Loader-created Tables [C]

The loader can create and initialize the contents of several tables in the generated program. Four of these loader-created tables, the _infoblk, $SEGRES, Segment Linkage, and Segment Description tables, are described in this appendix.

_infoblk
C.1

The _infoblk table is created whenever the SYSTEM=UNICOS directive is used. This directive is normally found in the default directives file, and _infoblk is normally created for all UNICOS programs. The table contains general information, such as the size of various program sections, time and date of program creation, and version of the loader. _infoblk is structured as follows:

Word	0	32	63
0:	vers	////	a	len
1:	name			
2:	cksum			
3:	date			
4:	time			
5:	pid			
6:	pvr			
7:	osvr			
8:	udt			
9:	fill			
10:	tbase		dbase	
11:	tlen		dlen	
12:	blen		zlen	
13:	cdatalen		lmlen	
14:	amlen		mbase	
15:	hinit		hinc	

Word	0	32	63
16:	`sinit`	`sinc`	
17:	`usxf`	`usxl`	
18:	`mtptr`	`cmptr`	
19:	`/ / / / / / / / / / /`		
20:	`sgptr`	`////`	
21:	`taskstk`	`taskincr`	
22:	`u s e r 1`		
23:	`u s e r 2`		

Table 6. `_infoblk` description

Field	Word	Bits	Description
`vers`	0	0–6	`infoblk` table version (currently equals 1).
`a`	0	31	`fill` Address Generation flag (used by the system startup routine to insert address in filled words).
`len`	0	32–63	Number of words in `_infoblk` (currently 24).
`name`	1	0–63	ASCII `_infoblk` table name (“infoblk”). Null-terminated.
`cksum`	2	0–63	Check sum of `_infoblk` contents.
`date`	3	0–63	Date of program creation in ASCII *mm*/*dd*/*yy* format.
`time`	4	0–63	Time of program creation in ASCII *hh:mm:ss* format.
`pid`	5	0–63	ASCII name of loader that created program. Null-terminated if name is less than 8 characters.
`pvr`	6	0–63	ASCII version of loader that created program. Null-terminated if name is less than 8 characters.
`osvr`	7	0–63	ASCII operating system active when program was created. Null-terminated if name is less than 8 characters.
`udt`	8	0–63	Date and time of program creation in UNICOS time-stamp format.

Table 6. `_infoblk` description
(continued)

Field	Word	Bits	Description
`fill`	9	0–63	Value used by system startup routine to fill uninitialized areas of memory.
`tbase`	10	0–31	Base address of program text address space.
`dbase`	10	32–63	Base address of program data address space.
`tlen`	11	0–31	Number of words in text section.
`dlen`	11	32–63	Number of words in initialized data section.
`blen`	12	0–31	Number of words in uninitialized data section.
`zlen`	12	32–63	Number of words in zeroset data section.
`cdatalen`	13	0–31	Number of words in initialized data section prior to compressed data expansion.
`amlen`	14	0–31	Number of words of auxiliary memory used.
`mbase`	14	32–63	Base address of managed memory area.
`hinit`	15	0–31	Initial size of program heap.
`hinc`	15	32–63	Heap expansion increment value.
`sinit`	16	0–31	Initial size of program stack.
`sinc`	16	32–63	Stack expansion increment value.
`usxf`	17	0–31	First address of `$USXMSG` jump table.
`usxl`	17	32–63	Last address of `$USXMSG` jump table.
`mtptr`	18	0–31	Address of machine targeting information block.
`cmptr`	18	32–63	Address of first entry in data compression entry list.
`sgptr`	20	0–31	Address of `$SEGRES` segmentation information block.
`taskstk`	21	0–31	Initial size of slave task stack.
`taskincr`	21	32–63	Task stack expansion increment.

Table 6. `_infoblk` description
(continued)

Field	Word	Bits	Description
`user1`	22	0–63	Reserved for users.
`user2`	23	0–63	Reserved for users.

The contents of the `_infoblk` table may be accessed from a C language routine by including the following statements:

```
#include <infoblk.h>
extern struct infoblk _infoblk;
```

Segmentation tables
C.2

The loader builds several tables into each segmented program. These tables are used by the segmentation routines included in the program to manage the segments in memory. The `$SEGRES` table contains general segmentation information, including the addresses of the other segmentation tables. The Segment Description table (SDT) contains one entry for each segment in the program. Each SDT entry describes the size, location, and residency status of each segment. The Segment Linkage table (SLT) contains one entry for each intercepted subroutine call that may result in loading a new segment. Each SLT entry describes the target segment and address needed to complete the subroutine reference.

