REALITY

DATA/BASIC PROGAMMER'S REFERENCE MANUAL

PRELIMINARY

TABLE OF CONTENTS

1 INTRODUCTION

DATA/BASIC programs pass through two phases: translation and execution. Translation is handled by the compiler, the assembler, and the loader. Execution is handled by the run-time interpreter.

Each of these major elements of the DATA/BASIC System is described in the following pages.

3 DATA/BASIC TCL Commands

3.1 META Command

META is the TCL-II verb issued to compile the DATA/BASIC grammar. This verb creates a new file item containing the intermediate object code for the grammar compiled.

The verb format is:

```
META file-name item-id item-id item-id
```

The file-name indicates the file containing each of the three items listed, typically BASIC-COMPILER. The first item-id indicates the grammar to be compiled, typically BASICBNF. The second item-id indicates the intermediate object code of the meta compiler , typically METACODE. The third item-id indicates the item where the intermediate object code generated is to be stored, typically BASICCODE.

The verb definition is:

```
1. P
2. 2
3. 88
4.
5. N
```

3.2 COMOPT Command

COMOPT is the TCL-II verb issued to assemble the intermediate object code for the DATA/BASIC compiler and load it into frames 250-262. The verb format is:

```
COMOPT file-name item-id
```

The file-name and item-id specify the intermediate object code of the DATA/BASIC compiler.

The verb definition is:

```
1. P
2. 2
3. 89
4.
5.
```

3.3 BASIC Command

BASIC is the TCL-II verb issued to compile a DATA/BASIC program. This verb creates a new file item containing the item object code version of the compiled program. The new file item-id is created by appending a dollar sign ($) to the beginning of the item-id of the source program. If the M option (see below) is used, a map of the descriptor table is generated for use by the BASIC DEBUGGER. The item-id for this item is created by appending an asterisk (*) to the beginning of the item-id of the source program. The verb format is:

BASIC file-name item-list (options)

The item-list consists of one or more item-ids separated by one or more blanks, or an asterisk specifying all items in the file.

Valid options are:

A List the intermediate object code. Requires the file BASIC-ASSY containing the opcode mnemonics.

B Allocates variables in the same manner as 2.4 DATA/BASIC for the purpose of 2.4 program compatibility when chaining.

E Suppresses end-of-line (EOL) opcodes from the item object code, in order to reduce the size of the debugged cataloged program.

F Print a full map showing compiler generated variables and labels. (Works only in conjunction with the M option.)

L List DATA/BASIC program.

M Print a variable map showing each variable's offset in the descriptor table, each label's offset in the run-time code, and a statement map showing the range of statements in each frame.

N Eliminates wait at the end of each page listed on the terminal.

P Print DATA/BASIC program listing and/or compiler error messages on the line printer.

X Generate cross index of variables and labels in file CSYM.

The verb definition is:

1. P
2. 2
3. BE
4.
5. UPZ

3.4 RUN Command

RUN is the TCL-II verb issued to execute a compiled DATA/BASIC program. This verb appends a dollar sign ($) to the beginning of the item-id specified to locate the item object code version of the program. This item object code program is loaded and executed.

The verb format is:

RUN file-name item-id (options)

The file-name and item-id specify the compiled DATA/BASIC program to be executed.

Valid options are:

D Retrieve the symbol table and enter the DATA/BASIC Debugger.

I Inhibit initialization of Common Descriptor Table and Free Storage area for CHAINing.

S Suppress warning error messages.

P Output to the line printer.

