XREF

XREF displays a complete cross-reference of variable names against where-used line numbers for any Level II BASIC program. When you run a BASIC program, the BASIC interpreter builds a table of names of all the variables created during that run. XREF works down through this table, and for each variable scans the complete BASIC program finding every reference to that variable in the program. It displays

- The variable name and data type.
- All line numbers where that name occurs.

After loading XREF into protected memory, CLOAD your BASIC program and run it. Then invoke XREF, for instance by typing ?USR(0). Or branch to it by:- SYSTEM (enter) *? /N + 9 (enter)
where N is the address at which XREF is loaded.

You will now get a display showing the variables in the order they were created in the program, and their where-used line numbers. When the screen is full, XREF will wait until you hit any key, and will then display the next screenful.

Notes

1) Variables in your program that have not been set will not be displayed.
2) If a variable is referred to several times in one BASIC line, multiple references to that line will be displayed.
3) Both scalar references and array references are displayed, but array subscript values are not distinguished.
4) The data type of the variable is explicitly shown (% ! # or $). Your program may use simple unqualified names, together with DEFine statements. XREF assumes that whatever set of DEFine values applies at the end of the program, applied to the whole program. If one name is used for two different data types, e.g. A% and A$, then both are treated separately, and their individual references are displayed.
5) Character strings inside quotes and remarks are not scanned for names. Nor are DATA and DEFine statements.

Selective Display

You can find out the references for a particular variable (or variables) as follows: Add a new line to the front of your program, such as

```
0     A = 0 : STOP
```

RUN this single line, thereby clearing the BASIC name table and establishing just one entry, A . Now type ?USR(0) and you will get all the references for A and A ! in the whole program. Use a DEFine statement to control how unqualified names are treated.

How to Load and Relocate a Southern Software Machine-Language Program.

You choose the location of the program in memory, to suit your machine size. This MUST be in protected memory, or the program will not run. So, taking account of your machine size, allow enough space for the program itself, plus any other machine-language subroutines you may need, either above or below the program you are loading.

As an example, suppose you are loading Southern Software DLOAD (size 160 bytes). You have already loaded, or are going to load, TRS KBFIX at the top of memory, and Southern Software TSAVE below DLOAD. Plan your memory use as follows, working out the values (T) and (A) for your situation:

	PROG SIZE	MACHINE SIZE				
	(bytes)	4K	16K	32K	48K	
Memory limit		20480	32768	49152	65536	
Space for KBFIX	56	20424	32712	49096	65480	
Space for DLOAD	160	20264	32552	48936	65320	(T)
Space for TSAVE	512	19752	32040	48429	64808	(A)

1)Turn on the computer. If you have a DISK system, enter Level2, not DISK BASIC.
2)answer the MEMORY SIZE question with your value of (A). (On Video Genie, this value is used after READY?).
3)prepare the cassette player to load the self-relocating program.

	TRS-80	You type:
4)	>	SYSTEM (enter)
5)	*?	DLOAD (enter) or your program name
6) After tape has loaded	*?	/ (enter)
7)	TARGET ADDR?	Your value of (T)
8)	READY	

Notes:
1)At step 5 the tape will load and a pair of asterisks will blink on the display. If there are no asterisks, or two unblinking asterisks, or C*, then there has been a loading error. Stop the recorder, reset, and retry with a new volume setting.

2)At step 7, the program will relocate itself to address T. If instead of typing a value you just hit enter, then the program will relate itself to A, the answer to the MEMORY SIZE question.

3)Under Level2, after relocation, the program is ready to be invoked with a USR(n) call, since the USR address is automatically primed. However this does not work under DISK BASIC (or Level3), and you must additionally set DEFUSRn to inform the system of this routine's address.

4)Once a program has been loaded and relocated, it can be dumped to a new tape using Southern Software TSAVE, or TRS TBUG. Then it will load directly to its final location. Use of TSAVE has the advantage that several programs can be dumped on a single file, which can also preprime the USR address.

5)During step 5, the program is temporarily load into locations 18944 and up. This means that
 a)You must perform all necessary relocating loads before loading a BASIC program, or entering DISK BASIC.
 b)The final location, T, of the self-relocating program can never be lower than 18960. (Hex 4A10).

6)If you run under DISK BASIC, then perform the initial self-relocating load under Level2, as described. Then reenter TRSDOS (or NEWDOS, etc) and use the DUMP command to save the core image directly from its relocated position. Subsequently you can LOAD the core image directly, under TRSDOS. But when you enter DISK BASIC, remember to set MEMORY SIZE to leave this area of core protected, and remember that the top 64 bytes of memory are corrupted by the DISK BASIC loader, and should not be used for programs.

