APPLICATION		REVISIONS		
NEXT ASSY	USED ON	LTR	DESCRIPTION	DATE
	7506			

REV STATUS OF SHEETS																		
REV																		
SHEET	1	2	3	4	5	6	7	8	9	10	11	12	13	14				

RELEASE INFORMATION, BASIC, RELEASE 4.0.0

TEXAS INSTRUMENTS
INCORPORATED
DATA SYSTEMS GROUP

drawing number
2308940-9901

REV. **

SECTION 1

CONVERSION

In order to convert a BASIC program from DX BASIC V3.2 or earlier, SBC BASIC or TX5 BASIC to BASIC version 4.0 it is necessary to SAVE the program using the LIST option under the original version of BASIC. The program can then be accessed normally under 4.0 BASIC.

SECTION 2

NEW FEATURES

The features new to DX BASIC include improved performance over previous versions for most large programs, better program security, more user space, and support of external BASIC subprograms.

2.1 Improved Performance and Security

BASIC 4.0 includes a faster execution mode. In order to execute a program in fast mode you should SAVE the program with the LOCK option specified, then RUN the program from the file specified in the SAVE statement. In most large programs a significant improvement in speed of execution results.

The original, human readable format program cannot be derived from a program saved with LOCK. Among other things, the names of variables, arrays, and functions have been deleted from the file altogether.

When a program is saved with LOCK specified, all line number references (such as in a GOTO statement) and all variables must be valid. You may encounter the following errors when trying to perform a SAVE/LOCK on a program which executes without error:

* Error 28: "Multiply Declared or Used Variable." This can occur if an array is used in a line which precedes the line in which it is defined (with a DIM, INTEGER, REAL, or DECIMAL statement). In some cases this is permitted in development mode (and in earlier versions of BASIC). For a program to be saved in LOCK format, you should move the declaration to a lower line number.

* Error 60: "Line Not Found." This may happen when there is, in fact, a reference to a line number which does not exist. In normal development mode, no error is issued if the statement with the error is not executed. This

error may also happen if one program RUNs another program with a destination line number. If the destination line number was a comment, then that line is deleted from the fast format program by the SAVE LOCK processor. The programmer should, therefore, avoid using a RUN to a line number that is a comment.

* Error 40: "Stack Overflow." For certain operations, a program saved with LOCK specified will consume more of the stack than the same program not saved with LOCK. See the manual description of error 40 for causes of a stack overflow. A program which executes without a stack overflow on previous versions of BASIC should execute correctly in lock format.

* Error 19: Illegal token for lock format program. The following statements are not supported in a program saved with LOCK:

  MERGE
  BRKPNT
  UNBRKPNT
  TRACE
  UNTRACE

  In order to SAVE a program containing a MERGE statement in lock format. The programmer may commence execution of the program to cause the merge to happen (which results in the elimination of the MERGE statement). The programmer may then halt the program and then SAVE it in LOCK format.

  Any debug statements should be removed from the program before it is saved with LOCK specified.

* Error in SAVE LOCK. If an error occurs during a SAVE with LOCK specified, the program may be lost from memory. You should, therefore, save the program in non-lock format before saving it with LOCK. Of course, the non-locked program should be kept so that the program can be modified in the future.

Note that, unlike previous versions of BASIC, you cannot do an OLD on a locked program. In order to execute the program you must issue a RUN command which includes the pathname of the locked program.

2.2 Increased User Space

The amount of user space has increased over previous releases of DX BASIC. This has been accomplished by taking parts of the BASIC interpreter and putting them in a file to be loaded when they are needed. These parts of the interpreter are called overlays. Since not all of the interpreter resides in memory at once, more memory is available to the user.

When an overlay is brought into memory, it is loaded into an "overlay buffer". If all overlay buffers are occupied and a new overlay is needed from the disk, then the least recently accessed overlay in memory will be replaced. The user (programmer) has responsibility for defining the number of overlay buffers. If too few buffers are allocated, an excessive amount of disk reads will occur and BASIC will perform slowly. If too many buffers are allocated, then there may be insufficient user memory (freespace) to execute the program. Different programs require different numbers of overlay buffers to perform optimally. The programmer can determine the optimal number of overlay buffers for a program by experimentation. The number of buffers should be increased as long as this improves the performance (speed) of the program, providing there is sufficient user space. Each overlay buffer consumes about one kilobyte of memory.

The number of overlay buffers and the maximum amount of freespace can be specified in the following ways:

* When BASIC is bid from SCI, the user is prompted for these values.

* A SIZE clause can be added to a RUN, OLD, or NEW statement using one of the following formats:

```
RUN <pathname>           SIZE <freespace> , <buffers>
RUN <pathname> <line#>   SIZE <freespace> , <buffers>
OLD <pathname>           SIZE <freespace> , <buffers>
NEW                      SIZE <freespace> , <buffers>
```

In all cases, the overlay buffers will be allocated before freespace. The amount of memory requested by <freespace> will be acquired if that space is available from the operating system.

