SETL Newsletter 100
Making SRTL Debugging Runs

February 20, 1973
Hank Warren

This note describes the present manner of making debugging runs involving the SETL run time library. The setup of a run is complicated because of the existence of global macros and variables, and because the present LITTLE compiler has no provisions designed to avoid recompiling a complete program to test any change to the program, no matter how small the change may be. The purpose of most of the complications is to circumvent the latter deficiency.

There are two files of central importance:

SETLLIBPL, and
SETLLIBBIN.

SETLLIBPL is a random file, or "program library," and it contains the source text of:

1. A list of the SETL run time library routines,
2. The complete SETL run time library, and
3. A set of test routines for exercising the SETL run time library.

SETLLIBBIN is a binary file, and it contains, in relocatable binary form:

1. Many (but not all) of the routines in SETLLIBPL, and
2. All the routines from another file called LITTLELIB.

Within SETLLIBPL, the first deck, INDEX, contains the list of all the SETL routines, with a one-sentence description of its function and an indication of its current status. The "status" indicates whether the routine has been tested, or merely coded, and whether or not it has been specified in SETL, etc. (Much of the run time library is specified in SETL; see SETL Newsletter 49.) Deck INDEX will not normally be compiled; however it is surrounded by commenting brackets so

so that it can be; this is one way to obtain a listing of it. Deck INDEX is arranged in the same order as the routines in the library itself, and hence aids in finding things.

The next deck in SETLLIBPL is named MACRO, and it contains all macros that are either of global significance or involve machine dependent code. This amounts to about 99% of the macros.

The third deck is named START, and it contains a single subroutine of the same name. This subroutine must be entered first in all runs involving the run time library. It initializes various variables and sets various constants that cannot be preset with a DATA statement. Subroutine START also contains the SIZE and DIMS statements for all global variables, and a small amount of information, some occurring in DATA statements,that is machine dependent (all machine dependent details in the SETL run time library are confined to decks MACRO and START).

At the end of subroutine START there occurs the statements CALL SETLMPG; CALL EOJSTAT; CALL EXIT. The routine that the SRTL user wishes to get control first (after START) must be named SETLMPG (for "SETL main program;" ultimately to be generated by the SETL compiler). SETLMPG should terminate via RETURN. If it terminates by CALL EXIT, or by executing an END statement (which is the same thing), or by CALL ABORT, then "end of job statistics" available through EOJSTAT will not be printed out. In addition, any termination activity that might be implemented in the future, such as the flushing of output buffers, will not be done. Alternatively, the user's program may terminate by the statement ERRSTOP;. This macro presently expands into CALL EOJSTAT; CALL ABORT; and hence allows one to terminate with an error code set (by ABORT). The error code will be recognized by the operating system and

will cause skipping ahead to an *FIN card. It may also be tested for by a statement such as *IF(.NOT.ERROR(ANY))*TERM., which might be followed by DMP cards.

Use of Pre-Compiled Binary

To make a test run of a program (e.g., one of the SRTL routines) that uses the run time library, it is possible to compile only decks MACRO, START, the new program, any SRTL routines that are being modified, and a small deck named LAST. The compiled code which results can then be combined with SETLLIBBIN by the loader; if duplicate subroutine names occur, only the new versions will be loaded. Proceeding in this way reduces the CP time necessary for a SRTL debug run from about 300-400 seconds (a figure that is increasing as the run time library evolves) to about 70-150 seconds (a figure that will remain about the same). This enables the user to get three to five runs per day instead of one or two. Before describing how to do this, however, we will first explain how SETLLIBBIN is made from SETLLIBPL.

A complete deck setup for making SETLLIBBIN is attached. It consists of three main phases:

1. an UPDATE to select the desired portions of SETLLIBPL and convert them from random to sequential form,
2. compilation, and
3. a final phase combining the appropriate binary decks into a single file, and cataloging it as SETLLIBBIN.

Most of the deck setup is self-explanatory, but I will explain a few points.