$SEGRES *table*
A.1.1

The $SEGRES table can be accessed through the common block /$SEGRES/. The other tables must be located through the addresses contained in $SEGRES. The $SEGRES format is as follows:

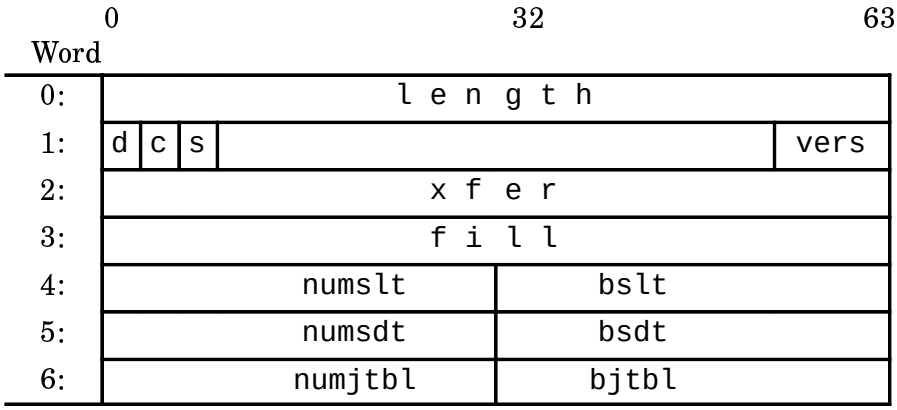

Table 7. $SEGRES description

Field	Word	Bits	Description
length	0	0–63	Number of words in $SEGRES table.
d	1	0–0	Flag indicating segmentation debug mode.
c	1	1–1	Flag indicating that segments should be copied to a scratch file.
s	1	2–2	Flag indicating that split segment mode is active.
vers	1	58–63	$SEGRES table version (currently equals 2).
xfer	2	0–63	Address of user main entry point.
fill	3	0–63	Fill value used to preset the uninitialized data section of each segment.
numslt	4	0–31	Number of entries in Segment Linkage table.
bslt	4	32–63	Base address of Segment Linkage table.
numsdt	5	0–31	Number of entries in Segment Description table.
bslt	5	32–63	Base address of Segment Description table.

Table 7. $SEGRES description
(continued)

Field	Word	Bits	Description
`numjtbl`	6	0–31	Number of entries in interception jump table.
`bjtbl`	6	32–63	Base address of interception jump table.

Segment Linkage table C.2.1

The Segment Linkage table (SLT) is included in every segmented program. The SLT describes the inter-segment linkages in the program. The Segment Linkage table entry format is as follows:

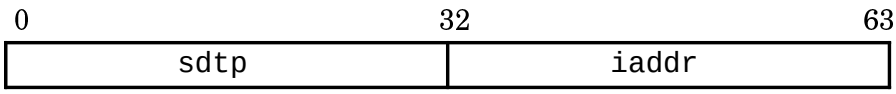

Table 8. SLT description

Field	Word	Bits	Description
`sdtp`	0	0–31	Address of SDT entry for target segment.
`iaddr`	0	32–63	Parcel address of target routine.

Segment Description table

C.2.2

The Segment Description table is included in every segmented program. It describes each segment included in the program. The Segment Description table entry format is as follows:

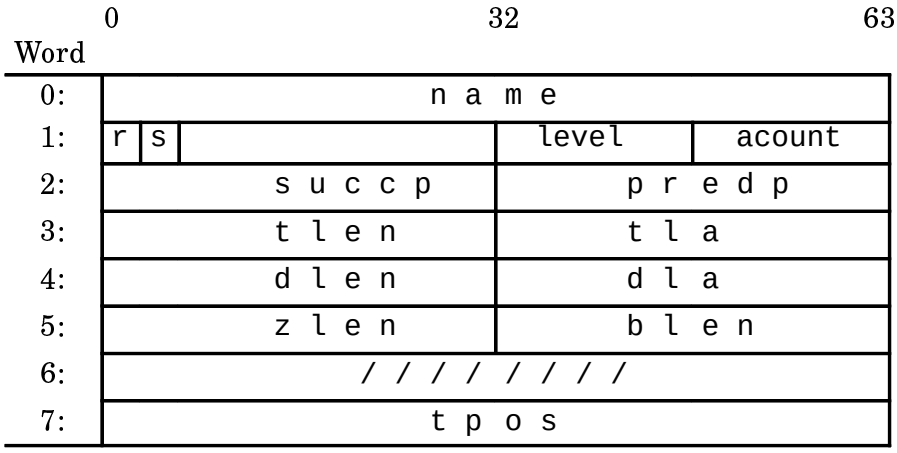

Table 9. SDT description

Field	Word	Bits	Description
`name`	0	0–63	ASCII name of segment. Null-terminated if name is less than 8 characters.
`r`	1	0–0	Flag indicating memory residency status of segment.
`s`	1	1–1	Flag indicating segment contents should be written to scratch file before overwriting with another segment.
`level`	1	32–47	Level of segment within segment tree.
`acount`	1	48–63	Number of active calls to routines within segment.
`succp`	2	0–31	SDT entry address of memory-resident successor segment.
`predp`	2	32–63	SDT entry address of predecessor segment.
`tlen`	3	0–31	Number of words in segment text section.
`tla`	3	32–63	Base address of segment text section.
`dlen`	4	0–31	Number of words in segment data section.
`dla`	4	32–63	Base address of segment data section.
`blen`	5	0–31	Number of words in segment uninitialized data section.
`zlen`	5	32–63	Number of words in segment zeroset data section.
`tpos`	7	0–63	Byte position within file of segment contents.

Glossary

absolute binary module — A binary module that the linkage editor has bound. All relative addresses within the bound object modules have been resolved. Also, all external and entry points in these modules have been resolved satisfactorily. This module is considered executable. The name for this module comes from COS where it was referenced as $ABS.

barrier — In macrotasking, a mechanism to synchronize tasks. Encountering a barrier causes a task to wait until all tasks have reached the barrier.

bin file — Files specified in `BIN` directives, which are specified as `segldr`(1) command-line option-arguments. By convention, `bin` files should be the portion of your program that you have written. See also *object module*.

block — (1) The smallest allocation unit in a file system; a group of contiguous characters recorded on and read from magnetic tape as a unit. Blocks are separated by record gaps. A block and a physical record are synonymous on magnetic tape. Usually, a block is the size of one physical disk sector. (2) A logical term denoting an arbitrary amount of data; generally a synonym for a 4096-byte hardware sector. See also *sector*. (3) A structure defined by each language processor that represents a contiguous area of memory. Blocks can be local to the defining object module (local blocks) or shared between modules (common blocks). A block can contain instructions, data, or both.

branch segment — Any segment in a segment program that is not the root segment. Branch segments are brought into memory when required, and they may be overwritten by other branch segments.

BSS — The part of a program containing uninitialized data. Space for the area is allocated at execution time.

BSSZ — A BSS area that is initialized to zero.

CAL Cray Assembly Language

CDBX An interactive, symbolic debugger that can be used to perform source-level debugging while executing programs running under UNICOS.

common block A block of memory that will be shared by more than one object module. (1) A Fortran data area that contains data that is accessible to multiple parts of a program. `COMMON` is a type of scope declaration in Fortran that makes variables accessible to multiple parts of a program. More than one program module can specify data for a common block, but if a conflict occurs, information from later programs is loaded on top of previously loaded information. A program may declare 0 to 125 common blocks, which can be either labeled or blank. (2) The C language `global data` items generate both a common block and an entry point.

data loading The process by which a loader inserts data into object module blocks. Occurs explicitly in response to program statements, such as the Fortran `DATA` statement or C language data initialization operation. Implicit data loading of locations within a subprogram code block can also occur if the compiler or assembler so dictates.

DEX Distributed EXpression table. A DEX contains many expressions that are evaluated at load time. These expressions are used for many purposes. A prominent use is relocation logic.

distributed mode In PVM message passing, distributed mode handles communications between a Cray MPP system and a Cray PVP host system.

entry point A location in a program or routine at which execution begins. A routine may have several entry points, each serving a different purpose. Linkage between program modules is performed when the linkage editor binds the external references of one group of modules to the entry points of another module. See also *absolute binary module*, *object module*, and *loader*.

events Events record the state of a program's execution (for instance, whether or not it has accessed data yet) and communicate that state to other tasks.

executable program The result of the load process. The executable program is a memory image built from the submitted object files and libraries that can be loaded into memory and executed. The default file name for the executable program is `a.out`.

external reference A reference to an entry point defined outside the referencing module. Fortran `CALL` statements and function calls generate external references. The CAL `EXT` pseudo instruction indicates an external reference. C procedure calls and `extern` statements generate external references.

floating The process by which the loader assigns movable blocks to segments.

force-loading The inclusion of a module that has no callers (for example, force-loading is performed on `BLOCKDATA` modules). The `FORCE` directive enables the force-loading of all uncalled entry points.