2.3 External BASIC Subprograms

BASIC 4.0 supports external (disk resident) BASIC language subprograms. When an external subprogram (or esub) is invoked, it is loaded from disk into memory and executed. Part of the main program is removed to allow more room for the esub. When execution is complete, the part of the main program which was overwritten is re-loaded and execution resumes just after the call.

Syntactically, esubs are just like (internal) BASIC subprograms except that the keyword SUB is replaced by the keyword ESUB. Invocation of both internal and external subprograms is done with a CALL statement followed by a parameter list, and in both cases the subprogram is terminated with a SUBEND or SUBEXIT statement.

The main program and esubs may have internal subprograms. The internal subprograms are local to the main program or the esub containing them. All esubs must follow the main part of the program. Comment lines are not permitted between esubs (though they are permitted between internal subprograms).

The addition of esub support complicates editing programs somewhat. In previous versions of BASIC, the whole program being edited resides in memory. Now, there is the possibility that the program being edited (the main plus the esubs) is larger than memory itself. We have added a work file to contain the current program and any esubs and we have added several commands to allow proper access to the main program or esub. These commands are as follows:

* EDIT MAIN - This causes the main program in the work file to be loaded into memory for editing.

* EDIT ESUB <esub name> - This causes the esub to be loaded into memory for editing.

* UPDATE - This causes the main program or esub in memory to be copied back to the work file. If the first word in the first line is not ESUB, then the program is considered to be the main program. If the keyword ESUB is encountered at the beginning of a line, then the UPDATE processor assumes that it is at the end of the current main program or esub and begins a new esub.

* DELETE ESUB <esub name> - This causes the specified esub to be deleted from the work file.

The following existing commands have their definitions extended as follows:

* SAVE - This causes an UPDATE to be performed. The work file is then copied to the destination file in the format specified (i.e. LIST, LOCK, or no specification).

* OLD - The file is copied into the work file and the main program is brought into memory for editing.

* RUN - Like OLD, except that execution commences on the main program.

* LIST - The LIST command will display only the part of the program that is in memory. In order to view the main program and all of its esubs, the following command can be entered:

  SAVE "ME" LIST

If an error occurs during the execution of an esub, the operator will have access to all variables in the esub, but to none of the variables in the main program. In order to change part of the main program or esub, an EDIT command must be entered.

Esubs have the following performance (speed) advantages over chaining (i.e. having one program transfer control to another program via a RUN statement):

* When you do a RUN, the file containing the destination program will have to be opened. This involves multiple disk accesses to locate the file. An esub resides in the work file which is always open.

* When you do a RUN, you must use files to pass information to the destination program. This file must be opened by the calling program and by the destination program, which again involves multiple disk accesses to locate the file. The information must be written and read, which again involves disk access. With esubs, memory resident data can be passed via parameters. This involves no disk access.

2.4 Sort and KFP Subprograms

Assembly language subprograms for sorting a relative record file and for providing a key index capability to relative record files are provided. These subprograms are essentially the same as subprograms released under SBC BASIC, and they are provided to help SBC users migrate to the new BASIC. Please read the manual for a description of these routines.

SBC users should note that the KFP buffer is not provided by the operating system. It is necessary to build a dumy subprogram which will act as the buffer and to include it in a LIBRARY statement in all programs using these subprograms. If chaining from one program to another, it will be necessary to re-initialize the buffer. Please read the manual for more information.

SECTION 3

O. S. SPECIFIC INFORMATION

BASIC 4.0 is released under the following three Operating Systems:

* DX10 Micro (or DXM)
* DX10
* DNOS

These forms of the BASIC interpreter are almost identical--the only differences are in some aspects of the operating system interface. Because of this similarity, migration of programs from one operating system to another will rarely require any programmer intervention.

BASIC for DXM should be used only on DXM O.S. Version 1.1 or later.

DX BASIC used to require that assembly language subprograms be linked using the TXXLE linker. The standard XLE linker may now be used.

Following are some differences in operating system capability which may effect program migration:

* Background mode. DXM does not have a background mode.
* Temporary files. DXM does not support temporary files. An attempt to create a file as temporary will result in an error under DXM.
* Pathnames. DXM supports only one level of directory between the volume and the file.
* Available space. There is significantly less available space under DXM than there is under DX10 and DNOS.
* DXM Key Indexed File Performance. In order to service KIF I/O calls, DXM needs to load system overlays from disk. If less than maximum total memory is consumed by

the BASIC program (i.e. if freespace is less than it might be) then DXM can keep more than one system overlay in memory at once and significantly improve performance. Thus when tuning the performance of a DXM program using KIF, the programmer should vary the total requested freespace as well as the number of BASIC overlay buffers.

* DXM SCI interface subroutines. Under DXM, the SCI interface subroutines (or S$ routines) may not be used in BASIC assembly language subroutines.

* DNOS Spooler Devices. Under DNOS you cannot open a printer which is defined to be a Spooler Device. An attempt to do so will result in OS Error >9D.

