Camel99 Forth

for the TI-99 Home Computer

Library File Reference

Revision 0.4 June 2021

For Camel99 Forth V2.67

by Brian Fox

Table of Contents

Introduction

This document contains a list of the library files released with Camel99 Forth for the TI-99 Home Computer with the kind of word it is, a stack diagram and short description for the definitions in each file. The glossary is organized by file name and the Forth words in each file are listed in the order they are compiled into Forth.
Much of the text in this document was machine generated by the DOCGEN program. Due to that fact, there is more information embedded in the source code files than was captured by DOCGEN. This document is a programmer's reference. Since the library files are source code, the best way to understand them fully is to study the source code in the files.

Using the Library Files

The Camel99 Forth system is a micro-kernel, 8 Kbytes in size that gives you only the Standard Forth "Core" word set and a few extra words from the Core Extension word set. You can write useful programs with these words but it is better to have code that is already working that you can use to get your project going faster.

Everything beyond the CORE words in the micro-kernel must be brought into the Forth system from the library files. Since Forth is extendable it is easy to add features to the language. The library files give you working code for the features of the TI-99. For example you might want to use HCHAR and VCHAR that you have come to expect from TI BASIC. These words and other for character definition, character placement and sprites are in the GRAFIX library file.
There is also ANS/ISO Forth file management words for programs that need file system support, multi-tasking, data structure creation and even named colours if want them. There is probably a library file for most of what you need in the beginning but you are also free to create your own library files as well. In fact most Forth programmers have their own special set of files that they use and refine over time.

How to INCLUDE

To use a library file use the word INCLUDE and the file name at the top of your program:

```
INCLUDE DSK1.GRAFIX
INCLUDE DSK1.AUTOMOTION
```

These commands will cause the all the graphics words and sprite words to compile into Forth.

Conditional Compilation: NEEDS/FROM

INCLUDE will just compile the file you specify without checking if the file is already in the system. This would waste a lot of memory so Camel99 has two magic commands that are used to prevent double loading a library file. This is the preferred way to do it and if you examine the library files you will see they use the words NEEDS and FROM.
NEEDS looks in the dictionary for a word of your choosing. It must be a word that is in the library file you are interested in.
FROM includes a file bit only if a "flag" on the data stack is FALSE. (0)

Example:

```
NEEDS  HCHAR  FROM DSK1.GRAFIX
```

This is understood to mean check to see if HCHAR is in the dictionary. If it is NOT in the dictionary INCLUDE DSK1.GRAFIX. Using NEEDS/FROM can prevent double loading library files and saving that memory for your programs.

Contact Info:

Brian.fox@brianfox.ca
theBF at: http://atariage.com/forums/forum/119-ti-994a-development/

DSK*.+CONSTANT

```
Creates an incrementing or decrementing constant

+CONSTANT   word       CREATE/DOES>

Lines: 11
Bytes: 324
--- End of file ---
```

DSK*.2ROT

```
2ROT        word       ( d d2 d3 -- d2 d3 d) ROT for double precision number.

Lines: 3
Bytes: 98
--- End of file ---
```

DSK*.3RD4TH

```
Fast access to items deeper in the stack. Works like OVER. 50% faster than PICK.

3RD         code word  ( a b c d --  a b c d b)
4TH         code word  ( a b c d e--  a b c d e a)

Lines: 26
Bytes: 776
--- End of file ---
```

DSK*.80COL

```
( default colors BLACK on CYAN )
40COL       create       40 col register data
80COL       create       80 col register data
VREGS       word       ( addr n -- )
40COLS      word       ( -- )
80COLS      word       ( -- )

Lines: 26
Bytes: 829
--- End of file ---
```

DSK*.ANSFILES

```
V2.24 removed file length and FAM=0 error test from OPEN-FILE
V2.25  used new FOPEN to simplify OPEN-FILE, FREAD for READ-FILE

#FILES      constant    ( -- n) Locked at 3 files
LASTH       variable    ( -- n) Last file handle used
FIDS        create     ( -- addr) Array of File identifiers. (Handles)
```

```
FATAL       word        ( -- true) Clear all handles return true.
?HNDL       word        ( n -- )   Is n a valid handle
]FID        word        ( hndl -- PAB_addr ) FIDS exposed as an array
NEWHNDL     word        ( -- hndl) Handle server. Tries to find unused handle to use
RELEASE     word        ( hndl -- ) Reset the hndl ]FID to zero
SELECT      word        ( hndl -- ) Make the hndl active.
VCOUNT      word        ( vdp$adr -- vdpadr len ) counted Vstring to stack Vstring.
.FNAME      word        ( padaddr -- ) print a file name
DUMP]       word        ( vaddr -- ) Dump a PAB usage: [PAB DUMP]
?FILERR     word        ( ior -- ) Default file error handler
FAM         variable    ( -- addr) Hold the active file access mode (FAM)
AND!        code word   ( mask addr -- ) Logical AND a variable with mask
OR!         code word   ( mask addr -- ) Logical OR a variable with mask
```

TI-99 file access mode modifiers

```
DISPLAY     word        ( -- )
SEQUENTIAL  word        ( -- )
RELATIVE    word        ( -- )
UPDATE      word        ( -- )
INPUT       word        ( -- )
OUTPUT      word        ( -- )
APPEND      word        ( -- )

B/REC       variable    ( -- addr) Internal use by OPEN-FILE,CREATE-FILE
VARI        word        ( size -- fam) Set “variable” length file mode
FIXED       word        ( size -- fam) Set “fixed” length file mode

R/W         word        ( -- fam) TI-99 update mode
R/O         word        ( -- fam) TI-99 input mode
W/O         word        ( -- fam) TI-99 output mode

BIN         word        ( fam -- fam')  ANS Forth BIN replaces TI-99 "INTERNAL"
DV80        word        ( -- ) Set file mode to Display variable 80

*IOR Input/output response ie: the error message
*FID File identifier ie: a file handle

OPEN-FILE   word        ( $addr len fam -- fid ior) ANS open file, return handle,ior
CLOSE-FILE  word        ( fid -- ior) ANS close a file
EOF         word        ( fid -- c) TI-99 test
CREATE-FILE word        ( caddr len fam -- fid ior ) ANS Create and open file.
FILE-POSITION word      ( fid -- rec# ior) ANS return record number
REPOSITION-FILE word    ( rec# fid -- ior) ANS move to record rec#
DELETE-FILE word        ( caddr len fam -- ior) ANS delete a file
READ-LINE   word        ( addr u1 fid -- u2 flag ior ) ANS read a file record
WRITE-LINE  word        ( c-addr u fileid -- ior ) ANS write a file record.

Lines: 120
Bytes: 3627
--- End of file ---
```

DSK*.ARRAYS

```
These work as expected but were a litte slow due to DOES> overhead
This version replaced runtime Forth with machine code that is 3X faster.

CARRAY      word       ( n -- ) Creates a byte array of size n

ARRAY       word       ( n -- ) ARRAY creates an integer array of size n


Usage:
(square bracket is a reminder, this is an array. NOT SYNTAX)

   20 CARRAY ]Q     99 6 ]Q C!   6 ]Q C@ . ( 99)
   20 ARRAY ]T    1234 3 ]T !    3 ]T @ . ( 1234)

Lines: 30
Bytes: 1081
--- End of file ---
```

DSK*.ASM9900

ORIGINAL TI-FORTH ASSEMBLER modified by Mark Wills, TurboForth
Dec 23,2020 Huge simplification with ANS style branching & looping. Brian Fox

Notes:

Compare instruction has been changed to CMP,
Changed A, and S, to ADD, SUB,

```
/ASM        Marker    Removes Assembler from the system.

R0          constant
R1          constant
R2          constant
R3          constant
R4          constant
R5          constant
R6          constant
R7          constant
R8          constant
R9          constant
R10         constant
R11         constant
R12         constant
R13         constant
R14         constant
R15         constant
```

```
ADR?        word       ( n -- ? )  Test if n is address or register?

GOP'        word       CREATE/DOES> General instructions creator
GROP'       word       CREATE/DOES> General instructions creator
GGOP'       word       CREATE/DOES> General instructions creator
0OP’        word       CREATE/DOES> No argument instructions creator
ROP'        word       CREATE/DOES> STWP, STST, instruction creator
IOP’        word       CREATE/DOES> LWPI, LIMI, instruction creator
RIOP’       word       CREATE/DOES> Immediate instructions creator
RCOP’       word       CREATE/DOES> Shift instructions creator
DOP’        word       CREATE/DOES> Jump instructions creator
```

Jump tokens

```
GTE         constant    Greater than or equal jump token
HI          constant    High jump token
NE          constant    Not equal jump token
LO          constant    Low jump token
LTE         constant    Less than or equal jump token
EQ          constant    Equal jump token
OC          constant    Jump on Carry token
NC          constant    Jump no Carry token
OO          constant    Jump on overflow token
HE          constant    High or equal jump token
LE          constant    Low or equal jump token
NP          constant    No Parity jump token

GCOP'       word       CREATE/DOES> CRU instructions creator
```