The lexical scanner (LTLLEXF) is executed with the listing control initialized to 00 (print nothing except error messages). The source text will be listed by the compiler itself (LITTLE). After executing the lexical scanner, a small portion (three lines) of the macro-expanded text is printed out (COPYSBF(EXPANDED,OUTPUT)). This text is subroutine LASTSUB (deck LAST), and examination of it in expanded form reveals the maximum values reached by counters ZZYZ and ZZYY, which are used in the TACK and PTR macros. The use of this information is explained below.

When executing LITTLE, we must set the line count to 20000 or so, as the default value of 10000 (octal) is not sufficient.

The binary output of LITTLE is on TAPE3 and TAPE5. The next steps make a file consisting of

TAPE3,
TAPE5, and
LITTLELIB.

Normally these could be combined by COPYBF. However, the present version of LITTLE produces each 17 words as a separate binary record, with the result that the file occupies a very large amount of disk space; so much so, in fact, that SCOPE refuses to catalog the file. This problem is circumvented by using RECMRG to combine the files. Note that RECMRG copies from the second parameter to the first parameter; a statement such as RECMRG(NEW,A,B,C) combines A, B, and C onto file NEW. Between the calls to RECMRG, backspace commands are needed to get rid of file marks.

Incidentally, it is planned to improve LITTLE in a very short time so that it writes out its binary files in a more

compressed manner. This will avoid the inefficient use of the disk when creating the temporary files TAPE3 and TAPE5.

After SETLLIBBIN is cataloged, it is loaded with MAP(PART) to obtain a load map (NOGO. must also be specified or the map will not be produced). The map is desirable for desk-checking purposes and also because we are interested in the size of common block START, as is explained below.

The remainder of the deck consists of UPDATE directives. The *COMPILE directives select the decks that we wish to be compiled. We do not select all of the run time library, because if we did, jobs using SETLLIBBIN would require a much larger region size. The region size required at present is in the neighborhood of 150,000 to 170,000 (octal) words, depending upon how much new code is added. It is important to keep this figure below 200,000, because 200,000 is a critical point in SCOPE's scheduling algorithm, and jobs exceeding that figure suffer substantially increased turn-around time (as they should).

The routines presently included in SETLLIBBIN are those that are necessary to build SETL objects, including sets and tuples, and print them out. This requires a number of auxiliary routines, notably EQUAL and NEXT (the iteration routine is used to print sets). The garbage collector is also included, as are subroutines START and LASTSUB. START must be included in this compilation to define the global variables used by the other routines, and LASTSUB is included merely to obtain the value of the counters used by the TACK and PTR macros. As will be seen, START and LASTSUB will not actually be loaded from SETLLIBBIN. Some routines that are not presently being included in SETLLIBBIN are ICNV (input conversion routines), ATOM, NELT (SETL #x), ELMT(SETL x ϵ S), ARB, OF, SOF ("storage

OF," i.e., f(x)=y), HEAD, TAIL, TYPE, PAIR, most Boolean operations, and most integer arithmetic operations. The user who wishes to employ these routines must re-compile them for every run, or alternatively make his own binary library.

Incidentally, it should be pointed out that SETLLIBBIN is not a "library" in the SCOPE sense of the word, i.e., it is not created by the GENLIB utility. However, this facility may be used at a later time, if LITTLE is ever fixed up so the binary decks it puts out can be loaded with the MACE loader. Doing so will permit building a library that includes binary decks for ATOM, NELT, ELMT, ARB, OF, etc., and then by using the MACE loader they will only be loaded if referenced. Then jobs that do not use these routines will run in a reasonable region size, and jobs that do need them will not have to compile them, and we will not be faced with the administrative nuisance of maintaining several versions of the library.

What is really needed to further improve the situation is a loader capability similar to the "NEVERCALL" feature found in some loaders. This permits the user to specify that certain routines will never be called, even though they may be referenced. The loader then does not load these routines and does not automatically bring in routines that they reference.

The deck setup shown includes three *D (delete) UPDATE directives. These delete certain subroutines that are not wanted, even though other subroutines in the same deck are needed. For example, from deck PLUS we delete ADDI (add integers), and retain CONCATC and other routines (CONCATC is used by the print routines).

To make a test run using SETLLIBBIN, you must set up a run in which the following is compiled:

1. deck MACRO,
2. deck START,
3. any routines from SETLLIBPL that you need and that were not included in SETLLIBBIN,
4. any routines from SETLLIBPL that you are revising,
5. new routines, and
6. deck LAST.

