Appendix D
Sample Applications

We have provided several complete applications written in JForth. They can be used either directly from JForth or they can be cloned and used as CLI commands. Some of them demonstrate how to parse the command line, how to handle input errors and print "usage" information. The file names all end with ".f" to distinguish the Forth source file from the cloned image file. These files can be found on the "JA:" logical volume.

Please see the chapter on Clone for instructions on how to clone these applications.

CR2LF.f - Carriage Return to Line Feed

Converts every Carriage Return in a file to a Line Feed. This is handy if you get a file from some unfortunate Macintosh user.

DIAL.f - Quick Dialer using a Modem

DIAL is a CLI based utility that will look up and dial telephone numbers through your modem. All you have to do is enter in a CLI:

```
DIAL FRED
```

then pick up the phone and wait for Fred to answer. DIAL will search a text file containing names and phone numbers. When it finds a match it opens the SER: device and send a Hayes compatible message to your modem to dial. DIAL was written as a shell command because a mouse driven interface would take longer to use then just dialing the phone directly.

To install DIAL you must:

1) Copy DIAL to your C: directory or other command directory.

2) Set your serial preferences to match your modem. Eg. 1200 baud, 8 data bits, etc.

3) Create a file called S:PHONELIST containing names and phone numbers. Names and phone numbers cannot have spaces inside them. For example:

```
JOE  1(707)555-1212                <-- GOOD
FRED FLINTSTONE 1 (707) 555-1234   <-- BAD
```

There are two ways to do this. One is using a text editor. The other is to use the ADD option of DIAL. For example:

```
DIAL ADD  WILMA  1(415)555-9876
```

The above will add WILMA's name to your phone list. If the phonelist file does not exist, DIAL will create it. To delete names, or to change names, just edit the file using a normal text editor like EMACS or Textra. You can optionally specify a filename other than S:PHONELIST. For example:

```
DIAL IRMA MYSTUFF:MYLIST
```

Since that would be more typing than just dialing it yourself, I recommend making an alias as follows:

```
ALIAS  DL DIAL [] MYSTUFF:MYLIST
```

Put that alias in your S:SHELL-STARTUP file.

Note that DIAL.f is shareware. If you bought JForth then ignore that. Consider yourself paid up. Please feel free to give a cloned version of DIAL to a friend. Just make sure you also give the Dial.README file which contains this documentation and information on the shareware aspects of

Dial.

Docu.f - Automatic "Documentation" Generator

This application can be used to automatically generate simple documentation from a file. It scans a file and prints any line that has a character in the first column, is not a comment, and has a stack diagram on it. Thus the following lines would be printed if they occurred in a file.

```
: ADD1  ( N -- N+1 , add one to a number )
and
VARIABLE NUM-FOO ( -- addr , number of foos)
and
: ADDEM  { aa bb -- aa+bb , add em up }
```

The word which should be cloned is:

```
: DOCU.FILE ( <filename> -- , document file )
```

DumpBrush.f - Dump a brush as JForth source code.

This application reads an IFF file containing a brush and outputs JForth compatible source code. Thus you can use DPaint to create images and use them in your programs as gadget images. Clone this program then enter:

```
DumpBrush >outfile brushfile
```

DumpIFF.f - Dump IFF file for analysis.

This scans an IFF file and prints out the contents of the chunks it encounters.

H2J.f - Convert a 'C' style ".h" file to a ".j" file.

This is the program we used to convert the Amiga DOS include files from 'C' code to JForth compatible code. See the chapter on Amiga Libraries and Structures for more information.