Wycove Forth Addressing Mode Selectors

```
@@          word         symbolic addressing

**          word         indirect addressing

*+          word         indirect addressing, auto-increment

()          word         indexed addressing
```

Structured branching and looping

```
AJUMP,      word        ( token --) >1000+token makes a jump instruction
RESOLVE     word        ( 'jmp offset --) compile offset into 'jmp'
<BACK       word        ( addr addr' -- ) Compute jump backwards, compile value
```

Note:

These structured Assembler branches and jumps compile normal instructions into your code behind the scene. If you prefer seeing these instructions in your code use the ASMLABELS library and use the JEQ, JNE, etc. Style of coding.

```
IF,         word        ( addr token -- 'jmp') Performs a jump to ENDIF, or ELSE,
ENDIF,      word        ( 'jmp addr --) Destination for IF, to jump to
ELSE,       word        ( -- addr ) Alternative destination for IF,

BEGIN,      word        ( -- addr) Start a loop
WHILE,      word        ( token -- *while *begin)  loop while token is true
AGAIN,      word        ( *begin --) jump back to BEGIN,
UNTIL,      word        ( *begin token --) Jump to BEGIN, until token is true
REPEAT,     word        ( *while *begin -- ) Jump back to BEGIN, in a WHILE, loop

;CODE       word
```

CAMEL99 Forth named registers

The Assembler has extra register names for the registers used by the Forth virtual machine to specify the different addressing modes. For other registers use the Wycove addressing mode modifiers or create your own extra register names using ASM9900 source code as examples.

```
TOS         constant    TOS "Top of stack register" is R4 in Camel99 Forth
(TOS)       word        Specify Indexed addressing mode for TOS: 4 (TOS)
*TOS        word        Specify Indirect addressing mode
*TOS+       word        Specify indirect, auto-increment mode

SP          constant    Forth DATA stack pointer register
(SP)        word
*SP         word
*SP+        word

RP          constant    Forth return stack pointer register
(RP)        word
*RP         word
*RP+        word

W           constant    Forth "working" register
(W)         word
*W          word
*W+         word

IP          constant    Forth Interpreter pointer register
(IP)        word
*IP         word
*IP+        word
*R10        word
*R11        word
```

Pseudo instructions

```
RT,         word       ( -- ) Return from BL sub-routine
NOP,        word       ( -- ) No operation
NEXT,       word       ( -- ) Return to Forth inner interpreter
PUSH,       word       ( src -- ) Push src register onto DATA stack
POP,        word       ( dst -- ) Pop from DATA stack into dst register
RPUSH,      word       ( src -- ) Push src register onto RETURN stack
RPOP,       word       ( dst -- ) Pop from RETURN stack into dst register

Lines: 190
Bytes: 5016
--- End of file ---
```

DSK*.ASMLABELS

```
Provides numbered labels for ASM9900                Apr 3 2021 Fox
Original idea from DxForth. Complete rewrite uses a stack for forward refs.

See: Camel99 manual for details

Dependancy: DSK*.ASM9900

#FWD        constant
#LABELS     constant
FS0         create
FSP         create
FSDEPTH     word        ( -- n)
>FS         word        ( addr --)
FS>         word        ( -- addr)
LABELS      create
]LBL        word        ( n -- addr)
NEWLABELS   word        ( -- ) clear label array  reset fwd stack pointer to base
address

$:          word        ( n -- )  code label creator
$           word        ( n -- 0) jump label creator

?LABEL      word        ( addr -- addr)
RESOLVER    word        ( -- )
 Resolves all reference on the label stack( lbladdress ) ( jmpaddr offset)

+CODE       word        ( <name> )  Used to jump across CODE words
CODE        word        ( <name> )

NEWLABELS   code word
ENDCODE     word        ( -- )
L:          word        ( <text> )
Lines: 47
Bytes: 1343
--- End of file ---
```

Example Code with labels

```
                    \ Equivalent structured words
CODE TEST2
1 $: R0 1000 ADDI,  \ BEGIN,
     R0   R1 CMP,
         2 $ JLT,   \ GTE IF,
          R3 CLR,
                    \ ENDIF,
2 $: R1   R0 SUB,
         1 $ JMP,   \ AGAIN,
ENDCODE
```

DSK*.AUTOMOTION

```
Interrupt Driven Sprite motion (like Extended BASIC)  BJF July 21 2019
Nov 2020 - corrected MOTION to correct X vector when Y vector is negative
- Changed ]SMT motion array to machine code. Same size as Forth

Dependancy: DSK*.GRAFIX
Dependancy: DSK*.DIRSPRIT

SMT         constant     SPRITE motion table VDP address
AMSQ        constant     interrupts, software DISABLE bits

AMSQ bit meaning:
 80 all interrupts disabled
 40 motion disabled
 20 Sound disabled
 10 quit key disabled
 access the sprite tables in VDP like arrays

]SMT        code word  ( spr# -- vaddr)
MOVING      word       ( n -- )   # of sprites moving automatically
INITMOTION  word       ( -- )
STOPMOTION  word       ( -- )
AUTOMOTION  word       ( -- )  Enable interrupt motion
MOTION      word       ( vx vy spr# -- )

Lines: 39
Bytes: 1421
--- End of file ---
```

DSK*.BASICHLP

```
Loads TOOLS, INPUT, RANDOM, STRINGS, GRAFIX and CHARSET
Gives Forth training wheels for new programmers. More like TI-BASIC

Includes  : DSK*.TOOLS
Includes  : DSK*.INPUT
Includes  : DSK*.RANDOM
Includes  : DSK*.STRINGS
Includes  : DSK*.GRAFIX
Includes  : DSK*.CHARSET

Lines: 13
Bytes: 334
--- End of file ---
```

DSK*.BGSOUND

```
Dependancy: DSK*.MTASK99

SILENT      word        ( --)
PLAY$       word        ( caddr -- )
PLAYLIST    word        ( addr -- ) ( <> 0)

SHEAD       variable
STAIL       variable
SOUNDQ      create

Q+!         word        ( fifo -- n)
Q@          word        ( fifo -- n)
Q!          word        ( n fifo --)
Q?          word        ( fifo -- ?)
BGPLAYER    word        ( -- )
PLAYER      create
>SNDQ       word        ( list -- )
PLAYQ       word        ( list -- )
KILLQ       word        ( -- )
?BYTE       word        ( c -- )
NUMBUF      create
BYTE        word        ( -- )
/END        word

Lines: 96
Bytes: 2752
--- End of file ---
```

DSK*.BLOCKS

```
Dependancy: DSK*.ANSFILES
Dependancy: DSK*.UDOTR

#BUFF       constant
B/BUF       constant
B/REC       constant
LIMIT       constant
FIRST       constant
B/SEC       constant
PREV        variable
USE         variable
LOWBLK      variable
HIGHBLK     variable
BHNDL       variable
ACTIVE      create

?BLOCKS     word        ( -- )
MASK        code word   ( n -- n)
SEEK        word        ( blk# -- )
RBLK        word        ( adr blk# -- adr)
WBLK        word        ( adr blk# -- )
UPDATE      word        ( -- )
+BUF        word        ( addr1-- addr2)
BUFFER      word        ( n -- addr )
BLOCK       word        ( block# --- addr )
FLUSH       word        ( -- )
EMPTY-BUFFERS word      ( -- )
DF128       word
OPEN-BLOCKS word        ( file$ len -- )
CLOSE-BLOCKSword        ( -- )
MAKE-BLOCKS word        ( n file len -- )

Lines: 147
Bytes: 4027
--- End of file ---
```

DSK*.BLWP

```
BLWP        code word   ( wksp entry -- )
            Perform BLWP from Forth to wksp/entry vector.

Lines: 10
Bytes: 365
--- End of file ---
```

DSK*.BOOLEAN

```
BITS/BYTE   constant
BITS/CELL   constant
BITS:       word        ( n -- )
```

```
BITS/BYTE   create
BITFLD      word       ( bit# bits[] -- bit#' addr)
BITMASK     word       ( bit# -- n )
BIT@        word       ( bit# bits[] -- ?)
BSET        word       ( bit# bits[] -- )
BRST        word       ( bit# bits[] -- )
BTOG        word       ( bit# bits[] -- )
Lines: 29
Bytes: 1106
--- End of file ---
```

DSK*.BREAK

```
?BREAK      word       ( -- )

Lines: 9
Bytes: 153
--- End of file ---
```

DSK*.BUFFER

```
BUFFER:     word     ( n --) <NAME>  create a buffer of n bytes called name

Lines: 3
Bytes: 43
--- End of file ---
```

DSK*.CALLCHAR

```
Dependancy: DSK*.VDPMEM
>NIB        word       ( char -- n)
CALLCHAR    word       ( addr len char --)

Lines: 21
Bytes: 488
--- End of file ---
```

