processing of the preface and text cards involves the use of three internal tables declared within the procedure "tbl":

Definitions Table (def)

```
dcl 1 def (d_top) based (d_pntr),

      2 sym bit (36),              /* external symbol */

      2 val fixed bin (18),        /* value of sym */

      2 xlk fixed bin (18),        /* index to undefined
                                      symbol chain */

      2 sdef bit (1);              /* defined switch */
```

"def" is a linear array of substructures, each consisting of four elements:

sym - a six character bcd symbol designated as a SYMDEF, SYMREF, or LABELED COMMON type external symbol within a preface card.

val - the relocated value of this symbol. In the case of a SYMREF, the value is undefined prior to encountering the corresponding SYMDEF.

xlk - zero unless the symbol is undefined and fields involving it have been loaded into the created procedure segment. In that case xlk is an index to a list in the lnk table (see below) of the fields in the loaded program which require the value of this symbol.

sdef - a switch indicating whether or not the value has been defined. "0"b = undefined; "1"b = defined.

It is appended to whenever an external symbol entry which does not already exist in def is encountered on a preface card. The definitions table is maintained throughout the loading process; it is cleared only through the action of the "segment_size n" or "lowload j" control line.

Reference Table (ref)

```
     dcl ref (r_top) fixed bin (18) based (r_pntr);
```

The reference table is a linear array of indices to symbol substructures within "def". It is cleared upon encountering the first preface card of a new subprogram. Each SYMREF or LABELED COMMON symbol encountered thereafter within the preface card group causes an index to the symbol's position in "def" to be appended onto the reference table. Thus a text card entry using the jth external symbol reference declared within the preface card(s) need only refer to it by the number j. The symbol's position in "def" can be picked up from ref(j).

Linkage Table (lnk)

```
     dcl 1 lnk_st(l_top) based (l_pntr),

           2 xlnk fixed bin (18),        /* index to next
                                            substructure in
                                            the list */

           2 xtseg fixed bin (18),       /* index to loaded
                                            program word */

           2 l_r bit (1),                /* left/right switch */

           2 p_m bit (1);                /* plus/minus switch */
```

It may happen that during the loading process a program with SYMREF external references is loaded before all those SYMREFs have been defined via SYMDEFs in the preface cards of other subprograms. Whenever a text card entry that uses such a SYMREF is encountered, the "def" entry for that symbol will be found to have an undefined value.

If this is to be a one pass loader, there must be a way to load these fields into the text segment as they are encountered and later relocate them when the SYMREF is defined with its SYMDEF in a subsequent program. A solution to this problem is to load the absolute value of the addend into the program field and to make an entry under a list for that symbol consisting of the following:

(1) the address of the field relative to the beginning of the text segment.

(2) a switch setting indicating whether the field is in the left or right half of the word.

(3) a switch setting indicating whether the sign of the addend is plus or minus.

When the external symbol is eventually defined by a SYMDEF entry in a subsequent program's preface card, the loader will go through the loaded field list for that symbol and properly relocate the listed fields using the newly defined value of the symbol. Subsequent fields using that SYMREF will be relocated and loaded directly.

The linkage table contains the loaded field lists of all undefined SYMREFs. It is essentially a free pool of substructures within an allocated structure. Each substructure is either unused (in which case it is chained to a free substructure list) or it is an entry in the loaded field list of some undefined SYMREF. Each substructure that is part of such a list contains the three elements "xtseg", "l_r", and "p_m" describing the position of the field requiring relocation and the sign of its addend, and also the index "xlnk" to the next entry in the list. The last entry in each list signals this fact with "xlnk" = 0. An index to the first entry in each list is contained in the substructure "xlk" of the corresponding SYMREF's entry in the definitions table.

Figure 2 illustrates the relationship between "ref", "def", "lnk", and the program segment.

The Preface Card

A preface card signals the beginning of a new subprogram. When it is encountered, space for the new program is allocated in an area of the text segment just below or above the previously allocated program region. The BLANK COMMON length becomes the maximum of the current length and the requirements of the new program. If (in highload) the new program and BLANK COMMON regions intersect following this allocation, loading is discontinued and a "segment full" error is signaled.