Deck MACRO must be included because your new routines will in all probability use some of the SRTL macros (such as THEN, ELSE, ENDIF, and BUMPZZ10). Deck START must be included because your new routines will in all probability use some of the SRTL global variables (in fact, deck LAST does). Deck LAST must be included so the proper values will be assigned to two global variables MAXZZYZ and MAXZZYY, which are needed by the run time library.

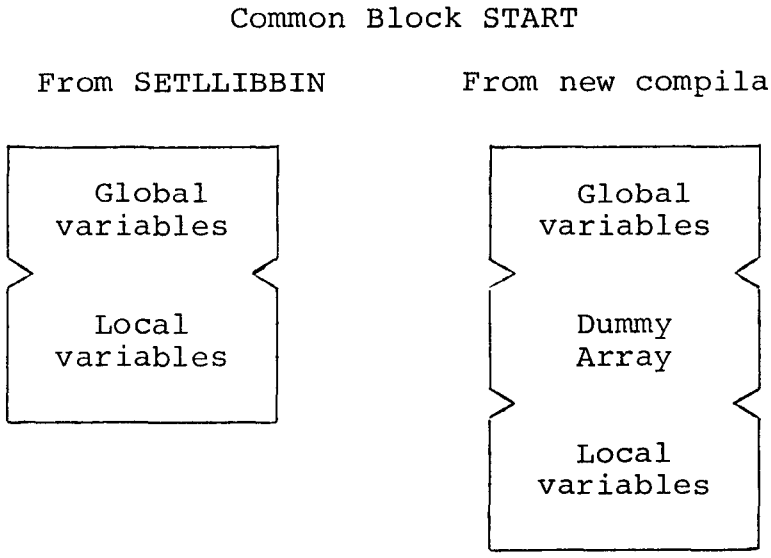
This would be a straightforward matter except for two difficulties. The first is that the run time library has a need to map certain variables into certain array locations. That is, we wish to accomplish something like the FORTRAN statement EQUIVALENCE (X,A(1)), (Y,A(2)), (Z,A(3)). This is done by the TACK and PTR macros, in conjunction with "counters" ZZYZ and ZZYY (see SETL Newsletter 73, User's Guide to the SETL Run Time Library, section 3.2). If the new code contains any TACK or PTR macros (as it usually will), we must assure that the corresponding variables are assigned locations <u>following</u> those used by other SRTL routines. This may be accomplished by artificially incrementing counters ZZYZ and ZZYY to some value at or above the maximum attained when the library was made. To do this easily, two macros called BUMPZZ10 and BUMPZY10 are included in deck MACRO. Invoking these macros causes the corresponding counters to be incremented by ten.

For example, when SETLLIBBIN cycle 6 was made, it was found that ZZYZ was 48 and ZZYY was 32. Many of the TACK- and PTR-variables are located in START, and hence it should be sufficient to increment ZZYZ by 40 and ZZYY by 30 (or 50 and 40 could be used to be sure). This is done by the four invocations of BUMPZZ10 and three of BUMPZY10 shown on the last page of this note. These must be included at the end of START and before the compilation of new or revised code.

The second difficulty alluded to above is due to the fact that LITTLE maps all variables into a single COMMON block. This block has the same name as the first compiled subroutine or function: START, in our case. Since some of the global variables are both set and used (e.g., T, the top of stack pointer, and STORAGE, the dynamic storage array), we must assure that code in the precompiled library and the new code reference the same location for the same global variable. This means that (1) we must run with only one copy of common block START (a fact which the loader insists on anyway), (2) the precompiled library and the new code must be compiled with identical versions of subroutine START (which contains the SIZE and DIMS statements of all global variables), and (3) we must assure that local variables in the new code are mapped into distinct locations from local variables in the precompiled code. This last requirement is met by placing a large dummy array at the end of subroutine START and before any new code, as shown on the last page. The size of the array must be at least as large as the size of common block START in the precompiled library, minus the size contributed by variables that are defined in subroutine START. A value of 30,000 (octal) is used in the attached sample run, but actually 20,000 or even less would suffice.

The common block START from the new compilation must, of course, be used.

The picture below will help to visualize the situation.