Global Symbol table The Global Symbol table is appended to the executable program, and contains information describing the modules, local blocks, common blocks, and entry points included in the program.

heap A section of memory within the user job area that provides a capability for dynamic allocation. See "`HEAP` directive," page 87, or see the heap memory management routines in the *Application Programmer's Library Reference Manual*, publication SR–2165.

include To make an object module encountered in an object file or library a part of the executable program.

initial transfer address The entry point at which your program begins execution.

library A collection of functions, or routines, that are functionally related, are called from within programs, and perform commonly used tasks. They are not operating system functions. Library functions let you use code that is already written (you do not have to reinvent wheels), make programs less complicated, and make changing programs easier. The loader includes any module in the library in the executable program only if one of the entry points in the module satisfies an external reference from another module included in the executable program. A library usually is built by a library maintenance tool, such as `bld`(1) or `ar`(1). The file name typically ends with `.a`, and the library is sometimes referred to as a `.a` file.

loader Generic term for the system software product that loads a compiled or assembled program into memory and prepares it for execution.

magic number A number UNICOS uses to identify the type of a file.

module (1) A hardware module is the basic building block of Cray Research systems; modules are made of cold plates and printed circuit boards, and fit into the mainframe chassis. (2) A software module is the basic building block of the IOS-E operating system. (3) A Fortran 90 program module is a program (or function) that contains or accesses definitions to be accessed by other program units.

movable block A module or common block not assigned by a segment description directive to a specific segment but assigned by SEGLDR to the highest-level segment that precedes all callers.

object module The executable binary program that SEGDLR produces.

ordered duplicate selection A method of selecting one of several duplicated entry points found in libraries. SEGLDR locates the first module that references the duplicated entry point and then looks for a definition of the symbol in succeeding modules. The first definition found in a succeeding module is the one used. If SEGLDR finds no succeeding definition, the first definition encountered anywhere is used.

partition (1) A contiguous set of blocks on a logical device that holds a file system. A partition of a logical device corresponds to a slice on a physical device. In file allocation, partitions permit the distribution of files across the physical devices underlying the logical device on which a file system is mounted. (2) A whole or partial disk unit that consists of an arbitrary number of consecutive tracks on a physical disk device.

primary entry point An entry point specified by the Fortran or Pascal `PROGRAM` statement, the CAL `START` pseudo-op, or the C main function; it serves as the default transfer address for the program. The first primary entry point encountered is the default transfer address.

PVM Parallel virtual machine. The message-passing model used by the Cray MPP system. It supports message passing between PEs working on the same application on the Cray MPP system, between the Cray PVP system and the Cray MPP system, and among other combinations of systems (including workstations).

relocatable binary module A binary module that cannot be executed because absolute machine addresses have not yet been set by the loader/linker; addresses are still only relative to others in the module and therefore, they can be relocated to anywhere in hardware memory.

root segment The segment that occupies the root node of the segment tree; always resides in memory during program execution.

SDT Segment Description table. The table is constructed by the loader and is included in every segmented program. It describes each segment included in the program.

sector A unit of disk storage space equal to 4096 bytes (a physical disk area that can store 512 Cray words). It is the smallest unit of transfer to or from a disk drive. The term block is often used rather than sector when discussing the concept at a high level. However, when disk storage space is meant, the term sector is used. See also *block*.

segment (1) A single node in the tree structure of a segmented program. (2) A 512-word (minimally) piece of the channel buffer that is allocated by a system's `getseg` code, at the request of the MUXIOP; used for system service requests such as central memory peek or poke executed from the OWS-E. (3) A part of a TCP data stream sent from TCP on one host to TCP on another host. Segments include control fields that identify the segment's location in the data stream and a checksum to validate the received data.

Segment Description table See *SDT*.

Segment Linkage table See *SLT*.

SLT Segment Linkage table. The table is constructed by the loader, and is included in every segmented program. The SLT describes the inter-segment linkages in the program.

special purpose program A program that will not run under control of the UNICOS operating system. Examples of special-purpose programs include the operating system kernel, or stand-alone diagnostics programs.

stack (1) A data structure providing a dynamic, sequential data list that can be accessed from one end or the other; a last-in, first-out (push down, pop up) stack is accessed from just one end. (2) A dynamic area of memory used to hold information temporarily; a push/pop method of adding and retrieving information is used.

static memory Memory that is not on the stack or not in the heap.

transfer entry point The primary entry point that will receive control from the system initialization routine when the program begins execution.

tree trimming The process by which SEGLDR eliminates modules that are not referenced in the executable program.

unsatisfied external reference An external reference for which no entry point of that name can be found in any of the object modules scanned by the loader.