Next the external symbols on the preface card are processed. The class code of each entry is checked to determine whether the symbol is a SYMDEF, SYMREF, or LABELED COMMON, and the indicated action is taken:

SYMDEF (class code = 0 or 1) -- def is searched for the symbol. If it does not exist a new structure is appended onto def to hold the symbol and the value supplied in its preface entry. If it exists and is already defined, no action is taken. If it exists in an undefined status (previously appended by a SYMREF) the SYMDEF's value is inserted and any existing loaded field list is operated upon.

SYMREF (class code = 5) -- def is searched for the symbol; if it isn't found, an entry for it with undefined status set is appended onto def. In either case, the index to the symbol's position in def is then appended onto the reference table.

LABELED COMMON (class code = 6 or 7) -- def is searched for the symbol. If it is not found in def, space is allocated for the labeled common at the bottom or top of the currently allocated program region (in highload check for BLANK COMMON and program region intersection; if so, do a "segment full" error return), and a structure for the symbol is appended onto def with "val" set to the starting cell of the LABELED COMMON area. If the LABELED COMMON symbol is originally found in def, no program space is allocated as a previous allocation applies. In both cases an index to the symbol's position in def is appended onto ref.

All external symbol entries are processed in this manner. When the preface card is exhausted a check is made to determine whether or not more entries are expected on immediately following preface cards. If so the next card is read in (any type other than preface card generates an error). Otherwise it is assumed that text cards for the current program follow. The next preface card encountered terminates the loading of the current program and allocates space and initiates loading of the new program.

The Text Card

Whenever a text card is encountered in the object segment the program area allocation and external symbol declarations of the immediately preceeding preface card(s) apply. The header of the text card provides information as to whether the text card entries are to be relocated and loaded into the current program area or into one of the LABELED COMMON regions declared in the preface card(s). It also provides a loading address relative to the beginning of that region for the first text entry. Subsequent entries are loaded into sequentially higher locations.

Relocation and loading of the text card entries occurs as described in pp. 21-23 of the G.E. General Loader Manual CPB-1008D. Whenever a field involving an undefined SYMREF is encountered, the absolute value of the addend is inserted into the field and a structure pointing to the loaded field is appended onto the symbol's external linkage chain. These fields are relocated when the symbol is defined.

It may happen that another header word follows the text entries of the previous header on the card. In this case, reload the header information and process the following text entries accordingly.

The control line

    reference_in gamma delta

causes a search of def for the 36 bit GMAP representation of delta and the creation in the external symbol table of the ascii symbol gamma and the value of delta found in def. If delta is not found in def or its value is undefined an error is generated.

The control line

    call_in gamma delta

causes a search of def for delta. If it is not found or is undefined, an error is generated. Otherwise a Multics save sequence, a GMAP TSX1 <value of delta in def> call instruction, and a Multics return sequence are appended onto the bottom or top of the program region through the loading of a pseudo-object deck. An entry for gamma and a linkage to the new call_in sequence is created in alpha's linkage section.

The control line

    call_out gamma delta

causes a Multics calling sequence to the external entry gamma to be created in the text segment under construction. The first instruction of this sequence has the SYMDEF delta.

The following GMAP calling sequence is required:

```
     loc     TSX1 delta     (delta is a SYMREF)

     loc+1   <return location>

     loc+2   ignored

     loc+3   <argument>
```

The instruction sequence created and loaded under SYMDEF delta does the following:

(a) saves the return address

(b) creates a one-argument list consisting of the argument count plus an its pair containing the address contained in the left half of loc+3.

(c) performs a standard Multics call to the external entry point gamma

(d) (after returning from gamma) returns to loc+1.

After processing all the control lines, the undefined symbol search is made, the list segment <unique_name>.list is prepared if requested, and then the segments are transferred from the process directory to the working directory as alpha, alpha.link, and alpha.list.

Procedure Segments of the Loader Program

ascii_out

contains entries used by various procedures to assemble and write-out console output, and entries used by "load_list" (see below) to insert assembled output lines into the listing segment.

deck_seg

is used to reference object deck card images in order of sequence and to retrieve a card image subfield.

ext

processes the control lines "reference_in", "call_in", and "call_out".

gecos_error

is used to signal all fatal errors occuring within the gecos_seg command.

gecos_seg

is the upper level procedure of the loader and is invoked by the shell upon receipt of a gecos_seg command.