G Generates listing of 12 garbage collection counters of byte and buffer utilization during program execution as follows:

1. Total number of bytes used
2. Total number of bytes abandoned
3. Number of 50 byte buffers used
4. Number of 50 byte buffers abandoned
5. Number of 50 byte buffers reused
6. Number of 150 byte buffers used
7. Number of 150 byte buffers abandoned
8. Number of 150 byte buffers reused
9. Number of 250 byte buffers used*
10. Number of 250 byte buffers abandoned*
11. Number of 250 byte buffers reused*

12.	Number of times space reallocation (or garbage collection) took place
*	buffers larger than 250 bytes are included in counter 9, 10, and 11

The verb definition is:

1. P
2. 2
3. B4
4.
5. F

3.5 CATALOG Command

CATALOG is a TCL-II verb issued to load a compiled DATA/BASIC program into the catalog area. This verb appends a dollar sign ($) to the beginning of the item-id specified to locate the item object code version of the program. This item object code is then loaded into the OS work space. Once the size of the run-time object code is known, it is copied to a contiguous block of frames taken from the overflow space. A pointer-item is created in the POINTER-FILE with the item-id:

Acct-name*C*program-item-id

The pointer-item looks like:

1. C
2. n
3. m
4.
5. Time date

where 'n' is the frame number in which the cataloged run-time object code begins and 'm' is the number of contiguous frames used. The time and date record when the program was catalogued. The following message is displayed:

[241] 'item-id' CATALOGED; n FRAMES USED.

A verb is created in the user's M/DICT with the same name as the program. This verb looks like:

1. P
2. 10B4
3.
4.
5. Acct-name

The CATALOG verb format is:

CATALOG file-name item-id

The verb definition is:

1. PX
2. 2
3. B4
4.
5. F

3.6 PRINT-CATALOG

PRINT-CATALOG is a TCL-I verb which prints compilation data of a cataloged BASIC program.

The form of the verb is as follows:

PRINT-CATALOG <program item-id> [<account name>]

where account name refers to the account under which the BASIC program was cataloged. The account name is optional and if not specified PRINT-CATALOG searches the Pointer-file for the item-id under the current user's account.

PRINT-CATALOG will print the file and item names of the cataloged BASIC program, the date and time of compilation, as well as the precision used by the program. (Refer to the DATA/BASIC Reference Manual in regards to the PRECISION declaration.)

The verb format is:

1. PC
2. 60C0

3.7 PRINT-HEADER

PRINT-HEADER is a TCL-II verb which prints the file and item names, the date and time of compilation, as well as the precision used in the program. (Refer to the DATA/BASIC reference Manual in regards to the PRECISION declaration.)

The form of the verb is as follows:

PRINT-HEADER <file-name> <item-id>

A "$" is appended to the specified item-id in order to locate the object code of the program.

The verb format is:

1. PC
2. 2
3. 40C0
4.
5. F

3.8 BVERIFY

BVERIFY is a TCL-I verb which will verify a cataloged DATA/BASIC program aginst the item-object version on file. This verb is simular to the MVERIFY verb in that it will print the location, expected byte, and found byte of the first mismatch, and unless the A-option is specified, will thereafter only count the mismatches to the end of object code.

The form of the verb is as follows:

BVERIFY <program id> [<account name>]

where account name refers to the account under which the BASIC program was cataloged, and is optional. The current user is assumed.

BVERIFY will obtain the source file and item-id from the cataloged program header, and will attempt to verify against the item-object version on that file.

Valid options are:

A Print all mismatches found. This verb DOES NOT PAGE, and will scroll up the page if there are a large number of mismatches.

P Output to the system printer.

The verb format is:

1. PC
2. 30AA

4 TRANSLATION

The DATA/BASIC COMPILER translates DATA/BASIC source code to zero address (STACK) interpretive object code.

The two processors that accomplish this are called the COMPILER, which only does string manipulation, and the ASSEMBLER, which translates variable names and statement numbers to addresses.

The BASIC verb invokes the COMPILER and then the ASSEMBLER. Since, binary numbers cannot be stored in an item, the ASSEMBLER must convert binary addresses to ASCII hexadecimal addresses before storing them in an item. These addresses are then converted back to binary at run-time by the LOADER.