Print.f - Print a File

PRINT will output a file to a printer. It prints the file name, page number, and date on each page. It will also put line numbers on each line and skip over page perforations. PRINT uses the left and right margins set in preferences to tell whether a line will wrap around. Thus, lines that wrap will not mess up the page breaks. You can turn off the line numbers using the "-n" option. You can set the number of spaces for a tab using the "-t#" option. The default tab spacing is 8 spaces between tabs. To print a file called MYSOURCE without line numbers and with tabs set to 6 spaces, enter:

```
PRINT MYSOURCE -N -T6
```

If you want to send the output to a file instead of the printer, give that filename after the source file. For example:

```
PRINT MYSOURCE OUTFILE
```

Rude.f - Print Rude Message using an Alert

You can use this application to scare your friends or to control command files. RUDE will read a quoted string from the command line and display it in an Alert just like a GURU Meditation Error. The sight of these messages is enough to send a chill down the spine of most Amiga users.

Rude asks you to hit the left button for Yes and the right for No. A Yes will return 5 which can then be used to control the command file. Here is a sample Amiga DOS command sequence (not Forth).

```
RUDE "Should I install TextMangler 3.4?"
IF WARN
        RUN TextMangler
ENDIF
```

SortMerge.f - Merge Presorted Files

SORTMERGE will take two presorted files and merge the output into a third file in sorted order. This application is useful for adding new data to a large presorted file. It can be handy when used with DOCU.FILE to create quicky documentation of a large multifile project. Use the Amiga DOS SORT command to presort the input files.

```
SORTMERGE infile1 infile2 outfile
```

SayNumber.f - Recite a single-precision number in decimal

This JForth program parses the next word in the input stream and pronounces it in decimal-quantified form. For example, the "Say" command, as furnished by Commodore, would pronounce 123 as "one, two, three", whereas this program will say "one hundred and twenty-three".

```
saynum 123
saynum 1989
```

Terminal.f - Very Dumb Terminal Program

This very simple terminal program can be used as a starting point for a fancier terminal program.

Update.f - Copy newer files from one directory to another

This JForth program examines two directories, then copies everything from <srcdir> to <destdir> that meet the specified criteria. This is useful for maintaining backups of projects on floppies. UPDATE was written by Mike Haas for us at Delta Research as we worked on JForth on multiple Amigas. We could merge our work by updating to a central computer.

Usage:

Update SourceDir DestDir [-date|-size -list -noask -both]

```
-date = SourceDir/file is later than DestDir/file (default)
-size = SourceDir/file is different size than DestDir/file
-list = Don't really copy, just list the files that WOULD.
-noask= Don't ask the user to verify input
-both = Update only those files that exist in BOTH directories
-all  = Check all files in the directory and any subdirectories.
```

Only the first letter of the option is required. Here is an example that updates everything from a hard disk resident volume called MYWORK: to a floppy in DF1:.

```
UPDATE MYWORK: DF1: -N -A
```

WordCount.f - Count Words Lines and Chars

This is very similar to the UNIX wc command except it gives you all the information at once. The main word is called WC and is used like this:

```
WC filename
```

Appendix E
Style Guidelines

Naming Conventions

Forth allows ultimate flexibility in the naming of words. For this reason, it is very important that the programmer be consistent in their naming. Naming conventions can provide a road map to your code. If you ever have to go back and revisit code months, or years, after writing it, you will be thankful if you used systematic naming.

This section will provide some of the naming conventions used in JForth and offer suggestions for your own naming. Since JForth naming is the combination of naming conventions from 3 developers, plus the standard Forth names, you may notice deviation from these suggested standards in JForth. These suggestions will be more rigidly adhered to in the code I wrote since I often take my own advice. Here goes:

Use Prefixes to Mark Related Code

By a 2 or 3 letter prefix to all the words of a given system, you can clearly identify those words when they appear in other code. This also helps eliminate naming conflicts if the prefixes are unique. Examples in JForth are:

```
GRxxx - for GRaphics words.
WD_xxx - for members of the WinDow structure
DBxxx - for internal Debugger Words
$xxx - for string related words.
```

Use Signature Characters to indicate a word's function.