DSK*.CASE

```
CASE        word       ( -- 0 )
OF          word       ( -- )
?OF         word       ( flag -- here )
ENDOF       word       ( -- )
ENDCASE     word       ( -- )

Lines: 28
Bytes: 493
--- End of file ---
```

DSK*.CATALOG

```
Dependancy: DSK*.ANSFILES
Dependancy: DSK*.CASE

U.R         word       ( u n --) ( adr len)
$.          word       ( $addr -- )
NEXT$       word       ( addr len -- addr len )
$.LEFT      word       ( $ width -- )
F>INT       word       ( addr len -- addr len n)
DIR.TYPE    word       ( addr -- )
HEAD.REC    word       ( addr -- ) ( addr len)
DIR.REC     word       ( addr -- ) ( addr len)
PAGEBRK     word       ( -- )
CAT         word       ( <DSK?.> ) ( PAD)

Lines: 90
Bytes: 2451
--- End of file ---
```

DSK*.CHAR

```
CHAR        word       ( -- <c>)
[CHAR]      word       ( -- <c>)

Lines: 9
Bytes: 204
--- End of file ---
```

DSK*.CHARSET

```
GROM        word       ( addr -- )
GC@+        word       ( -- c)
]PDT        word       ( char# -- 'pdt[n] )
]GFONT      word       ( ascii -- grom_adr)
GVMOVE      word       ( grom_addr vdp_addr cnt -- )
CHARSET     word       ( -- )
HICHARSET   word       ( -- )

Lines: 38
Bytes: 1531
--- End of file ---
```

DSK*.CLOCK

```
SECONDS     create
SECONDS++   word       ( -- )
TICKER      constant
1/60        word       ( -- )
1SEC        word       ( -- )
SEXTAL      word
```

```
<:>         word
HOLD        word
<.>         word
##:         word
.TIME       word        ( d -- )
CLOCK       word        ( -- )

Lines: 43
Bytes: 981
--- End of file ---
```

DSK*.CODEMACROS

```
MACRO       word        Create machine code macro
;MACRO      word
DUP,        word        ( n -- n n) compile machine code for DUP
DROP,       word        ( n --)   compile machine for DROP
2*,         word        ( n -- n') compile machine code for 2*
()@,        word        ( addr -- ) compile code for indexed addressed fetch
()!,        word        ( addr -- ) compile code for indexed addressed store
()C@,       word        ( addr -- ) compile index-addressed character @
()C!,       word        ( c addr --) compile index-addressed character !
LIT,        word        ( n -- ) compile code for n as immediate number
@,          word        ( addr --) compile code for a Forth fetch to R4
!,          word        ( n addr -- ) compile code for a Forth store n to *R4
C@,         word        ( addr --) compile code for Forth character fetch
C!,         word        ( n addr --) compile code for Forth character store

Lines: 34
Bytes: 1033
--- End of file ---
```

DSK*.CODEX1

```
Experimental SAMS overlay system.

Dependancy: DSK*.SAMSFTH

CSEG        constant
PLINKS      constant
CREG        constant
TOTAL-AMS   variable
BANK#       variable
PAGE#       variable
HOME        create
CODEX-RESET word        ( -- )
NEWPAGE     word        ( - n)
pages"      code word
CMAP        word        ( bank# -- ) ( bank#)
AMS-HERE    word        ( -- addr)
DICTIONARY  word        ( -- dp context)
RELINK      word        ( dp context -- )
ACTIVATE    word        ( bank# -- )
FAR:        word        ( -- )
```

```
LOCAL:      word       ( -- )
END-LOCAL   word       ( -- )
BANK-MEM    word       ( -- n )
END-SAMS    word       ( -- )
.SAMSCODE   word
CODEX:      word       ( -- ) build a vocabulary in SAMS memory
                              CREATE/DOES>

Lines: 57
Bytes: 1850
--- End of file ---
```

DSK*.COLORS

```
ENUM        word       ( 0 <text> -- n)  Begin an enumerated list

TRANS       Enumerated  Transparent colour parameter
BLACK       Enumerated  Black parameter
MEDGRN      Enumerated  Medium green parameter
LTGRN       Enumerated  Light green parameter
DKBLU       Enumerated  Dark blue parameter
LTBLU       Enumerated  Light blue parameter
DKRED       Enumerated  Dark Red parameter
CYAN        Enumerated  Cyan parameter
MEDRED      Enumerated  Medium red parameter
LTRED       Enumerated  Light Red parameter
DKYEL       Enumerated  Dark Yellow parameter
LTYEL       Enumerated  Light Yellow parameter
DKGRN       Enumerated  Dark green parameter
MAGENTA     Enumerated  Magenta parameter
GRAY        Enumerated  Gray parameter
WHT         Enumerated  White parameter

Lines: 21
Bytes: 277
--- End of file ---
```

DSK*.COMPARE

```
S=          code word  ( addr1 addr2 len --)   compare counted strings
COMPARE     word       ( addr n addr2 n2 -- -1|0|1)   compare stack strings

Lines: 11
Bytes: 344
--- End of file ---
```

DSK*.CRU

```
CRU!        code word  ( CRUaddr -- )
CRU@        code word  ( -- CRUaddr )
0SBO        code word  ( -- )
0SBZ        code word  ( -- )
0TB         code word  ( -- )
```

```
SBO         code word  ( c -- )
SBZ         code word  ( c -- )
TB          code word  ( c -- ? )
Lines: 50
Bytes: 1117
--- End of file ---
```

DSK*.CRU2

```
'R12        user
CRU!        word       ( CRUaddr -- )
CRU@        word       ( -- CRUaddr )
0SBO        code word  ( -- )
0SBZ        code word  ( -- )
0TB         code word  ( -- )
SBO         code word  ( c -- )
SBZ         code word  ( c -- )
TB          code word  ( c -- ? )
LDCR        code word  ( data bits CRU-- )
STCR        code word  ( bits cru --- n )
Lines: 85
Bytes: 2490
--- End of file ---
```

DSK*.DATABYTE

These words are used with CREATE to make data tables that look like Assembly language tables.

```
CREATE TABLE1
HEX
DATA DEAD,BEEF,1234,994A
BYTE 11,22,33,44,AB,BA,FF,0F

BYTE        word       ( -- ) Compile a line of comma delimited bytes
DATA        word       ( -- ) Compile a line of comma delimited integers

Lines: 28
Bytes: 520
--- End of file ---
```

DSK*.DEFER

```
DEFER!      word       ( xt2 xt1 -- )
DEFER@      word       ( 'deferred -- xt)
IS          word       ( xt "<spaces>name" -- ) ( -- XT)
?DEFER      word       ( -- )
DEFER       word       ( -- <text>) CREATE/DOES>

ACTION-OF   word       ( <text> -- xt) ( returns execution token of <text>)

Lines: 30
Bytes: 913
--- End of file ---
```

DSK*.DIR

```
Print the disk directory of the specified disk device

?CR         word        ( -- ) newline if column is past a limit

DIR         word        (<name>) Print directory of <name>
```

DSK*.DIRSPRIT

```
Sprite motion control under direct program control.
 Removed SP.X@, SP.Y@.  Now use: VC@/VC! with SP.Y SP.X

/DIRSPRIT   Marker

Dependancy: DSK*.GRAFIX

VDPSTS      constant
SAT         constant
SATsize     constant
MAX.SP      constant
VDPR1       constant
SPR#        variable

SP.Y        code word  ( spr# -- vaddr)
            * SP.Y is the base address of each 4 byte sprite record

SP.X        code word  ( spr# -- vaddr)
FUSEXY      code word  ( x y spr# -- yyxx spr#)
SP.PAT      word       ( n spr# -- vaddr)
SP.COLR     word       ( n spr# -- vaddr)
?NDX        word       ( n -- n )
DELALL      word       ( -- )
POSITION    word       ( sprt# -- dx dy ) ( ?NDX)
LOCATE      word       ( dx dy sprt# -- )
PATTERN     word       ( char sprt# -- ) ( ?NDX)
SP.COLOR    word       ( col sprt# -- ) ( ?NDX)
SPRITE      word       ( char colr x y sp# -- )
MAGNIFY     word       ( mag-factor -- )
RANGE?      code word  ( n n n -- n')
DXY         code word  ( x2 y2 x1 y1 --- dx dy )
DIST        word       ( x2 y2 x1 y1 -- distance^2)
SP.DISTXY   word       ( x y spr# -- dist^2 )
SP.DIST     word       ( spr#1 spr#2 -- dist^2 )
COINCALL    word       ( -- ? )
COINC       word       ( spr#1 spr#2 tol -- ?) ( -- x1 y1 x2  y2 )
COINCXY     word       ( dx dy sp# tol -- ? ) ( -- x1 y1 x2  y2 )

Lines: 155
Bytes: 5655
--- End of file ---
```