4.1 COMPILER

The Compiler is composed of 2 parts:

1. Run-time object code compiler in frames 250-262
2. Compiler run-time interpreter (Modes COM0 and COM1)

The DATA/BASI Compiler run-time package interprets the object code in frames 250-262 and translates the DATA/BASIC source code contained in an item to produce intermediate object code in the OS buffer.

4.1.1 The Compiler Run-Time Object Code

The followng TCL commands are used to translate the DATA/BASIC grammar into run-time object code stored in frames 250-262.

```
META BASIC-COMPILER BASICBNF METACODE BASICCODE

COMOPT BASIC-COMPILER BASICCODE
```

4.1.2 Intermediate Object Code

Intermediate object code consists of 1-byte opcodes, optionally followed by an ASCII variable name or number, and always terminated by an AM.

For example:

The intermediate object code generated for the program:

```
A=8+5
END
```

would be:

```
X'20' B       LOAD    B
X'05' 5       LOADN   5
X'A9'         ADD
X'B0' A       STORE   A
X'01'         EOL
X'01'         EOL
X'C0'         STOP
```

An opcode of X'FC' (SVM) precedes a label (statement number). For example: The intermediate object code generated for the program:

```
10 GOTO 10
   END
```

would be

```
X'FC' 10      010
X'06' 10      BRANCH  10
X'01'         EOL
X'01'         EOL
X'C0'         STOP
```

4.2 ASSEMBLER

The assembler (Modes COM2, COM3, COM4, COM5, COM6, COM7, and COM8) converts the intermediate object code in the OS buffer to Item Object Code and stores it in an item with the name 'S'<source-name>, where source-name is the name of the item that contains the source code.

4.2.1 Pass 1 (Modes COM3 and COM4)

Pass 1 builds a Symbol Table in the HS buffer of all the variables and labels in the DATA/BASIC program.

Each entry in the symbol table contains a variable length Symbol Name terminated by a system delimiter which indicates its type, followed by 9-bytes of binary data. The symbol table is terminated by a SM.

Data types, along with their associated system delimiters, are given below:

Simple variable:

AM
1 L if Local symbol, C if Common symbol
2-3 -byte offset it into the descriptor table
4-9 Unused

Dimensioned variable:

VM
1 L if Local symbol, C if Common symbol
2-3 2-byte offset into the descriptor table
4-5 Number of rows
6-7 Number of columns
8-9 Unused

Label:

SVM
1 L
2-3 2-byte offset into the Run-Time code.
4-9 Unused

Equated Label:

SB
1 Equate Type Code
2-9 Different uses depending upon the type code.

Code:
X'53' 2-7 SR to numeric literal
X'54' 2-7 SR to string literal
8-9 Length of string
X'55' 2-7 SR to numeric value
X'56' 2-7 SR pointing to equated symbol name
X'57' 2-7 SR pointing to equated symbol name

4.2.2 Pass 2 (Mode COM5)

Pass 2 converts the intermediate object code in the OS buffer to item object code in the IS buffer, and then calls UPDITM to store the object code in a file item.

In the process of conversion, Pass 2:

1. Replaces each label and variable reference with an ASCII hexadecimal address.

2. Replaces each array reference with an ASCII hexadecimal address, the number of rows in the array, and the number of columns in the array.

4.3 LOADER

The loader (located in BRP15) converts item object code (object code stored in an item) to run-time object code (object code that can be interpreted by the DATA/BASIC run-time interpreter). The loader is invoked by the RUN and CATALOG verbs. When the loader is called by the RUN verb, the converted code is loaded into the OS work space. When the loader is called by the CATALOG verb, the converted code is loaded into the OS work space and then copied into a contiguous block of overflow space.

4.3.1 Item Object Code

DATA/BASIC Item Object Code consists of an Object Code Header and 1-byte opcodes, optionally followed by one of the five operand types discussed below. The format of the Object Code Header is:

<file name>VM<item name>VM<date>VM<time>AM

Following the AM is the local descriptor table size (2-bytes), the common descriptor table size (2-byte), and the precision (1-byte), all binary values.

