UNIVERSITY OF EDINBURGH
Edinburgh Regional Computing Centre

User Note 33

(September 1985)

Title:

EMAS 2900 Fundamentals
Object file format, Stacks, GLAs, Logging-on, Loading

Author: Neil Hamilton-Smith

Contact: Advisory service

Software Support Category: A

Synopsis

This Note, which is an extract of the complete loader manual, describes some of the loading and linking activities that take place when a user logs on to EMAS and then executes a command or program.

Keywords

General Linkage Area (GLA), Logging-on, Object file format, Stacks

Loading fundamentals -Object file format, Stacks, GLAs, Logging-on, Stack switching

2900 Object File Format

The basic task of the loader is to load executable files i.e. EMAS 2900 object files. These are normally generated as the primary output of the various language compilers available on EMAS. The files conform to a standard format so that, subject to any parameter passing restrictions imposed by the various languages, cross-calling between modules written in different languages is possible.

The object file created by ERCC 2900 compilers using LPUT may contain up to seven areas in addition to red tape and linkage information, laid out in a contiguous area as below:

	area code
standard file header	
code	1
gla	2
plt	3
sst	4
ust	5
initcmn	6
initstack	7
linkage data	
object file map	

The standard file header consists of eight words and is described in User Note 35.

The generated object file may have up to seven areas, as defined below, in addition to the linkage information generated by LPUT. The content of each of these areas, any of which may be empty, is at the compiler writer's discretion, subject to the comments below.

area

1 code — should ideally contain only executable code and constants accessed only from within the code area, thus enabling a connect mode of execute, shared. As this mode apparently inhibits the most efficient method of access to constant strings and vectors the connect mode, initially at least, will be execute, read, shared. It is anticipated that connection in execute only mode will be considered essential in some instances.

2 gla (general linkage area) normally contains descriptors for accessing external objects, entry descriptors and static (normally scalar) data. The connect mode is read, write, unshared. The first eight words have a prescribed use (see below).

3 plt (procedure linkage table) may be used to contain entry and reference descriptors. If a plt exists then all such descriptors must be contained in it, rather than in gla, which may still exist to contain static data. The connect mode is read, unshared. The first eight words have a prescribed use (see below).

4 sst (shareable symbol tables) is expected to contain information relating to run-time diagnostics and is connected in read, shared mode.

5 ust (unshared symbol tables) normally contains static arrays and is connected in read, write, unshared mode.

6 initcmn (initialised common areas) is an accumulation of separately specified and initialised common areas connected in read, write, unshared mode.

7 initstack (static initialised area on stack) may be used to contain local data and data descriptors. This area should not normally be used for arrays as total stack space is constrained.

The first eight words of the gla, or plt if it exists, are currently defined as follows:

word 0	code descriptor to the	
1	first or only entry point	
2	address of ust area	
3	address of sst area	
4	byte 0 language flag	1 IMP
		2 FORTE
		3 IMPS
		4 NASS
		5 ALGOL
		6 optimised code (no diag tables)
		7 PASCAL
		8 SIMULA
		10 FORTRAN 77
		11 C
	byte 1 compiler version	
	byte 2 compiler options	
	byte 3 (may be language dependent)	
5	reserved for address of gla if plt is used, otherwise 0	
6	reserved	
7	reserved	

The linkage data contains a fifteen element array, LDATA, which provides links to records, or lists of records, also held in the linkage data area. These records provide information about entries and references, both for procedures and data items, common areas and relocation requirements between areas. All links within the linkage data are byte displacements from the start of the object file.

Stacks and GLAs

The essence of EMAS is shareability and object file format is designed to exploit this. Many users can be executing the same object file at the same time with all the efficiency and savings in resources that implies, while each has his own unique values of variables and run-time addresses.

Therefore an object file consists of two parts - one shareable and capable of being executed directly in place and the other unshareable and representing a database for initialisation.

To achieve overall shareability of object files, areas described by the database to contain run-time dependent items are set up by the loader in the user's private space.

The shareable part of the object file comprises two separate areas, the executable code (CODE) and the shared symbol tables (SST). The latter contains information relevant to run-time diagnostics.