Hints on Tape Loading.

1) Listen to the tape to establish exactly where the data starts. Note this on the tape label.
2) Turn the volume down to zero, and "attempt" a tape load, very slowly increasing the volume until you get asterisks on the screen. Stop the tape (not the computer), note the volume level, Reboot.
3) Turn the volume up to maximum, and "attempt" a tape load, very slowly decreasing the volume until you get asterisks on the screen. Again, stop the tape, and note the vole,
4) Set the volume to slightly above the mid-point of the two extremes of volume, and attempt a real load.

Possible Tape or Recorder Faults.

1) Kink or fold in the tape. Even a minor fold may render the tape unloadable, (Southern Software tapes carry a second copy of the file, in case the first gets damaged).
2) Noise caused by RESET when tape is running, Always stop the tape before hitting RESET.
3) Being small, all the plugs are prone to intermittent error and should be protected against movement.
4) Inconsistent tracking of the tape over the head.

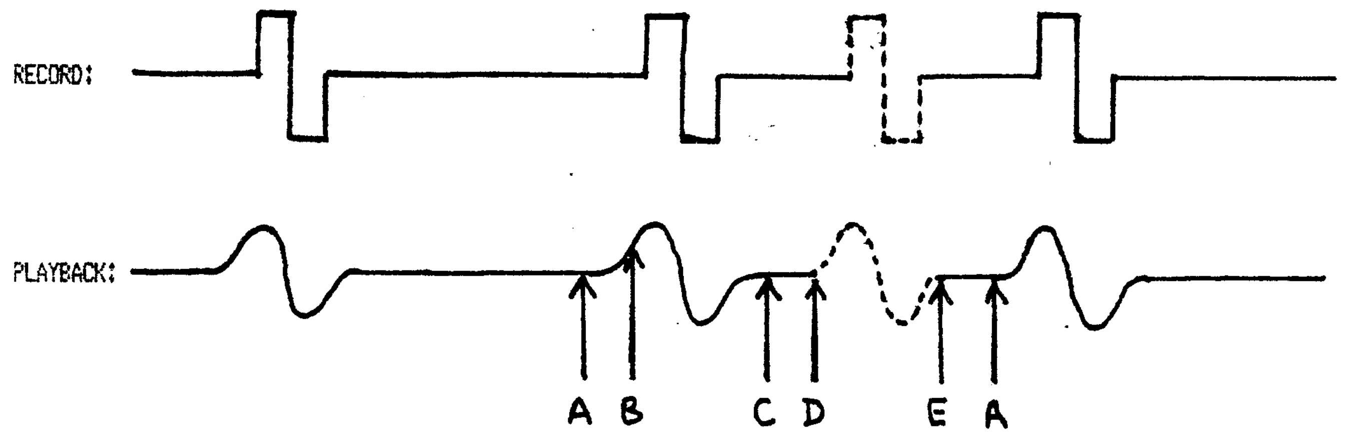
This list does not include poor tape quality, since it is very unlikely to be a problem, at the frequency bits are recorded. However, you may have found that one make of cassette seems much better than another. This is probably due to the construction of the cassette, rather than the tape. Generally more expensive cassettes run more smoothly, and therefore reduce the chance of poor tracking of tape over the head.

How DATA is Recorded and Read.

The computer contains hardware to generate an "above-and-below-zero" pulse, as shown below. This is fired by direct program control. The output routine produces one such clock pulse every 500th of a second (by looping). Data ones and zeroes are recorded as pulses halfway between these clock signals, A zero is the absence of a pulse, a one is the presence of a pulse.

The playback logic is analogous to a keyboard "debounce" routine, To read a single bit, start somewhere near (A). Loop, until the hardware recognises a signal, at (B). This is a clock pulse. Now loop until that signal is bound to have died away, and reset the hardware latch, at (C). Now wait an exact length of time, till (D), and listen for another signal, YES, then it's a one, NO, then it's a zero. In either case reset the latch after the sampling time, at (E), and loop again until the next time (A).

As you can see, the TIMER must not be running during either record or playback, since exact looping times are vital. Nor does the logic take time off to test the keyboard for the BREAK key. However tape speed is not ultra-critical, since there is a resynchronisation wires at (A) on every bit.


RECORD:
PLAYBACK:
A
B
C
D
E
A