The valid operand types are as follows:

1. A 4-byte hexadecimal ASCII offset into the Descriptor Table (byte displacement relative to DESCBEG).

2. A 4-byte hexadecimal ASCII offset into the run-time Object Code (byte displacement relative to IR).

3. A 12 byte hexadecimal ASCII Array Dope Vector (3 4-byte numbers).

4. A decimal ASCII number terminated by an AM.

5. A literal ASCII string terminated by an AM.

The loader converts the 5 types of operands specified above to the corresponding 5 types of run-time object code operands specified below.

4.3.2 Run-time Object Code

DATA/BASIC Run-time Object Code consists of one byte opcodes, optionally followed by one of the following types of operands:

1. A 2-byte binary offset into the Descriptor Table (byte displacement relative to DESCBEG)
2. A 2-byte binary offset into the Run-time Object Code (byte displacement relative to IR).
3. A 6-byte binary Array Dope Vector (3 2-byte binary numbers).
4. A 6-byte binary number
5. A literal ASCII string terminated by a SM.

The Run-time Object Code is stored either in the OS work space or the Catalog area. OSBEG points at the AM following the object code header, starting with the 7th frame of OS, or the frame indicated by the second attribute of the cataloged program's POINTER-FILE item.

4.3.3 Storage Table

The first 5 bytes after the Object Code Header contain the following three values:

1-2	Descriptor Table Size
3-4	Common Descriptor Table Size
5	Precision

The first opcode starts with the 6th byte, after the Object Code Header, of the Run-time Object Code.

4.3.4 No-ops

The Run-time Object Code contains no-ops which do the following:

1. Opcode X'00' (PAD) which pads the last few bytes of some frames, so that operands never cross a frame boundary.
2. Opcode X'01' (EOL) marks the end of each line of source code.

5 EXECUTION

5.1 RUN-TIME Data Structures

5.1.1 Work Space Assignment

The IS, OS, and HS work spaces are used to store the Data/basic program and its data. These work spaces are allocated as shown below:

Work-space	Un-cataloged	Cataloged
IS	Descriptor Table Temporary Space	Descriptor Tables
OS	Run-time Code	Temporary Space
HS	Stack Page Heading Free Space	Stack Page Heading Free Space

5.1.2 Descriptor Table

ISBEG points to the first byte of the Descriptor Table, starting with the 1st byte of the 7th frame of IS. The Descriptor Table contains a 10 byte data descriptor for each variable (including array elements) in the DATA/BASIC program.

5.1.2.1 Data Descriptor Formats

Type	Bytes	Contents
Unassigned	1-10	Zeroes
Direct Number	1	Type code = X'01'
	2	Unused
	3-8	Binary number
	9-10	Unused
Direct String	1	Type code = X'02'
	2-10	Short string terminated by a SM
Indirect String	1	Type code = X'82'
	2	Unused
	3-8	SR pointing 1 byte before string in Free Space terminated by SM

	9-10	Free Space buffer size
File BMS	1	Type code = X'04'
	2	Unused
	3-6	Base
	7-8	Modulo
	9-10	Separation
Ext. Sub Pointer	1	Type code = X'40'
	2	Unused
	3-8	Pointer to cataloged subroutine
	9-10	Unused

5.1.3 Temporary Space

ISEND points to the 1st byte of Temporary Space. Temporary Space follows the Descriptor Table when the program is not cataloged. Temporary Space starts with the 1st byte of the 7th frame of OS if the program is cataloged.

5.1.4 Run-time Code

OSBEG points to the 1st byte of the Run-time Code. The Run-time Code starts with the 1st byte of the 7th frame of OS when the program is not cataloged. OSBEG points to the 1st byte of the contiguous block of frames when the program is cataloged.