The unshareable database, sometimes known as the General Linkage And initialisation Pattern or GLAP, describes five separate areas to be set up by the loader - the general linkage area (GLA), the procedure linkage table (PLT), the unshared symbol tables (UST), the initialised common areas (INITCMN) and the static initialised area on stack (INITSTACK). The first four of these are set up in the 'user GLA', which is a file specially created by the loader for each user, and contain such user dependent information as global variables, linkage tables to external objects and common areas. The INITSTACK is set up by the loader in a reserved area of the 'user stack', which is another file created by the loader for the user when required, and used exclusively for stack operations.

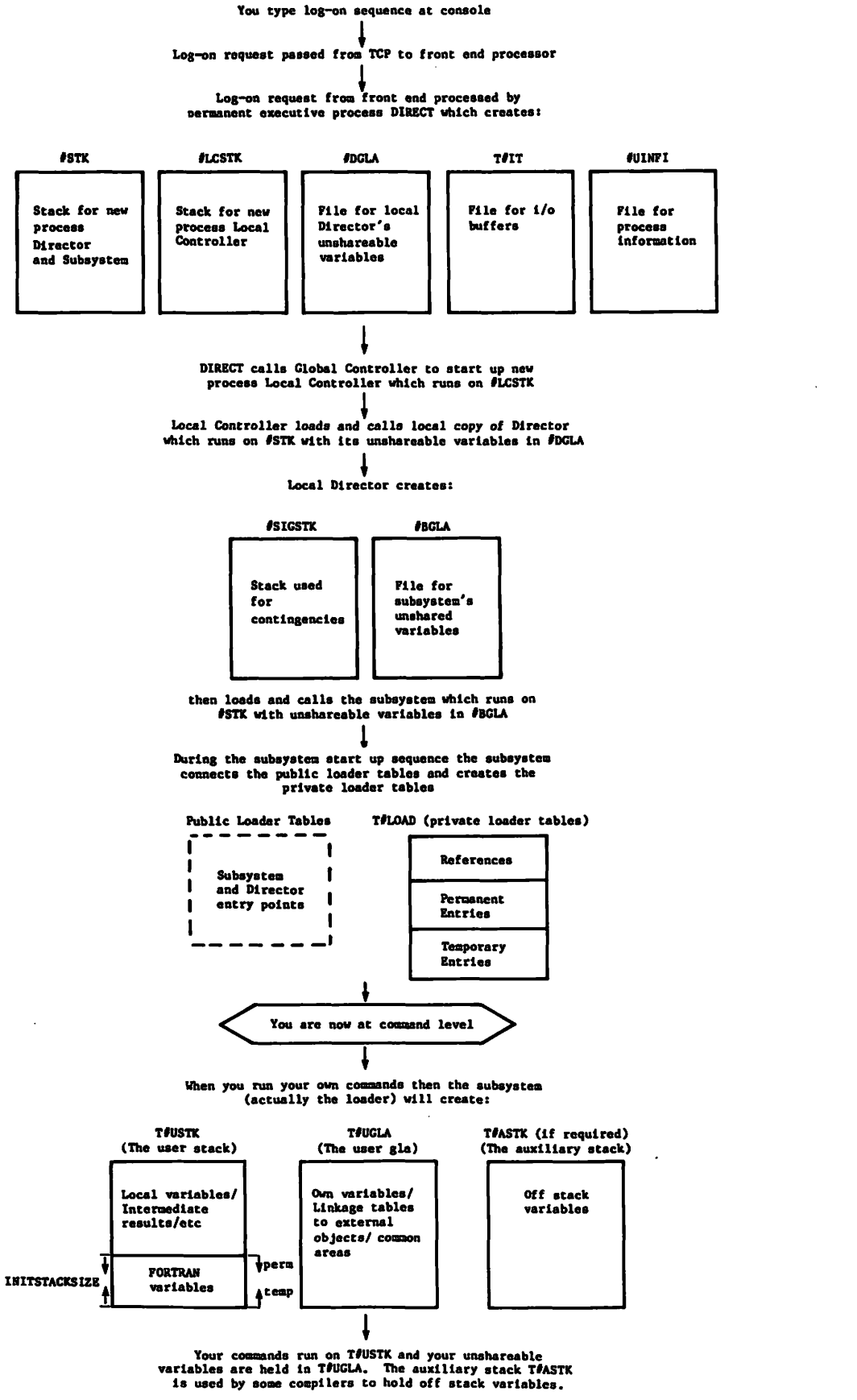
The executable code always refers to run-time dependent items in terms of offsets in the user gla or the user stack to achieve the desired generality and shareability.