DSK*.DOUBLE

```
Double precision (32bit) operations for Camel99 Forth.

D+          code word  ( lo hi lo' hi' -- d)
2LITERAL    word       ( d -- )
2CONSTANT   word       ( d -- <text>) CREATE/DOES>

2VARIABLE   word       ( d -- <text>)
D0<         word       ( d -- ? )
S>D         word       ( n -- d )
D2*         word       ( d -- d')
D2/         word       ( d -- d') ( -- >8000)
D<          word       ( d d -- ?)
DU<         word       ( d d -- ?)
D0=         word       ( d -- ?)
D=          word       ( d d -- ?)
D>S         word       ( d -- n)
2OVER       code word  ( d1 d2 -- d1 d2 d1)
2ROT        word       ( d d2 d3 -- d2 d3 d)
4DUP        word       ( d d -- d d d d )
2NIP        word       ( d d' -- d')
DMAX        word       ( d d -- d)
DMIN        word       ( d d -- d)
?NEGATE     word       ( n1 n2 -- n3)
DNEGATE     word       ( d1 -- d2 )
D-          word       ( d d -- d)
?DNEGATE    word       ( d1 n -- d2)
DABS        word       ( d -- d)
M*          word       ( n1 n2 -- d) ( signed 16*16->32 multiply)
M*/         word       ( d1 n1 +n2 -- d2 ) ( 52.9 mS LOL! )
DU.         word       ( u -- )
D.R         word       ( d n --)
D.          word       ( d  -- )
?DNUMBER    word       ( addr len -- d ?) convert stack string to Double

Lines: 121
Bytes: 3735
--- End of file ---
```

DSK*.EASYFILE

```
Simpler and smaller libary for disk file access. Text files (DV*) only.

Includes  : DSK*.VALUES

#1          value
#2          value
#3          value
OPEN        word       ( addr len -- fid)
CLOSE       word       ( fid -- )
WRITE       word       ( addr len fid -- )
WRITE$      word       ( caddr fid -- )
LINPUT      word       ( caddr handle -- ) ( -- $ $+1 handle)
AS:         word       ( n -- <value>)
```

```
Lines: 35
Bytes: 1193
--- End of file ---
```

DSK*.ELAPSE

```
Test timer for your programs. Nine (9) minute maximum duration.
* YOU CANNOT CALL KSCAN WHILE TIMING *
\ example
ELAPSE MYWORD

TICKER      constant
SEXTAL      word        Change to base 6
<:>         word       Insert ':' into number format buffer
<.>         word       Insert '.' into number format buffer
TIME$       word       ( n -- addr len) Return time as a stack string
.ELAPSED    word       ( -- ) Print the elapse duration
ELAPSE      word       ( -- <text> ) Measure duration of <text> routine.

Lines: 30
Bytes: 823
--- End of file ---
```

DSK*.ENUM

```
ENUM        word       ( 0 <text> -- n) Create enumerated list of constants

Lines: 19
Bytes: 223
--- End of file ---
```

DSK*.FASTCASE

```
CASE:       word       ( -- <name> ) Name a fast case word
|           word       ( <name> ) Compile XT of <name> into memory
;CASE       word       ( n -- ) End a fast case statement

Lines: 28
Bytes: 680
--- End of file ---
```

DSK*.FLOORED

```
ANS/ISO Forth Floored and symmetric division routines. Forth compatibility only.

?NEGATE     word       ( n1 n2 -- n3)
DNEGATE     word       ( d1 -- d2 )
?DNEGATE    word       ( d1 n -- d2)
DABS        word       ( d -- d)
SM/REM      word       ( d1 n1 -- n2 n3 ) ( Ref. dpANS-6 section 3.2.2.1.)
FM/MOD      word       ( d1 n1 -- n2 n3)
M*          word       ( n1 n2 -- d)
```

```
*/MOD       word       ( n1 n2 n3 -- n4 n5)
S>D         word       ( n -- d)
/MOD        word       ( n1 n2 -- n3 n4)
*           word       ( n n -- n)
/           word       ( n n -- n)
MOD         word       ( n n -- n)
*/          word       ( n n n -- n)
Lines: 54
Bytes: 1903
--- End of file ---
```

DSK*.FORGET

```
CFA>NFA     word       ( cfa -- nfa | 0 ) Find name field address of CFA
FORGET      word       ( <name> )   Move dictionary back to before <name>

Lines: 6
Bytes: 246
--- End of file ---
```

DSK*.GKEY

```
?KEY        word
BLINK       variable
OKEY        variable
RL          constant
RH          constant
KC          variable
RLOG        variable
BR          constant
BLINKER     word
NEWKEY?     word       ( char -- ?)
RKEY?       word       ( -- c ?)
RKEY        word

Lines: 55
Bytes: 1236
--- End of file ---
```

DSK*.GPLMENU

```
GPL         word      ( -- ) Exits Forth back to E/A cartridge menu

Lines: 11
Bytes: 191
```

DSK*.GRAFIX

```
Dependancy: DSK*.VDPMEM
CTAB        constant
PDT         constant
]CTAB       word        ( set# -- 'ctab[n])
]PDT        word        ( char# -- 'pdt[n] )
?MODE       word        ( n -- )
?COLOR      word        ( n -- n )
?SCR        word        ( vdpadr -- )

>COLR       word        ( fg bg - byte)
                  Takes fg nibble, bg nibble, convert to TI hardware number,test
                  for legal values, and combine into 1 byte.

CLEAR       word        ( -- ) Clear screen. Place cursor at bottom line.
COLOR       word        ( character-set fg-color bg-color -- )
SET#        word        ( ascii -- set#)
COLORS      word        ( set1 set2 fg bg  -- ) change character sets set1 to set2
SCREEN      word        ( color -- ) ( -- n)
GRAPHICS    word        ( -- ) ( -- E0)
>DIG        word        ( char -- n)
CALLCHAR    word        ( addr len char --) Set pattern of ascii char from string
PATTERN:    word        ( u u u u -- ) Deprecated. Just use CREATE and comma.
CHARDEF     word        ( addr char# --) Write pattern addr data to ascii char VDP
CHARPAT     word        ( addr char# --) Get VDP pattern of char# to CPU addr
GCHAR       word        ( col row -- char) Read screen char at col row to data stack
HCHAR       word        ( col row char cnt -- ) Write cnt chars to screen at col row
VCHAR       word        ( col row char cnt -- )
                        Write vertically to screen at col row

Lines: 103
Bytes: 3552
--- End of file ---
```

DSK*.HEXNUMBER

```
H#          word        ( hexnumber -- n ) Convert text to Hex number

Lines: 12
Bytes: 520
--- End of file ---
```

DSK*.HEXQUOTE

```
Deprecated:  See CALLCHAR

Dependancy: DSK*.GRAFIX
()          word        ( addr ndx -- addr')
HEX"        word        ( <text> -- addr) Convert 4 integers in data for CHARDEF

Lines: 35
Bytes: 1120
--- End of file ---
```

DSK*.HILITE

```
Dependancy: DSK*.MALLOC
]PDT        word        ( char# -- 'pdt[n] )
VCMOVE      word        ( vaddr1 vaddr2 n --) ( R-- n heap n)
INVERTFONT  word        ( -- )
HITYPE      word        ( addr len --)
HILITE      word        ( Vaddr len --)
NORMAL      word        ( Vaddr len --)

Lines: 26
Bytes: 928
--- End of file ---
```

DSK*.INLINE

```
Not portable Forth code**  Uses TMS9900/CAMEL99 CARNAL Knowledge

Dependancy: DSK*.CASE

/INLINE     Marker
NOTCODE?    word        ( -- ?)
HEAP        word        ( -- addr)
HALLOT      word        ( n -- )
HEAP,       word        ( n -- )
'NEXT'      constant
CODE,       word        ( xt --) Expand XT of code word into memory.
DUP,        word
LIT,        word        ( n -- )
INLINE[     word        ( -- addr)
@           code word
C@          code word
DROP        code word

Lines: 91
Bytes: 3228
--- End of file ---
```

DSK*.INPUT

```
Similar to BASIC INPUT statement. Separate word needed for string or number

$ACCEPT     word        ( $addr -- )
$INPUT      word        ( $addr -- )
#INPUT      word        ( variable -- )

Lines: 30
Bytes: 717
--- End of file ---
```

DSK*.ISRSOUND

```
Dependancy: DSK*.VDPMEM

AMSQ        constant    ( -- addr) Address of AMSQ byte in TI-99 system
?BYTE       word        ( n -- ) Abort is n is not a byte value
VBYTE       word        ( -- ) Compile comma delimited bytes into VDP RAM
/VEND       word        ( -- ) End a sound list in VDP RAM
0LIMI       code word   ( -- ) Disable interrupts
2LIMI       code word   ( -- ) Enable interrupts
ISRPLAY     word        ( vaddr -- )  Play sound list at VDP address Vaddr.

Lines: 58
Bytes: 1569
--- End of file ---
```

DSK*.ISRSUPPORT

```
?CODE       word        ( cfa -- )
ISR'        word        ( -- code-address)
INSTALL     word        ( isr-addr -- )

Lines: 31
Bytes: 925
--- End of file ---
```

DSK*.JOYST

```
JOYST       code word   ( joystick# -- value )
Lines: 24
Bytes: 882
--- End of file ---
```