You can often place characters in the name that indicate what a given word does. Examples are:

4+ - adds 4

```
CONSOLE!  - set pointer in CONSOLE variables.
GR.COLOR!  - set graphics color (store in RastPort)
```

(xxx) - parentheses usually indicate an internal function that is called from a deferred word, or that is compiled for run time action. Sometimes <> and [] are used. Examples are:

```
(EMIT)  - simplest EMIT word.
(($"))  - run time action of ".
```

{ and } are handy for marking words that are balanced. An example is:

```
DEBUG{  : FOO DUP + ; }DEBUG
```

Use Separators that indicate the Type of Word.

Compound words can use a separator that give you some idea of what the word is. The separators I use are:

- "**hyphen**" for variables, objects, and other data structures that leave an **address** on the stack. These will generally be followed by a @ or !. Examples are:

```
HIGHLIGHT-INPUT    GR-CURWINDOW    MAX-INLINE DL-LINENUM
```

_ "**underscore**" for constants, values, or other words that leave their actual **value** on the stack. Examples are:

```
OFFSET_BEGINNING    KH_HISTORY_SIZE    MEMF_CLEAR
```

. "**dot**" for action words, verbs. Examples are:

```
GR.DRAW    DEBUG.START    PIC.LOAD
```

Use full names for obscure words, short names for common words.

Short names are good for common words because no one likes to type long names all the time. For rarely used words, however, you should use long names because you are less likely to remember what they do. There is also less likelihood of naming conflicts with long names. Remember when an application is cloned the names are removed and do not add size to the final image.

Writing Style

Here are some random thoughts on programming style.

Beware of compile time initialization.

If your program needs to open files, allocate memory, build jump tables or do anything with addresses, please put that code in a colon definition. And please do not call that word in the file itself. Doing so might seem like a handy little trick but it will bite you if you do a SAVE-FORTH or try to CLONE your program. Opening files, and allocating memory generate addresses that are only valid for that time. If you do a SAVE-FORTH or CLONE you may save an address that will not be valid the next time you use it.

Here is an example of some BAD code in a file:

```
VARIABLE MYFILE
VARIABLE MYMEM
FOPEN RAM:DATA MYFILE !  \ Bad! Bad programmer!
MEMF_CLEAR 2000 ALLOCBLOCK MYMEM !  \ Very Bad!
```

Here is an example of doing it right. Notice that we also check for error flags on initialization, and provide for automatic if we forget the code. Using these techniques will save you from many puzzling errors.

```
VARIABLE MYFILE
VARIABLE MYMEM
: DOITRIGHT  ( -- error? , "Good programmer! Here bisquit.")
   TRUE \ default error return
   " RAM:DATA" $FOPEN ?DUP  \ did it work
   IF
      MYFILE !
      MEMF_CLEAR 2000 ALLOCBLOCK ?DUP
      IF
         MYMEM !
         DROP FALSE  \ change error?
      THEN
   THEN
;
: CLEANUP ( -- )
   MYFILE FCLOSEVAR
   MYMEM FREEVAR
;
IF.FORGOTTEN CLEANUP
```

Write short words.

Chuck Moore first pointed out that short definitions give you the most flexibility. Nothing is worse than a word that does TOO much. If you want part , but not all, of what a word does you cannot use it. It is useless. But if a word does only part of what you need you can add the rest.

Place Stack Diagrams on Every Word.

and sometimes inside words. It is nearly impossible to read someone else's code if there are no stack diagrams. I have also seen people trying to write a word when they don't know what its stack diagram should be, another impossible task.

Indent Paired Words to the Same Level.

This really helps improve the readability of code and can prevent many common mistake. I have come to believe that IF, ELSE ,THEN, BEGIN, WHILE, REPEAT, UNTIL, CASE and ENDCASE should always be on their own line and cause a change in indentation level. Here is an example of the suggested indentation style.

```
: FOO ( a -- b , do something )
    DUP 21 <  ( -- a flag, generate flag on previous line
    IF
        0  ( -- a 0 )
        DO  I . CR
        LOOP ." All done!" CR
    ELSE  ( a -- , good place for stack diagram )
        100 >
        IF  ." A really big!" CR
        THEN
    THEN
;
```

Use >R R@ R> or Local Variables -

to avoid excessive stack dancing. Too much DUP SWAP DROP will ROT your brain.

