TECHNICAL NOTE FOR SIMSCRIPT USERS

W. V. Neisius, E. D. Katz, and D. B. Townsend
TRW Systems Group
Redondo Beach, California

SIMSCRIPT is a very powerful and versatile language, however those who have used it will probably agree that its flexibility creates problems in learning to use it properly--at least, that was our experience. The purpose of this paper is to present our experiences and to share some of the things we have learned.

This paper will be divided into three sections:

- Special Techniques
- SIMSCRIPT vs SIMSCRIPT I.5
- Computer Comparisons

SPECIAL TECHNIQUES

Events vs Activities. An activity has a starting time and a duration. In Simscript it is necessary to use separate event notices for the beginning and end of an activity. Beginning Simscript programmers frequently, when entering the routine representing the beginning of an activity (BEGIN) will-

1. Create an event notice (END) to call the routine for the end of the activity,
2. Transfer attributes from BEGIN to END,
3. Destroy BEGIN,
4. CAUSE END AT---etc.

The first three steps can be omitted by the single statement:

CAUSE END CALLED BEGIN AT---etc.

This technique can also be used for the beginning of the next activity (NEXT) by

CAUSE NEXT CALLED END AT---etc.

Subscripting Temporary Attributes. Sometimes it is useful to be able to treat the attribute of a temporary entity as a single dimensioned variable in order to index over several of the attributes. This can be accomplished by writing ATT(TELV+K*I) where ATT is the name of the attribute, TELV is the Temporary Entity Local Variable name, I takes on the value from 0 to a maximum of 7, and K is a constant dependent upon the computer and Simscript version.

Thus K = -1 for 7094 Simscript I,
K = +1 for 7094 Simscript I.5 and
K = +8 for 360 Simscript I.5.

Another procedure, which has been used, is to let the temporary entity own a set of temporary entities and index over the members of the set.

Automatic Max Time by System. One of our frustrating early experiences was the running of a long job, of indeterminate duration, only to be maxtimed just before the printing of our final summary reports. On our particular system, we were able to do two things. First, we inserted the address of a "wrap-up" routine at the Max Time Exit location, and second we read the system clock and printed elapsed time on the extreme right of our paper at various program check points. This later gave us an indication of those portions of the program where we could efficiently spend our time in program speed-up.

Error Stops. When Simscript detects an error, during execution, a cryptic remark is printed and execution is terminated. Sometimes the only way to track down the error is to rerun the program with a sprinkling of suitable "trace" statements. Our approach has been to introduce a new routine with the name EXXIT which deletes the existing routine and calls for a dump and any other print outs which might be useful.

Besides adding dumps, other error procedures have been changed. For example the "computed go to out of range" error was modified to preserve the location of where the error occurred.

Initial Conditions Data Deck Problems. Simscript reads each Initialization Card until an error is found and then stops. Too often, there are multiple errors in the deck which are uncovered one by one. Aside from the obvious solution of not making mistakes there are a number of approaches. For example, the TRANSIM program (written in Simscript by UCLA) uses a special IBM 1401 program to analyze the Initialization Deck according to the special TRANSIM requirements. This is usually very efficient in flagging multiple errors in what is usually a very large and complex deck. There are other approaches which are also useful. There are essentially two types. The first is to decide upon certain maximum array sizes and zero everything out. Then use an Exogenous Event to read in the Initial Conditions. By printing a card image of each card read, we have a record of our initial conditions. This has the further advantage of permitting the attributes, associated with a given permanent entity, to be read in as a group and then distributed over all of the appropriate tables.

Some versions of Simscript provide for printing a duplicate of the Initialization Deck. It has been our approach to use a series of

reports to consolidate the Initial Conditions as part of the documentation of the particular run and for verification.

SIMSCRIPT vs SIMSCRIPT I.5

The following comments relate to Simscript on the IBM 7094 except where noted.

Deck Names. One important difference between the two Simscript versions concerns the two compiled binary decks. In the original Simscript (hereafter referred to as I), a separate binary deck is created for each routine, report, event, definition, etc., and, in addition a complete SYSTEM deck is provided. In I.5, binary decks are punched for all definitions, but all other routines are lumped together into one deck in standard IBJOB fashion. No SYSTEM deck is punched. In order to modify a routine, it is necessary to compile it with a new deck name. When running, the first routine encountered will be used, and the same routine in other decks will be deleted. This procedure sounds simple enough, but unfortunately led to many problems. Although each problem is minor, the sum total leads to the conclusion that perhaps the old method was better.

Deck Size. You would think that since I.5 omits the SYSTEM deck, the compiled binary deck would be smaller. This is not so. A previous I program was converted to I.5 with the following results. Deck sizes are given in inches.

	Source Program	I	I.5
SYSTEM Deck	0.0	1.8	0.0
Compiled Definitions	.8	1.7	4.8
Compiled Programs	6.6	3.5	6.1
TOTAL	7.4	7.0	10.9

The initialization and data cards now make the I.5 program a "two-box" program! (Most of this increase is due to standard IBJOB control cards.) Furthermore, as individual routines are recompiled, the total deck size increases unless all programs within a given deck are also recompiled.