DSK*.LINPUT

```
LINPUT      word        ( addr handle -- ) Read file record into addr buffer

Lines: 10
Bytes: 377
--- End of file ---
```

DSK*.LISTS

```
{NIL}       create
{           word       ( -- )
}           word       ( -- )
"           word       ( -- )
{NEXT}      word       ( list -- list' )
{$}         word       ( link -- $)
{NTH}       word       ( list n -- $addr )
{PRINT}     word       ( link -- )
{LEN}       word       ( list -- n )
{LIST}      word       ( list -- )
{WITH       word
MAP}        word       ( list xt -- )
MFLAG       variable
POSITION    variable
{MEMBER}    word       ( $ {list} -- -1 | ndx )

Lines: 90
Bytes: 2386
--- End of file ---
```

DSK*.LOADER

```
Loads a binary image into memory

/LOADER     Marker
Dependancy: DSK*.LOADSAVE
LASTCHAR++  word       ( Caddr --)
FIELD       word       ( n -- Vaddr)
BLOAD       word       ( addr len -- ?)
LOADER      word       ( addr len -- )

Lines: 21
Bytes: 647
--- End of file ---
```

DSK*.LOADSAVE

```
W/O100      constant
R/O100      constant
NEWPAB      word       (  file$ len VDPaddr #bytes mode -- )
POPPAB      word       ( -- )
SAVE-FILE   word       ( file$ len VDPaddr size mode -- )
LOAD-FILE   word       ( file$ len VDPaddr size mode -- )
SAVE-FONT   word       ( file$ len --)
LOAD-FONT   word       ( file$ len --)

Lines: 41
Bytes: 1472
--- End of file ---
```

DSK*.LOWTOOLS

Loads tools into LOW RAM so you have more space for your program in High RAM. After compiling and testing you can remove all the tools with REMOVE-TOOLS command.

```
Dependancy: DSK*.MALLOC

SAVEDP      variable
KEEP        constant
Includes  : DSK*.ELAPSE
Includes  : DSK*.TOOLS
Includes  : DSK*.ASM9900
REMOVE-TOOLS word       ( -- )
~           word     dummy word to mark dictionary end in LOW RAM.
Lines: 36
Bytes: 892
--- End of file ---
```

DSK*.MALLOC

```
MALLOC      word       ( n -- addr )
MFREE       word       ( n -- )
Lines: 7
Bytes: 153
--- End of file ---
1
DSK*.MARKER

MARKER      word       ( -- ) CREATE/DOES>

LOCK        word

Lines: 22
Bytes: 585
--- End of file ---
```

DSK*.MORE

```
Dependancy: DSK*.ANSFILES

MORE        word       ( <filename>)

Lines: 28
Bytes: 673
--- End of file ---
```

DSK*.MOTION

```
Dependancy: DSK*.DIRSPRIT

VECTORS     create
```

```
]VECTOR     code word  ( spr# -- addr)
VECT+       code word  ( dx dy x y -- x' y')
SP.MOVE     word       ( spr# -- )
ALL-SPRITES word       ( -- 1st last)
TRANSLATE   word       ( 1st last  -- )
MOTION      word       ( dy dx spr# -- )
MOTIONS     word       ( dy dx first last -- )

Lines: 66
Bytes: 2090
--- End of file ---
```

DSK*.MSTAR

```
?NEGATE     word       ( n1 n2 -- n3)
DNEGATE     word       ( d1 -- d2 )
?DNEGATE    word       ( d1 n -- d2)
M*          word       ( n1 n2 - d)

Lines: 11
Bytes: 321
--- End of file ---
```

DSK*.MTASK99

```
FROM        code word
MYSELF      code word  ( -- pid) Returns a tasks program identifier (PID)
USER0       constant    The TI-99 console Forth is called USER0
USIZE       constant    Default size of a task's user area. (192 bytes)
'SP         user       Forth DATA stack register (R6) USER variable
'RP         user       Forth Return stack register (R7) USER variable
'IP         user       Forth Interpreter register (R9) USER variable
'R10        user       Holds address of NEXT interpreter
TLINK       user       Link to next task's workspace
TPC         user       Next task's program counter
TST         user       Next task's status register

'YIELD      create     ( -- addr) Address of the task switcher code routine
'TSTAT      constant    ( -- addr) Address of the 2nd half of switcher code
'NEXT       constant    ( -- addr) Address of Forth's NEXT code (>838A)

LOCAL       word       ( PID uvar -- addr' )
                        return task local address of a USER variable
SLEEP       word       ( PID -- ) Reset TFLAG user variable of task(PID)
WAKE        word       ( PID -- ) Set TFLAG user variable of task(PID)
SINGLE      word       ( -- ) Patch PAUSE to run NEXT (no switching)
MULTI       word       ( -- ) Patches the word PAUSE to run YIELD

** YOU  M U S T  use INIT-MULTI before multi-tasking **

INIT-MULTI  word       ( -- ) Set up Forth for multi-tasking
FORK        word       ( PID -- ) Copy workspace of active task to PID's workspace
JOB->IP     word       ( xt pid -- xt pid) Used by ASSIGN
```

```
ASSIGN      word        ( xt pid -- ) Assign word XT to run in TASK pid
RESTART     word        ( pid -- ) Restarts a TASK.
COLD        word        ( -- ) COLD redefined to stop multi-tasker first
BYE         word        ( -- ) BYE redefined to stop multi-tasker first

PAUSE       word       ( -- )
                        Defined in the kernel. Patched by the word MULTI to become
                        the task switcher.

Lines: 135
Bytes: 5453
--- End of file ---
```

DSK*.MTOOLS

```
Tools for monitoring the status of the system when multi-tasking

PULSE       word        ( -- n) Returns the round robin time of PAUSE
.AWAKE      word        ( ? -- ) Sub-routine used by .TASKS
.LOC        word        ( adr -- ) Sub-routine used by .TASKS
.TASK       word        ( pid -- ) Sub-routine used by .TASKS
.HDR        word        ( -- ) Sub-routine used by .TASKS
NEXTJOB     word        ( pid -- pid') Sub-routine used by .TASKS
.TASKS      word        ( -- ) Show all tasks and their status as table
MONITOR     word        ( -- ) Continuous display of system PULSE. FCTN 4 to stop.
JOBCOUNT    word        ( -- n ) Returns tasks that are created
STOPALL     word        ( -- ) Stop all tasks in the queue except USER0
WAKEALL     word        ( -- ) Wake all tasks in the queue

Lines: 89
Bytes: 2029
--- End of file ---
```

DSK*.NEEDFROM

```
NEEDS       word        ( -- ?) Search dictionary for <name>, return flag
FROM        word        ( ? <dsk*.name> -- ) Load file <dsk*.name> if flag is FALSE

Lines: 11
Bytes: 199
--- End of file ---
```

DSK*.PROG

```
Creates sub-programs with separate work spaces that are called by name like Forth
words but use BLWP internally. Very neat.
Dependancy: DSK*.ASM9900

PROG:       word        ( wksp -- )
,           create      ( *W BLWP, )
;PROG       word        ( -- )
[TOS]       word        ( -- )
Lines: 17
Bytes: 568
--- End of file ---
```

DSK*.QUITKEY

```
QUIT-OFF    word       ( -- ) Disable QUIT key
QUIT-ON     word       ( -- ) Enable Quit key

Lines: 5
Bytes: 128
--- End of file ---
```

DSK*.RANDOM

```
SEED        constant
RNDW        word       ( -- n )
RND         word       ( n -- 0..n-1 )
RANDOMIZE   word       ( -- )
Lines: 13
Bytes: 455
--- End of file ---
```

DSK*.RKEY

```
Repeating key word. Can be used in place of KEY
LD          constant
SD          constant
OUTKEY      variable
OLDKEY      variable
RPT         variable
RKEY?       word       ( -- char) ( dly)
RKEY        word       ( -- char)

Lines: 44
Bytes: 729
--- End of file ---
```

DSK*.SAMS

```
Access SAMS memory card in 64K segments. For 1Mbyte card.
* RUN SAMSINI before using the card ON REAL TI99 *

BANK#       variable
PMEM        constant
SEG         variable
SEGMENT     word       ( 1..F -- )
SAMS-OFF    code word  ( --)
SAMS-ON     code word  ( -- )
SAMSINI     code word
PAGED       code word  ( addr -- paged_address )

Lines: 81
Bytes: 2839
--- End of file ---
```

DSK*.SAMSBLOCK

```
SBLOCK is like Forth BLOCK but it pulls in SAMS memory in 4K blocks. Uses two Low-
RAM buffer.

'R12        user
SAMSCARD    word       ( -- )
DMEM        constant
SREG        constant
BANK#       variable
1SBO        code word  ( -- )
1SBZ        code word  ( -- )
SAMS-ON     word       ( -- )
SAMS-OFF    word       ( -- )
SAMSINI     code word
SBLOCK      code word  ( bank# -- addr )

Lines: 63
Bytes: 2006
--- End of file ---
```