The above scheme would not work if any local variables in SETLLIBBIN were initialized with DATA statements, because the left block above is not loaded. All DATA statements in the SETL run time library are confined to deck START (there are also some in deck TEST, the SRTL test routines, but this is safe as they are not precompiled ... at least not at the present time).

Most of the remainder of the deck setup on the last page should be clear. Note that CHLISTSTAT (change listing status) is used to suppress printing of decks MACRO and START. There are at present no CHLISTSTAT or $SETLISTC_{xx}$ statements in SETLLIBPL, and it probably should be kept free of these to avoid hopeless confusion. There are already a sufficient number of opportunities for error in the way we are working.

```
J117306,CM152500,DT350,       HANK WARREN   MAKE SETLLIBBIN.
RFL(40000)
ATTACH(OLDPL,SETLLIBPL,CY=16)
UPDATE(Q,B,L=A12)
RETURN(OLDPL)
*TIME.
MAP(OFF)
ATTACH(LEXDO,LTLLEXF)
RFL(1525000)
SETCORE.
LEXDO(COMPILE,OUTPUT,EXPANDED,BINFILE) (SL=00)
RETURN(LEXDO)
REWIND(EXPANDED)
COPYSBF(EXPANDED,OUTPUT)
*TIME.
ATTACH(NGO,LITTLE,CY=6)
REWIND(BINFILE)
NGO(LC=20000,BINFILE)
*TIME.
MAP(PART)
REWIND(TAPE3,TAPE5)
ATTACH(LLIB,LITTLELIB,CY=1)
RECMRG(SLIB,TAPE3)
BKSP(SLIB,1)
RECMRG(SLIB,TAPE5)
BKSP(SLIB,1)
RECMRG(SLIB,LLIB)
CATALOG(SLIB,SETLLIBBIN,PW=SETL,RP=999,CY=6)
LOAD(SLIB)
NOGO.
BATCH(OUTPUT,SITE17,END)
E=O=R
*IDENT TEMP
*D PLUS.32,242
*D SUBS.29,125
*D MISC.109,185
*I LAST.2
          SETLISTC10
*COMPILE MACRO.GENS
*COMPILE OCNV,HASH
*COMPILE EQU
*COMPILE WITH,LESS
*COMPILE NEXT
*COMPILE COPY
*COMPILE PLUS
*COMPILE SUBS
*COMPILE MISC
*COMPILE LAST
E=O=F
```

```
J117306,CM170000,DT100,      HANK WARREN   TEST LESS AND ARB.
RFL(35000)
ATTACH(OLDPL,SETLLIBPL,CY=16)
UPDATE(Q,8,L=A12)
RETURN(OLDPL)
*TIME.
ATTACH(LEXDO,LTLLEXF)
MAP(OFF)
RFL(151000)
SETCORE.
LEXDO(COMPILE,DUMMYF,EXPANDED,BINFILE) (SL=00)
RETURN(LEXDO)
*TIME.
ATTACH(NGO,LITTLE,CY=6)
REWIND(BINFILE)
NGO(LC=20000,BINFILE)
RETURN(NGO)
*TIME.
ATTACH(SLIB,SETLLIBBIN,CY=6)
MAP(PART)
RFL(170000)
SETCORE(65766,66666,66666,66666)
LOAD(TAPE3)
LOAD(TAPE5)
SLIB(OUTPUT)
*FIN.
E=0=R
*IDENT TEMP
*BEFORE MACRO,2
          CHLISTSTAT           /* TURN OFF LISTING. */
*I START.252
          CHLISTSTAT           /* TURN ON LISTING. */
          BUMPZZ10 BUMPZZ10 BUMPZZ10 BUMPZZ10 MACDROP(BUMPZZ10)
          BUMPZY10 BUMPZY10 BUMPZY10 MACDROP(BUMPZY10)
          SIZE DUMMYARRAY(WS); DIMS DUMMYARRAY(12288); /* 30000 OCTAL.*/
*D TEST,15,20
*D TEST,23,26
*D TEST,29,300
*D TEST,397,933
*D TEST,1138,1679
*COMPILE MACRO,START
*COMPILE ARB
*COMPILE TEST,LAST
E=0=F
```