Suppose you decide to divide your original source program into many small decks! Then you must include the definition deck in each deck. This is not mentioned in the documentation. In the above program we first decided not to duplicate the definitions, and compile all source programs into one big deck--but that did not work--we exceeded the maximum size for a single deck and ran out of Control Dictionary space.

Finally, we got the program compiled into three decks and then we ran into our next problem.

Hollerith Output. I.5 offers the convenience of simple write statements containing Hollerith outputs. Instead of always using reports, it is possible to output simple one-line statements. But here is the problem. Assume a deck contains a number of routines and each has Hollerith output. When compiling the deck, I.5 stores all Hollerith data for all routines at the end of the deck.

If later one of the routines is recompiled and deleted from the deck, then the remaining routines ahead of that routine will now lose their pointers to their stored Hollerith and will print out garbage.

The net result is that in order to preserve the output, it may be necessary to recompile routines containing Hollerith even though they are not modified.

Memory Requirements. Another problem arose when converting a fairly large Simscript I program. This program had about 8K of memory left for creating Temporary Entities and Event Notices, not counting the memory released by the Initializing routine. The I.5 version was about 500 words short in storing permanent arrays, thus the I.5 program required about 40% more memory on the 7094. (This increase is made up of the following items:

Routine	Words
Simscript System	1200
Library Routine and Buffers	3800
System Resident	1400
	6400

In addition, reports require about 1/3 more memory. To offset this, however, I.5 releases 1100 more words for temporary storage than Simscript I).

Miscellaneous Comments. In our experience with I.5, CACI has been very efficient in issuing modifications and responding to the few bugs we have uncovered. There is little point in discussing problems which no longer exist, except one of them caused so much trouble, it deserves to be mentioned. This is related to "N'Attribute Name". We first encountered this in the CDC 3600 Simscript I.5 when we found that this was not automatically defined. One part of the CDC manual stated that it was defined, while another part described the proper procedure for programming in its absence. When we first used the CACI version of I.5 for the 7094, we found that it was okay for a fixed point ragged table but not a floating point ragged table. We attempted to program around this by using the same method as used for the CDC version. Unfortunately, this fix does not work for a "packed" table and since this was a repetition statement in a report generator, we got over 2000 pages of paper with nothing but a title and a page number before being--mercifully--max timed! The moral of this

is that those using non-maintained versions of I.5 (perhaps the SHARE version) should watch for this bug.

COMPUTER COMPARISONS

On the IBM 7094 we have run Simscript I and Simscript I.5. In addition, Simscript runs have been made on the IBM 360, the CDC 3600 and 6500, and the UNIVAC 1108. It has become apparent that certain factors, other than basic machine speed, have a major influence upon running time. The most important factor appears to be the operating system; second, the particular Simscript version; and finally, the ratio of input/output to actual computing and the total length of the job. A few examples will illustrate some of these points. Our original IBM 7094 Simscript I.5 was combined into a IBSYS system tape not compatible with our normal system. Tape searching for Simscript routines and return to IBSYS introduced a fixed overhead of almost two minutes per run. This was true, even when running a previously compiled job since the Simscript system routines are on the tape and not part of the input deck, as in Simscript I. By editing this tape, the overhead was reduced to about 1.4 minutes. By contrast, on the CDC 3600, since Simscript was on the disc as part of the operating system, typical load times were 15 to 20 seconds. A second example relates to our experiences with the IBM 360-65. We understand that Simscript I.5 can be reconfigured for various memory sizes. The available memory, on our 360, is about 10 K bytes short of optimum and a substantial penalty is imposed in compiling.

A simple SIMSCRIPT program was run as a baseline problem on a number of computers. The following table lists the results in seconds.

	IBM 7094-I5	IBM 360	UNIVAC 1108	CDC 6500
1. Setup & Source Deck Copy	80	8	14	0
2. Compile and Assemble	33	213	29	56
3. Load and Execute	15	70	3	3
TOTAL SECONDS	128	291	46	59
4. Time to run compiled program (1+3) in seconds	95	78	17	3

This table, based upon only one simple case, must be read with caution. However, some general observations can be made. The major penalty for the 7094 is Item 1, modifications in operating system could greatly improve this time. The inefficiency of the IBM 360 version is due to non-optimum memory size, however, we do not understand how this could make Item 3 so large. The major time penalty for the CDC 6500 is assembly time.

SUMMARY

A paper such as this, presenting a collection of difficulties, is apt to discourage potential users of Simscript and give the impression that Simscript is not a good language. On the contrary, we are enthusiastic supporters of Simscript and believe that no available language can offer such versatility, ease of use, and rapid response for certain classes of problems. It is hoped that the presentation of this paper will encourage other Simscript users to share their experiences.