It initiates the control line segment alpha.gecos_seg and reads the control lines, calling the proper procedure to handle each. Upon exhaustion of the control lines, it initiates the necessary cleanup routines.

link_seg

is called by "ext" to add linkage definitions, external symbols, and external entries to the segment. Its initializing and cleanup routines are called directly from "gecos_seg".

load_list

is called by "gecos_seg" to produce a load listing of the created text and linkage segments.

object

is called by "gecos_seg" or "ext" to process an object deck. It calls "deck_seg" to reference and identify the card images, calling "pref" or "text" (see below) upon recognition of a preface or text card respectively.

pref

is called by "object" to process a preface card. If the preface card indicates that related preface cards follow, it reads these cards itself and does not return to "object" until all related preface cards have been processed.

seg_control

is the interface between the gecos_seg command and the basic file system. It is used to create and move segments, to retrieve base pointers and current lengths, and to update segment lengths.

tbl

contains all the entries involved in initializing and referencing the internal tables "def", "ref", and "lnk".

text

is called by "object" to process a single text card. It returns to "object" after all entries on this card have been relocated and loaded.

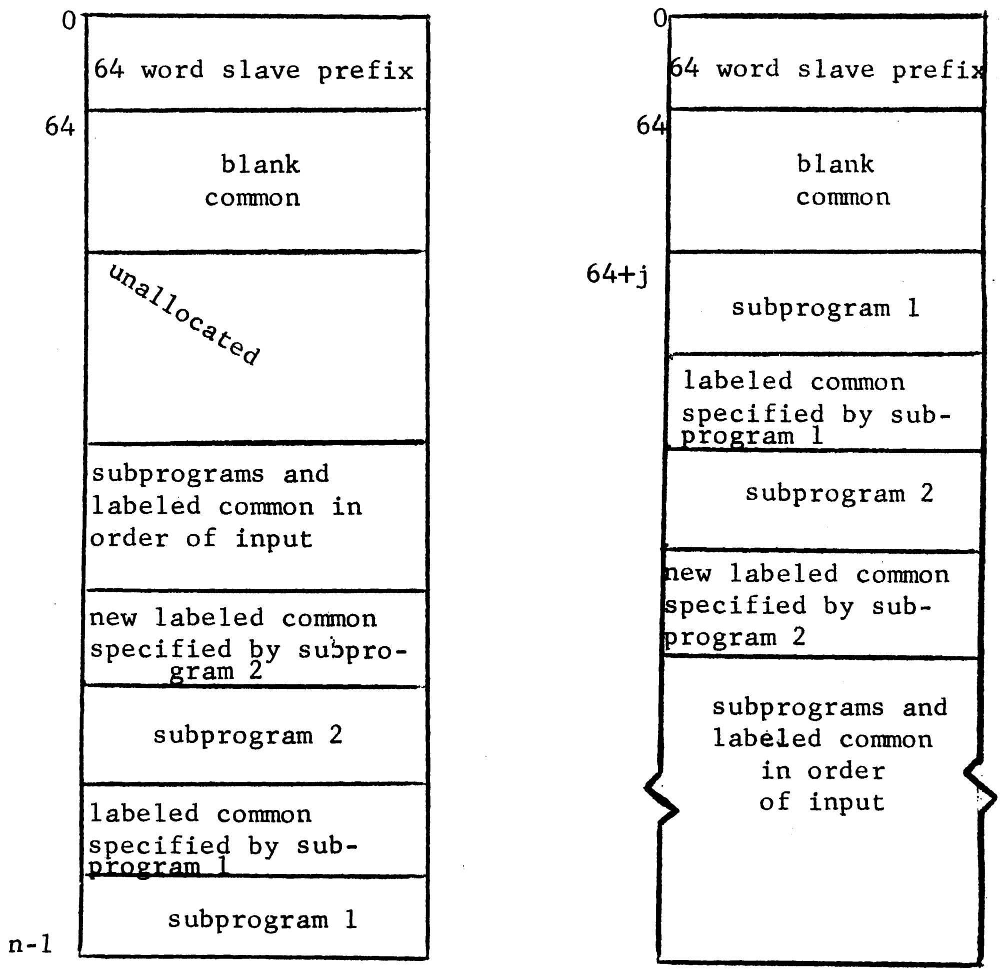
text_seg

contains entries to retrieve and insert cells into the text segment.

util

contains various utility routines used by other procedures of gecos_seg.

In addition, there are two data segments, "call_in.635object", and "call_out.635object", which are used as object deck prototypes by the "call_in" and "call_out" control line handlers.