The following involves some advanced topics in operating system differences intended for programmers versed in the operating system organization, file structure, and generation (SYSGEN).

* DXM Unblocked Files. An unblocked file is a file whose physical record size is less than twice the (logical) record size. For a file created under BASIC, the physical record size will be the volume's Default Physical Record Size established when the volume is initialized (or the logical record size, if it is larger). Under DXM, an unblocked file requires less memory (in other words, allows the system to be generated with fewer blocking buffers) than a blocked file, but a blocked file usually requires less disk space than an unblocked file. In general, files with large records should be unblocked and files with small records should be blocked.

SECTION 4

KNOWN BUGS

Following are the known bugs:

* If a PRINT to a Key Indexed File may not function correctly if it follows an INPUT to that file which ends in a comma. The PRINT processor may "think" that it has reached the end of record before it actually has. To avoid this problem, the programmer should assure that the last INPUT statement accessing a KIF record does not end in a comma.

* An attempt to perform a RUN with no pathname from a LOCK format program will cause the program to terminate.

* An attempt to INPUT a string value from an empty relative record file may produce no error (as it would in previous versions of BASIC). The value of the string will be empty.

* It is possible for a parameter in a SUB statement to have the same name as the subprogram itself in normal development mode. An attempt to SAVE with LOCK a program containing such a SUB statement will produce an error. You should not, therefore, use the name of a subprogram as one of its parameters.

* It is possible for a variable in a subprogram to have the same name as the subprogram itself. Saving such a program in lock format will produce bad object and will result in execution errors. You should not use the name of a subprogram as the name of one of its variables.

* If you stop a locked program with the blank orange key followed by Control-X, an invalid line number may be displayed.

* If a locked program encounters an error in accessing to pre-load an assembly language subprogram (i.e. one whose pathname in the LIBRARY statement is preceded by an asterisk), an invalid line number will be displayed.

Following are known errors in the manual:

* The first example on page 5-13 of the manual has two mistakes. First, the virtual array ENUM should be specified as REAL rather than INTEGER since the numbers given are too large to be contained in an integer. Second, no hyphens should appear in the numeric output.

* The second example on page 6-4 is incorrect. A record length should be supplied in the "FIXED" clause of the OPEN statement, or the "OUTPUT" clause should be removed so that a create operation is not implied. Either of the following is correct:

```
OPEN #1:"DS01.MYFILE",RELATIVE,INTERNAL,FIXED 80,OUTPUT

OPEN #1:"DS01.MYFILE",RELATIVE,INTERNAL,FIXED
```

SECTION 5

MISCELLANEOUS NOTES

Following are some miscellaneous notes on this product:

* Rounding of a value exactly equal to one half is performed slightly differently when assigning a value to a variable of type INTEGER and when assigning to a variable of type DECIMAL. For INTEGER, x.5 will always round to the next higher value. For DECIMAL(0), x.5 will round to a higher value for positive numbers and to a lower number for negative numbers.

* In several places the manual states that if a value being written is too large to fit in a record of a Relative Record file, it will be truncated. In fact, the value is continued in the next record. INPUT will also continue to the next record to get the remainder of the value.

* The size and precision of real numbers may not be clear from some parts of the manual. Thirteen or fourteen significant digits are maintained and exponents can be from -127 to +127. Real numbers will be displayed in 12 characters unless more are provided by a Formatted Output statement ("PRINT .. USING .." or "DISPLAY .. USING .."). Only two digits are provided for displaying the exponent. A three digit exponent will be displayed as asterisks unless a sufficiently large exponent field is provided by a Formatted Output statement.

* If a program is halted by a breakpoint, by keying in the break sequence, or by an error, the variables will retain their values, unit numbers will continue to be associated with their files or devices, and the Operating System LUNOs will remain assigned. An attempt to delete a file with a LUNO assigned to it will result in an Operating System error. Performing the NEW command or performing an edit function will clear the variables and release the LUNOs, allowing the files to be deleted.

* An array can be created by using it without specifically declaring it only within a program. An attempt to auto-create an array from top level (outside a program) will,

therefore, result in error 22 (Symbol Not Found or Creatable).

* A breakpoint set at a line containing a SUB statement will not halt execution.

* The default record length used when creating a sequential file is 256 rather than 80 as stated in the manual.

* The following keywords will be truncated to three characters when put into a program:

```
BRKPNT
UNBRKPNT
TRACE
UNTRACE
UPDATE
MERGE
```

* The description for Error 50 should state that this can occur if there is a FNEND without a corresponding DEF.

* The Keyed File Package example in the manual will not work under DXM unless the number of overlay buffers is 3.

* The manual does not clearly state how to unlock a record of a Relative Record file that has been locked by an INPUT statement. The record may be unlocked by performing a PRINT to that record or by performing an UNLOCK statement as shown below:

```
UNLOCK #1, REC 5
```

* As stated in paragraph 5.7 of the manual, virtual arrays are relative record files. A file used as a virtual array must, however, be created by the ASSIGN statement and should be accessed only as a virtual array.