DSK*.SAMSDUMP

```
Dependancy: DSK*.SAMS
Dependancy: DSK*.TOOLS

@P          word       ( addr -- n)
C@P         word       ( addr - n)
SDUMP       word

Lines: 9
Bytes: 190
--- End of file ---
```

DSK*.SAMSFTH

```
'R12        user
SAMSCARD    word       ( -- )
DMEM        constant
SEG         variable
BANK#       variable
0SBO        code word  ( -- )
0SBZ        code word  ( -- )
1SBO        code word  ( -- )
1SBZ        code word  ( -- )
SAMS-ON     word       ( -- )
SAMS-OFF    word       ( -- )
SEGMENT     word       ( 1..F -- )
SAMSINI     word       ( -- 4100 4000)
DMAP        word       ( bank# -- ) ( bank#)
PAGED       word       ( virtual-addr - real-addr)

Lines: 74
Bytes: 2123
--- End of file ---
```

DSK*.SAVESYS

Given an execution token and a file path, SAVESYS saves Forth as an E/A5 binary program.

```
/SAVESYS    Marker
Dependancy: DSK*.LOADSAVE


'ORG        constant
VDPBUFF     constant
8K          constant
PROG        constant
MULTIFLAG   constant
PROGSIZE    constant
LOADADDR    constant
CODEORG     constant
SYS-SIZE    word        ( -- n)
#FILES      word        ( -- n)
CODECHUNK   word        ( n -- addr) ( -- n addr)
CHUNKSIZE   word        ( n -- n )
LASTCHAR++  word        ( Caddr len --)
?PATH       word        ( addr len -- addr len )
SAVESYS     word        ( XT -- <textpath> ) ( caddr len )

Lines: 72
Bytes: 2353
--- End of file ---
```

DSK*.SBLOCKS

```
REMOVE      Marker
'R12        user
SAMSCARD    word        ( -- )
0SBO        code word   ( -- )
0SBZ        code word   ( -- )
1SBO        code word   ( -- )
1SBZ        code word   ( -- )
SAMS-ON     word        ( -- )
SAMS-OFF    word        ( -- )
SAMSINI     word        ( -- 4100 4000)
USE         variable
BLK#S       create
WINDOWS     create
BLOCK       code word   ( bank -- buffer)
SEG         variable
SEGMENT     word        ( 1..F -- )
PAGED       word        ( virtual-addr -- real-addr)

Lines: 120
Bytes: 2980
--- End of file ---
```

DSK*.SCRCAPTURE

```
Dependancy: DSK*.ANSFILES

HNDL        variable
(CAPTURE)   word        ( -- )
CAPTURE     word        ( <PATH> ) ( -- caddr len )
Lines: 26
Bytes: 848
--- End of file ---

DSK*.SCREENS

SCREEN      word        ( n -- )
SCREEN:     word        ( scr# fg bg -- )  CREATE/DOES>

Lines: 21
Bytes: 458
--- End of file ---
```

DSK*.SEARCH

```
Dependancy: DSK*.COMPARE

2OVER       code word   ( a b c d --  a b c d a b)
2NIP        word        ( a b c d -- c d )
SEARCH      word        ( caddr1 u1 caddr2 u2 -- caddr3 u3 flag)

Lines: 35
Bytes: 1129
--- End of file ---
```

DSK*.SEE

```
Dependancy: DSK*.CASE
Dependancy: DSK*.TOOLS

CFA>NFA     word        ( cfa -- nfa | 0 )
VALIDWORD?  word        ( nfa -- nfa | 0 )
LOOKUP      word        ( <text> -- cfa)
IMMFLAG     variable
$NEXT       constant
'EXIT       constant
'(S")       constant
'DOVAR      constant
'DOCON      constant
'DOUSER     constant
TAB         word        ( -- )
?NEWLINE    word        ( -- )
IMMED?      word        ( nfa -- f )
CLEANSTK    word        ( -- )
.VARIABLE   word        ( nfa cfa -- nfa)
"           variable
.CONSTANT   word        ( nfa cfa -- nfa)
```

```
"           constant
.USER       word        ( nfa cfa -- nfa)
"           user
.EXIT       word        ( cfa -- )
'"'         constant
.SQUOTE     word        ( addr -- addr' )
?CODE       word        ( cfa -- )
?COLON      word        ( cfa -- )
.CODEWORD   word        ( NFA CFA -- )
"           code word
.COLONWORD  word        ( cfa -- )
DECOMPILE   word        ( nfa cfa -- )
"           word        ( cfa)
DATA-DECODERword        ( cfa -- ) ( -- xt)
SEE         word        (  -- <string> )

Lines: 148
Bytes: 4098
--- End of file ---
```

DSK*.SMPLSND

```
>DOUBLE     word        ( addr len -- d)
f(clk)      create
>FCODE      word        ( 0abc -- 0cab)
HZ>CODE     word        ( freq -- fcode )
[HZ]        word        ( freq -- fcode )
FREQ!       word        ( fcode -- )
]HZ         word        ( freq -- )
ATT         word        ( n -- )
MUTE        word        ( -- )
WAIT        word        ( n -- )
DECAY       word        ( speed -- )
ATTACK      word        ( speed -- )

Lines: 45
Bytes: 1601
--- End of file ---
```

DSK*.SOUND

```
OSC1        constant
OSC2        constant    ( oscillators take 2 nibbles)
OSC3        constant
OSC4        constant    ( noise takes 1 nibble) ( 0= max, 15 = off)
ATT1        constant
ATT2        constant
ATT3        constant
ATT4        constant    ( OSC4 volume adjust)
(CLK)       create
f(clk)      word        ( -- d)
>FCODE      code word   ( 0abc -- 0cab)
OSC         variable
```

```
ATT          variable
HZ>CODE      word        ( freq -- fcode )

GEN!         word        ( osc att -- )
GEN1         word        ( -- )
GEN2         word        ( -- )
GEN3         word        ( -- )
GEN4         word        ( -- )
NOISE        word        ( n -- )
(HZ)         word        ( f -- n)
(DB)         word        ( level -- c)
HZ           word        ( f -- )
DB           word        ( level -- )
MUTE         word        ( -- )
SILENT       word        ( --)
WOOP         word
SWEEP        word
SIREN        word

Lines: 83
Bytes: 2611
--- End of file ---
```

DSK*.SPRITES

```
MAX.SP       constant
SDT          constant
SPR#         constant
VDPSTS       constant
STAB         constant
]SDT         word        ( char# -- sdt[n])
]STAB        word        ( char# -- stab[n])
?SPR#        word        ( n -- n )
->PAT        word        ( addr -- )
->COLR       word        ( addr -- )
ERASE        word        ( addr cnt -- )
DELALL       word        ( -- )
* You must run SP.WRITE to affect sprites on the screen ***
PATTERN      word        ( char sprt# -- ) ( ?SPR#)
LOCATE       word        ( dx dy sprt# -- ) ( ?SPR#)
POSITION     word        ( sprt# -- dx dy ) ( ?SPR#)
SPCOLOR      word        ( col sprt# -- )
SPRITE       word        ( colr pat y x Spr# --  )
SP.DEL       word        ( # -- )
MAGNIFY      word        ( mag-factor -- ) (
factored DIST out from SPRDISTXY in TI-Forth)
DIST         word        ( x2 y2 x1 y1 -- distance^2)
SP.DIST      word        ( #1 #2 -- dist^2 )
SP.DISTXY    word        ( x y # -- dist^2 )
2(X^2)       word        ( n -- 2(n^2)
COINC        word        ( sp#1 sp#2 tol -- ? ) ( 0 = no coinc  ) ( <= )
COINCXY      word        ( dx dy sp# tol -- ? )
COINCALL     word        ( -- ? )
SP.WRITE     word        ( -- ) ( 694 dictionary bytes, 128 bytes of HEAP )
```

```
Lines: 125
Bytes: 4873
--- End of file ---
```

DSK*.SQUOTE

```
* Load this at start of program if you use the heap dynamically. It creates a
buffer in Low RAM. The buffer is a circular buffer for inputting multiple strings
with S”

Dependancy: DSK*.MALLOC
SBUF        constant
P           variable
POOL        word        ( -- addr)
P+!         word        ( n -- )
S"          word        ( -- ) ( -- adr len)
Lines: 37
Bytes: 1374
--- End of file ---
```

DSK*.STACKS

```
LIFO:       word        ( #items -- ) Create a named stack.
CELL+@      word        ( addr -- addr' n )
CELL-       word        ( n -- n')
STACK-SIZE  word        ( 'stack -- n )
STACK-DEPTH word        ( stack-addr -- n )
CELL+!      code word   ( addr -- ) ( *TOS INCT, DROP, )
CELL-!      code word   ( addr -- ) ( *TOS DECT, DROP, )
PUSH        word        ( n stack-adr - )
POP         word        ( stack-adr -- n )

Lines: 38
Bytes: 1086
--- End of file ---
```