0
64 word slave prefix
64
blank common
unallocated
subprograms and labeled common in order of input
new labeled common specified by subprogram 2
subprogram 2
labeled common specified by subprogram 1
subprogram 1
n-1
0
64 word slave prefix
64
blank common
64+j
subprogram 1
labeled common specified by subprogram 1
subprogram 2
new labeled common specified by subprogram 2
subprograms and labeled common in order of input

(a) highload (b) lowload

Figure 1. gecos_seg loading order

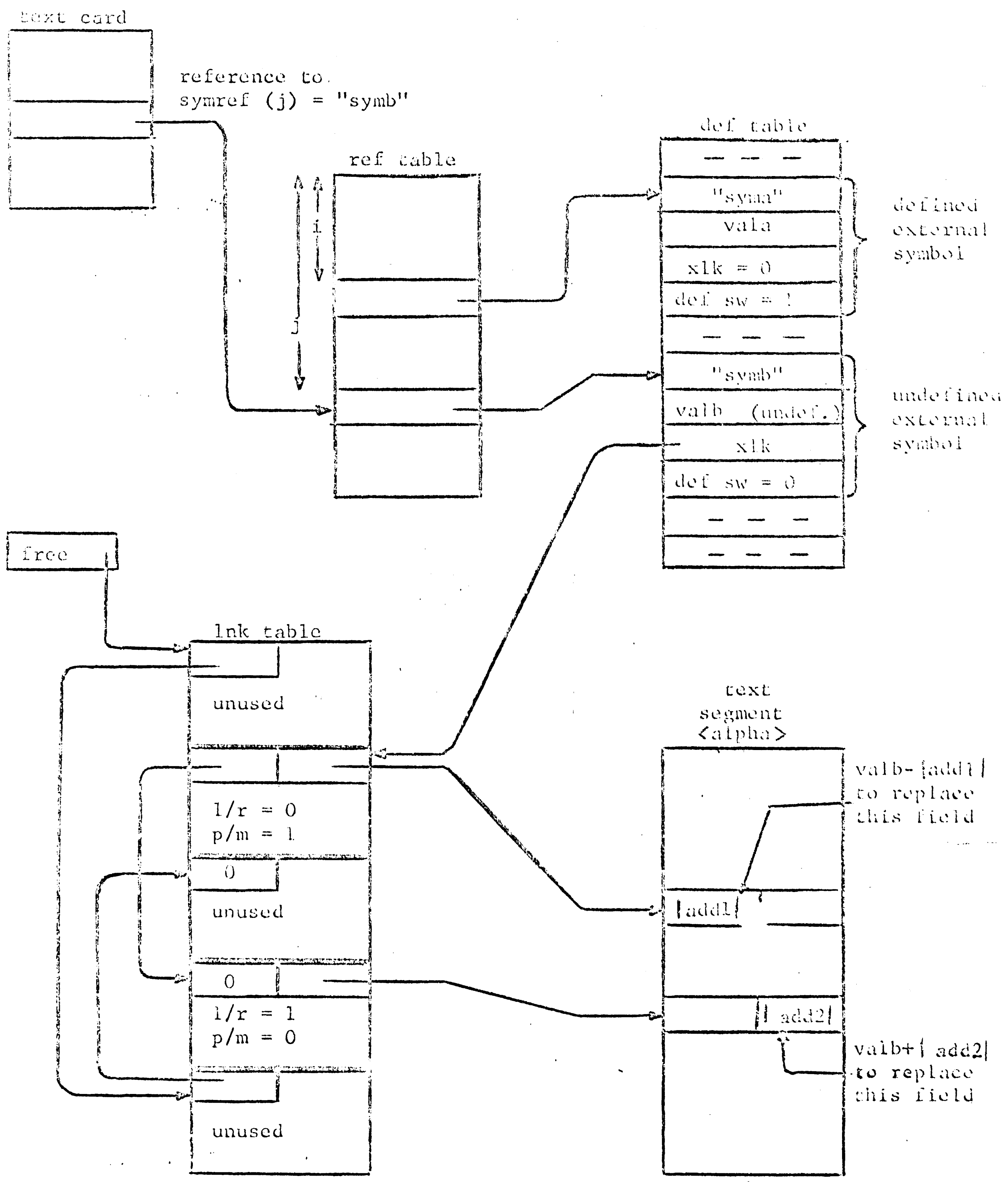


Figure 2. Relationships among the Internal Tables and the Text Segment