5.1.5 Stack

HSBEG points to the 1st byte of the Stack. The Stack starts with the 1st byte of the 7th frame of HS and extends 4 frames. The Stack may contain up to 200 10-byte stack descriptors.

5.1.5.1 Stack Descriptor Formats

If Bit 5 of the Type Code (X'04') is set, then any offset refers to the Common Descriptor Table, unless otherwise specified by context in which used.

Type	Bytes	Contents
Dope Vector	1	Type code = X'01'
	2	Unused
	3-4	2-byte offset of 1st array

		descriptor in Descriptor Table
	5-6	Number of rows
	7-8	Number of columns
	9-10	Unused
Abs. Dope Vector	1	Type code = X'20'
	2	Unused
	3-8	SR pointing to 1st array descriptor in Descriptor Table
	9-10	Cell count (i.e. the number of descriptors in the array)
Address	1	Type code = X'00'
	2	Unused
	3-4	2-byte offset of variable's descriptor in Descriptor Table
	5-10	Unused
Abs. Address	1	Type code = X'10'
	2	Unused
	3-8	SR pointing to variable's descriptor in Descriptor Table
	9-10	Unused
Return Address	1	Type codes: Local return = X'08' External return = X'18'
	2	Unused
	3-8	SR pointing into run-time object code
	9-10	Unused
Temporary String	1	Type code = X'C2'
	2	Unused
	3-8	SR pointing 1 byte before string in Temporary Space terminated by a SM
	9-10	Unused
End of Parmlist	1	Type code = X'7f'
	2-10	Unused

5.1.6 Page Heading

PAGHEAD points to one byte before the Page Heading. The Page Heading starts with the 100th byte of the 11th frame of HS and extends 1400 bytes. The 1st 100-bytes of the 11th frame are left unused in case of stack overflow.

5.1.7 Free Space

UPDBEG points to the 1st byte of Free Space. Free Space starts with the 1st byte of the 12th frame of HS and extends to the end of the HS work space. Free Space contains two entries, the Abandoned Buffer Table and the Free Storage Buffer Area.

Two counters are kept of Free Space used and abandoned. D4 is the Abandoned Space count, and D5 is the Free Space Used count. Garbage collection takes place when the space abandoned exceeds the space used.

5.1.7.1 Abandoned Buffer Pool

A pool containing not more than ten of the largest abandoned buffers is maintained by the system. Each entry in the pool contains: 1) the size of the abandoned buffer, and 2) a SR pointing to that buffer. DATA/BASIC will search the pool for a buffer of the appropriate size before going to the end of the used Free Space. This has the effect of preventing excessive 'CHECKER BOARDING' of Free Space.

5.1.7.2 Free Space

UPDBEG+80 points to the first byte of Free Space. The Abandoned Buffer Table and Free Space start with the 1st byte of the HS work space. UPDEND points to the 1st byte of available Free Space. Initially, UPDEND = UPDBEG+80

D4 is the Abandoned Buffer Byte Count, and D5 is the Free Space Byte Count. When D4 becomes greater then D5, (that is, when the amount of Abandoned space becomes greater then the amount of Free Space used) the system will garbage-collect.

5.1.8 Buffer Utilization Counters

The RUN or cataloged BASIC verbs, used with the G-option, keep a table of 12 counters on the utilization of free space and there associated buffers. They are:

1) Total number of buffer bytes allocated (SCB element D6).

2) Total number of buffer bytes abandoned (SCB

element D8).

3) Number of time garbage collection took palce (SCB element C5).

4) 3 Classed of 3 counters, located at R0+22 (ISBEG+6).

The 3 classes are:

a) 50 byte buffers

b) 150 byte buffers

c) 250 byte buffers, and multiples thereof.

The 3 counter groups are:

a) Buffers abandoned.

b) Buffers reused from the abandond buffer table.

c) Buffers allocated from available free space

SET-TBL (7,BRP17) sets R15 to the first element of this 9 element table, each element being a tally.