DSK*.START

```
START is read when CAMEL99 boots. Used to load DSK*.SYSTEM and whatever else you
want loaded when the COLD command is invoked.

Dependancy: DSK*.LOADSAVE

Lines: 28
Bytes: 564
--- End of file ---
```

DSK*.STOD

```
This word is in KERNEL V2.5 and higher. Source provided for your interest.

S>D         word       ( n -- d ) Convert single signed number to double number

Lines: 3
Bytes: 29
--- End of file ---
```

DSK*.STRINGS

```
RE-ENTRANT STRING LEXICON                    OCT 8 1987 Brian Fox

Dependancy: DSK*.COMPARE
Dependancy: DSK*.MALLOC

MXLEN       constant
SSW         constant
$STACK      constant
SSP         variable   String stack pointer
NEW:        word       ( -- ) Make space on the string stack for a string
COLLAPSE    word       ( -- ) Remove everything from the string stack
TOP$        word       ( -- $) The top item on the string as a counted string
+PLACE      word       ( addr n $ -- ) Concatenate stack string to $
SPUSH       word       ( addr len -- top$ )
                       Push stack string to string stack. Return TOP$

?SSP        word       ( -- ) Check for string stack underflow
DROP$       word       ( -- ) Remove top item of string stack
LEN         word       ( $ -- n ) Return the length of a string
SEG$        word       ( $ n1 n2 – top$)
                       Cut $ at position n1, width n2 to string stack

STR$        word       ( n – top$)  Convert n to string on string stack
VAL$        word       ( adr$ - # ) Convert counted string to number at base
CHR$        word       ( ascii# -- top$ ) Convert Ascii # to a string
ASC         word       ( $ -- c) Return ascii value of 1st character $
&           word       ( $1 $2 – top$) Combine $2 to $1
POS$        word       ( $1 $2 -- n ) Find position of $1 in $2
CPOS        word       ( $ char -- n) Find char position in $
COPY$       word       ( $1 $2 -- ) Copy $1 to $2
COMPARE$    word       ( $1 $2 -- -1|0|1) ANS string Compare word
=$          word       ( $1 $1 -- flag) Compare strings as equal
<>$         word       ( $1 $1 -- flag) Compare strings not equal
>$          word       ( $1 $2 -- flag) Compare if $1 > $2
<$          word       ( $1 $2 -- flag) compare is $1 < $2
="          word       ( $addr -- <text> ) Compile time string assignment
=""         word       ( $addr -- ) Set string to null
?$IZE       word       ( n -- ) Test n to be legal string size.
DIM         word       ( n -- ) Create a string of n bytes in dictionary
HDIM        word       ( n -- ) Create a string of n bytes in HEAP memory
PUT         word       ( $1 $2 -- ) Copy $1 into $2
PRINT$      word       ( $ -- ) Print a string on same line
PRINT       word       ( $ -- ) Print a string on new line
(")         word       ( -- ) Internal use.
"           word       ( -- ) " create a string up to next quote character

Lines: 119
Bytes: 3804
--- End of file ---
```

DSK*.STRUC12

```
Forth 2012 Structure creation words

+FIELD      word       CREATE/DOES> Primitive makes a new field in a structure
FIELD:      word       ( n1 "name" -- n2 ; addr1 -- addr2 ) integer field
2FIELD:     word       ( d1 "name" -- d2 ; addr1 -- addr2 ) double field
CFIELD:     word       ( n1 "name" -- n2 ; addr1 -- addr2 ) char field
CELLS:      word       ( n -- ) define field of n cells in width
?STRING     word       ( n -- n) Test string field is <255 bytes
CHARS:      word       ( n -- ) define field of n characters in width ie: a string

Lines: 33
Bytes: 1020
--- End of file ---
```

DSK*.SUPERTOOLS

```
Compile tools into SUPERCART memory at >6000. Leaves more space for your program.

/SCTEST     Marker     Removes supertools code during testing cycles. Don't use.
?SUPERCART  word       Internal use only
SAVEDP      variable   Internal use only
KEEP        constant   Internal use only

Includes  : DSK*.WORDLISTS
Includes  : DSK*.ELAPSE
Includes  : DSK*.TOOLS
Includes  : DSK*.ASM9900
Includes  : DSK*.ASMLABELS

REMOVE-TOOLS   word       ( -- ) Removes super-tools. Keep normal dictionary.

Lines: 49
Bytes: 1061
--- End of file ---
```

DSK*.SYNONYM

```
SYNONYM     word   CREATE/DOES> word to create a synonym for another word

Lines: 19
Bytes: 592
--- End of file ---
```

DSK*.SYSTEM

```
The system file is loaded at boot time or when COLD is invoked. It compiles some
extra words into Camel99 that are part of ANS/ISO Forth.

PARSE-NAME  word       ( <text> -- adr len ) Parse space delimited word from input
NEEDS       word       ( -- ?)  Test dictionary for presence of a word
FROM        word       ( ? -- )  Load specified filename if flag on stack is false.
```

```
INCLUDE     word       ( <text>) Load the file specified by <text>.
CODE        word       ( -- ) Define a new Machine code or Assembler word.
NEXT,       word       ( -- ) Compile address of Forth NEXT into a code word.
ENDCODE     word       ( -- ) End a CODE word definition. Check stack for change.
;CODE       word       Define code section of a CREATE word.
ALIAS       word       ( <newname> <oldword> )
                        Define a fast synonym of a code word.

POSTPONE    word       ( <name> -- ) Delay compilation of <name>
CHARS       word       ( n -- n) Compute size of n characters. (NOOP on 9900)
CHAR        word       ( -- <c>) Return ascii value of CHAR <c>
[CHAR]      word       ( -- <c>) Compile ascii value of <c> into a Forth word.

Lines: 35
Bytes: 1012
--- End of file ---
```

DSK*.TOOLS

Tools is a group of handy tools for programmers. It is used while developing programs but is normally not needed for a final program.

```
Dependancy: DSK*.FORGET
Dependancy: DSK*.CASE
Dependancy: DSK*.DEFER

<.>         word
(.)         word       ( n -- )
FAST#S      word
SMART#S     word
?           word       ( adr -- )
.S          word       ( -- )
SPACEBAR    word       ( -- )
?BREAK      word       ( -- )
.ID         word       ( NFAaddr --)
WORDS       word       ( -- )
.####       word       ( n --)
.ASCII      word       ( adr n --)
?80         word       ( -- 16 | 8)
(DUMP)      word       ( offset n -- )
EMIT        word
DUMP        word
VDUMP       word
UNUSED      word       ( -- u)
.FREE       word

Lines: 115
Bytes: 2625
--- End of file ---
```

DSK*.TRACE

```
Dependancy: DSK*.TOOLS

CFA>NFA     word        ( cfa -- nfa)
XT.ID       word        ( xt -- )
?FREEZE     word        ( -- )
TRACE       variable
(TRACE)     word        ( -- )
:           word        ( -- )

Lines: 43
Bytes: 1339
--- End of file ---
```

DSK*.TRAILING

```
-TRAILING   code word   ( addr len – addr len') Strip trailing blanks from string

Lines: 29
Bytes: 694
--- End of file ---
```

DSK*.TRIG

```
SINTAB      create      ( 182 bytes) Table of sin values
2*,         word        ( n -- 2(n)  Fast access macro to compute 2*
+@,         word        ( addr -- ) ( addr) Index addressing macro
]SIN        code word   ( ndx -- sin)  Expose SINTAB as an array.
90^         constant
180^        constant
360^        constant
(SIN)       word        ( n -- n)
SIN         word        ( n -- n ) Return SIN(n)
COS         word        ( n -- n ) Return COS(n)
Lines: 46
Bytes: 1644
--- End of file ---
```

DSK*.UDOTR

```
UD.R        word        ( ud n --)
U.R         word        ( u n -- )
.R          word        ( u n -- )

Lines: 14
Bytes: 346
--- End of file ---
```

DSK*.VALUES

```
VALUE       word
;           constant
TO          word        ( n -- )
+TO         word        ( n -- )

Lines: 21
Bytes: 491
--- End of file ---
```

DSK*.VBYTEQ

```
Dependancy: DSK*.VDPMEM

VBYTEQ:     word        ( size -- addr )

^TAIL       word        ( fifo -- vaddr)
^HEAD       word        ( fifo -- vaddr)
TAIL+!      word        ( FIFOaddr --)
HEAD+!      word        ( FIFOaddr --)
Q@          word        ( fifo -- n)
QLEN        word        ( fifo -- n)
Q?          word        ( fifo -- ?)
Q!          word        ( n fifo -- )
QRST        word        ( fifo -- )

Lines: 47
Bytes: 1163
--- End of file ---
```