Note that the unshareable areas in total are generally very small relative to the size of the (shared) object file. This is why sharing is so valuable and is one of the great virtues of EMAS object file format.

An executing object file will require access to a stack file, where local variables and the intermediate results of calculations are stored, and a 'GLA' (General Linkage Area) file, where linkage information, global variables, common areas etc. unique to that process are kept.

Diagram 1 - Creating a process

Note: File sizes are not drawn to scale



Log-on Sequence

In this section we are primarily concerned with the loading and linking activities which take place at the subsystem interface. However, it may be of some interest to place this in context by considering how a process is created and how the system protects itself against programs which, so to speak, 'go berserk'.

The log-on sequence is initiated by a request from the front end processor which is passed through the global part of the Supervisor, the Global Controller, to a permanent executive process called DIRECT. DIRECT creates several files on behalf of the embryonic process: #STK, the process or base stack on which the process Director and Subsystem will run, #LCSTK, the Local Controller stack, on which the process local Supervisor will run, #DGLA, a file for the Director's GLA, T#IT, a file used for i/o buffers and #UINFI, a file to contain information about the user. DIRECT then calls the Global Controller to start the process.

The Global Controller starts up the Local Controller for the process which loads and calls the local copy of the Director (running on #STK, GLA in #DGLA). The Director continues the initialisation sequence by creating and connecting another stack file, #SIGSTK, the signal stack, which is used when contingencies occur. When Director's initialisation is complete it connects the subsystem basefile, which is the file containing the code of the standard or required subsystem. It then creates the file #BGLA, the base GLA. The local Director loads and calls the Subsystem (running on #STK, GLA in #BGLA). After Subsystem initialisation we finally arrive at 'command level'.

It will have been observed that at each stage a given component loads and calls the next in the sequence. Each component has its own piece of code which functions as a loader. The Subsystem's piece of code is what we are calling 'the loader' in this note. It will load and call the next level up, i.e. user commands. The other 'loaders' are all short and simple since they only have one task to perform.

The Local Controller runs at a higher level of privilege than the rest of the user process and the two can be regarded as co-operating processes with the Local Controller in charge. When cpu becomes available to the process as a whole, the Local Controller has priority and runs on its own stack. When the local Director is called a stack switch is executed and the Director (and eventually the Subsystem) runs on the stack file #STK.

To summarize thus far: after the log-on sequence, Subsystem code is executing, stack operations are being carried on the base stack (#STK) and linkage information and own variables for the system are held in the base GLA (#BGLA). The situation remains thus until the first command which is not in the subsystem is called. User commands are intrinsically less trustworthy than system commands, but run at the same level of privilege. To protect the system, which is running on the base stack, a new stack, the user stack (T#USTK), is created. A stack switch is then executed which ensures that the user command runs on this stack. Return from the user command causes a return to the base stack. A catastrophic program failure which corrupts the user stack will probably not therefore corrupt the subsystem's stack.

The 2900 hardware provides particularly efficient access to a subroutine's local scalar variables which are stored on the currently used stack. The space occupied by such local variables is de-allocated at exit from the subroutine, and hence their values are lost. In FORTRAN programs, the language definition guarantees preservation of the values of local variables between calls of a given function or subroutine, and hence a different location must be allocated for them. The user stack has a hardware imposed upper size limit of 252 Kbytes; of this 252 Kbytes, a portion, called the initialised stack area, can be reserved for such variables by

using the OPTION INITSTACKSIZE= command (in Edinburgh the default reserved area is currently 100 Kbytes). This area, though part of the stack, is essentially static; normal stack operations take place between the top of the initialised stack area and the top of the stack.

The loader distinguishes between routines which are to be 'permanently' loaded (i.e. those which are to remain loaded to the end of the current session or the first call of RESETLOADER) and those which are only loaded until the end of the current command. To avoid fragmentation of the initialised stack area, therefore, 'permanent' initialised stack is taken from the top of the area and temporary initialised stack from the bottom (see diagram 1). The GLA requirements of permanently loaded files are taken from the basegla for convenience but a call to load the first temporarily loaded object file triggers off the creation of the user GLA, T#UGLA. This file is used to satisfy the GLA requirements of all temporarily loaded object files - except bound object files which are treated somewhat differently (see User Note 32).

Acknowledgements

This edition of the Note consists of the text originally written by the late Colin McCallum, and extracts of material written for another purpose by Geoff Millard. All enquiries should be Mailed to the editor, 'N.Hamilton-Smith'.