DSK*.VDPBGSND

```
Dependancy: DSK*.MTASK99
Dependancy: DSK*.VDPMEM

?BYTE       word       ( n -- )
VBYTE       word       ( -- )
/VEND       word
SILENT      word       ( -- )
VPLAY$      word       ( VDP_sound_string -- )
VPLAYLIST   word       ( Vaddr -- )
SHEAD       variable
STAIL       variable
SOUNDQ      create
Q+!         word       ( fifo -- n)
Q@          word       ( fifo -- n)
Q!          word       ( n fifo --)
Q?          word       ( fifo -- ?)
BGPLAYER    word       ( -- )
PLAYER      create
>SNDQ       word       ( list -- )
PLAYQ       word       ( list -- )
KILLQ       word       ( -- )

Lines: 95
Bytes: 2823
--- End of file ---
```

DSK*.VDPDOTQ

```
Includes  : DSK*.VDPMEM
VS,         word       ( $adr len-- )
VDPTYPE     word       ( vdp_addr len -- )
."          word       ( <tex> )
PRINT."     word
Lines: 33
Bytes: 992
--- End of file ---
1
```

DSK*.VDPEXTRAS

```
Used only by CAMEL99 TTY version to allow DSK1.GRAFIX to compile.

>VPOS       code word  ( col row -- vaddr)
VCLIP       code word  ( lim char addr -- lim char addr)

Lines: 30
Bytes: 803
--- End of file ---
```

DSK*.VDPLIB

```
VC@          word       ( VDP-adr -- char )
V@           word       ( VDPadr -- n)
VREAD        word       ( VDP-adr RAM-addr cnt  -- )
VWRITE       word       ( RAM-addr VDP-addr cnt -- )
VC!          word       ( char vaddr -- )
V!           word       ( n Vaddr  -- )
VFILL        word       ( VDP-addr count char-- )
VWTR         word       ( c reg -- )

Lines: 115
Bytes: 2671
--- End of file ---
```

DSK*.VDPMEM

```
VP           variable
VHERE        word       ( -- addr)
VALLOT       word       ( n -- )
VC,          word       ( n -- )
V,           word       ( n -- )
VCOUNT       word       ( vdp$adr -- vdpadr len )
VCREATE      word       ( <text> -- )

Lines: 14
Bytes: 415
--- End of file ---
```

DSK*.VDPSAVE

```
Dependancy: DSK*.VDPMEM

PRGRM        constant
NEWPAB       word       (  file$ len VDPaddr #bytes mode -- ) Create new PAB
POPPAB       word       ( -- ) Remove new PAB
VDPUSED      word       ( -- Vaddr size) Compute VDP RAM used from >1000
SAVE-VDP     word       ( file$ len VDPaddr #bytes mode -- ) Save VDP RAM as program
LOAD-VDP     word       ( file$ len VDPaddr #bytes mode -- ) Load program to VDP RAM

Lines: 37
Bytes: 1124
--- End of file ---
```

DSK*.VDPSOUND

```
Includes  : DSK*.VDPMEM
SILENT       word       ( --)
VPLAY$       word       ( sound_string -- )
VPLAYLIST    word       ( addr -- )
?BYTE        word       ( n -- )
VBYTE        word       ( -- )
```

```
/VEND       word

Lines: 40
Bytes: 1343
--- End of file ---
```

DSK*.VDPSTRNG

```
Strings are stored in VDP RAM as counted strings. (1st byte is length)
When we invoke the string's name it returns a VDP address and a length
onto the Forth DATA stack. This is called a "stack string".
The power of this method is we don't cut strings to make a sub-strings.
We simply adjust the address and length on the Data stack. Very fast!

Dependancy: DSK*.VDPMEM

VGET$       word       ( VDP-adr len addr -- )
VS,         word       ( $adr len-- )
VCOUNT      word       ( vdp$adr -- vdpadr len )
VTYPE       word       ( vdp_addr len -- )
VASC        word       ( vaddr len -- c)
VLEN        word       ( vaddr len -- vaddr len)
VAL$        word       ( vaddr len - # )
VLEFT$      word       ( vaddr len len' -- vaddr len')
VRIGHT$     word       ( vaddr len len' -- vaddr len')
VSEG$       word       ( vaddr len n1 n2 -- vaddr len)
VSTR$       word       ( n -- addr len)
VCHR$       word       ( ascii# -- addr len)
VTRIM       word       ( adr len char -- adr len')
VSKIP       word       ( adr len char -- adr len')
V+CHAR      word       ( char addr len -- )
VPRINT      word       ( vaddr len -- )
V$!         word       ( addr len  vaddr len -- )
:="         word       ( vaddr len -- )
VCOMPARE    word       ( adr1 n1 adr2 n2 -- -1|0|1 )
```

DSK*.VTYPE

```
: VTYPE     word  ( adr len  ) Fast type with no scrolling or screen protection
: AT"       word  ( x y -- ) Compiling only. Fast type at X/Y location

Lines: 8
Bytes: 375
--- End of file ---
```

DSK*.WORDLISTS

```
Code adapted from Web: https://forth-standard.org/standard/search
Dec 2020: Removed SET-CURRENT to save precious bytes
Jan 5, 2020: back migrated some enhancements from CODEX work

'wid' is a word-list ID.
In Camel Forth, wid is a pointer to a Name Field Address (NFA)
It is a counted string of the last word defined in the wordlist.

This implementation uses a CONTEXT array defined in the kernel to hold the
ROOT wordlist plus 8 user defined wordlists. The CURRENT variable is defined in the
Camel99 Kernel.
```

Terms to Understand:

- CONTEXT holds the wordlists that are searchable by the interpreter.
- CURRENT holds the wordlist where new definitions will be created.
- Search-order is the order that FIND will search for words in each CONTEXT wordlist. There are a maximum of 8 that you set to be searched
- ROOT is a wordlist that is always visible to FIND. It is a small default wordlist that allows us to set the search order no matter what wordlists have been removed from the search order.

```
WID-LINK    variable    Pointer to the most recently defined wordlist
#ORDER      variable    No. of active wordlists starts at 1

WORDLIST    word        ( -- wid)   return a wordlist data structure

.WID        word        ( wid -- ) print name of a word list

FORTH-WORDLIST    WORDLIST  Name of the Forth wordlist
ROOT        WORDLIST        Wordlist that is always accessible

]CONTEXT    word        ( n -- addr)      context array 8 cells

GET-ORDER   word        ( -- widn ... wid1 n ) Put search order on data stack
SET-ORDER   word        ( wid1x ... wid1 n -- )Set search order from the data stack
ONLY        word        ( -- ) set search order to ROOT ROOT

SET-CONTEXT word        ( wid -- ) place 'wid' at beginning of search order
```

User API

```
ALSO        word        ( -- ) Duplicate the 1st wordlist in the search order
PREVIOUS    word        ( -- ) Return to previous search order
DEFINITIONS word        ( -- ) Set context vocabulary to current.

VOCABULARY  word        ( wid -- )
                        Create wordlist that makes itself first in search order

ORDER       word        ( -- ) Display search order of all wordlists & show CURRENT
FORTH       word        ( -- ) Forth wordlist as a vocabulary. Makes itself 1st.

--- End of file ---
```

DSK*.XASM9900

```
CROSS ASSEMBLER for Camel99 Forth   Brian Fox 2021 )  Notes:
Compare instruction has been changed to CMP,to remove name conflict with C,
Changed A, and S, to ADD, SUB,
MAR 30 2021 Added enumerated labels

This assembler syntax is the same as ASM9900 but it Assembles code into the memory
specified by the TDP variable. TDP is set with the ORG command.

Dependancy: DSK1.TOOLS
Dependancy: DSK1.CASE
Dependancy: DSK1.DEFER


TDP          variable    'target' dictionary pointer. Where code is assembled.
ORG          word        ( addr -- ) Set the code origin.
THERE        word        ( -- addr)  Target HERE. The next free target memory address
TALLOT       word        ( n -- ) Target ALLOT n bytes of memory
TC!          word        ( n addr --) Target character store
T,           word        ( n -- ) Target comma. Compile n into memory. Bump TDP.
TC,          word        ( c -- ) Target char comma. Compile char into memory

#FWD         constant    Internal use only for labels
#LABELS      constant    Maximum number of labels we can use. (20)
FS0          create      Forward reference stack base address
FSP          create      Forward reference stack spointer
FSDEPTH      word        ( -- n) Return number of items on FS stack
>FS          word        ( addr --) Push addr onto FS stack
FS>          word        ( -- addr) Pop addr off FS stack
LABELS       create      Internal use by the assembler. Data space for labels
]LBL         word        ( n -- addr) Label array
NEWLABELS    word        ( -- ) Internal use.
$:           word        ( n -- ) Define a destination label called n
$            word        ( n -- 0)  Define a source label n
?LABEL       word        ( addr -- addr) Check addr is a label
RESOLVER     word        ( -- ) Resolves all label jumps on FS stack
+CODE        word        ( <name> ) Define a code word. Do NOT reset label stack
CODE         word        ( <name> ) Define a code word. Reset the label stack.
NEWLABELS    code word   ( -- ) Clear label array. Clear FS stack.
ENDCODE      word        ( -- ) End code word. Resolve all label jumps

Lines: 223
Bytes: 6237
--- End of file ---
```