Write your Code Backwards using "Top Down" programming.

Start by writing the top word in your application using an English like set of words. Once this makes sense, write the next lower level, and so on. Forth is enough like English that your "pseudo-code" can end up being legal Forth by the time you are done. Forth is typically thought of as a "bottom up" language because it is so easy to hack in. This is fine for pure experimentation but big projects demand the discipline of "top down" coding.

Initialization, Action, and Termination

Divide each module of your program into three parts: Initialization, Action, and Termination. This will greatly simplify debugging because you will be able to completely initialize the system and then examine it before taking any action. It also helps to organize your code. Initialization typically consists of things like opening files, allocating memory, opening windows, or creating gadgets.

Transportability Techniques

by Brian Donovan

This appendix describes some techniques to use for writing transportable high level Forth code.

The CELL concept.

In Forth programs, you often want to skip over a single precision unit of data, or several single

precision units of data, called "cell"s in Forth. Until recently, most Forth programmers were explicitly using the numeric count needed on the machine they happened to be using. This meant that on a 16 bit 8086 Forth they would use a 2+, on a 68000 32 bit Forth, like JForth, they would use 4+ , and on a NOVIX Forth CPU with word addressing, they would use 1+ . For subtraction, multiplication and division by the single precision unit addressing size, the same situation would arise, thus, all programs written that way would be totally untransportable. A very simple and low overhead solution to this problem is to use CELL+ CELLS CELL/ CELL- and CELL instead of 1+ 2+ 4* etc.. whenever the program is working with cells. We have painlessly used the same source for 16 bit and 32 bit computers of many types, and have found that programs written with CELL are much less obscure to read.

In JForth: CELL = 4

One way to automatically assign CELL size in your programs is as follows:

```
SP@ SP@ - ABS CONSTANT CELL
: CELL+  CELL + ;
: CELL-  CELL - ;
: CELLS  CELL * ;
: CELL/  CELL / ;
```

With conditional compilation, you can pick out the optimal speed words for functions like CELLS and CELL/ . In JForth, all these words are machine coded.

INLINE concept.

In more advanced Forth programs, programmers often want to create new inline data words (like ."). With the advent of segment threaded Forth, relative address Forths (like JForth) , and other new ways of implementing Forth, The top return stack item does not necessarily point data following the called word (inline data). We use a slow version of LIT to illustrate some of the problems.

This is slow LIT the way it would be in absolute addressing, address threaded Forth (the good old standard unfancy Forth on a Z80 for example.)

```
: SLOW-LIT  ( --- N )  ( CELL --INLINE-- )
    R@ @ R> CELL+ >R ;
```

SLOW-LIT on a relative addressing Forth (like JForth) :

```
: SLOW-LIT   R@ >ABS @ R> CELL+ >R ;
```

SLOW-LIT on some segment threaded 32 BIT, 8086 Forths:

```
: SLOW-LIT-SEG   R@  SEG+OFF>ADDR  @
    R> SEG+OFF>ADDR CELL+ ADDR>SEG+OFF >R ;
```

...pretty bad ? You would have to rewrite any section that uses inline data, when you changed Forths. You can avoid this problem by using the INLINE words: INLINE+ INLINE@ >INLINE and INLINE> (and adding them to other systems, as appropriate.) you can write fully transportable inline code:

```
: SLOW-LIT  ( --- N )  INLINE@ @ CELL INLINE+ ;
```

This definition will work on ALL of the Forths I have over encountered, as long as you add the appropriate 4 inline definitions to the dictionary first. by the way, SLOW-LIT is used as follows:

```
: SLOW-LITERAL   ( N --- )
    COMPILE SLOW-LIT , ; IMMEDIATE
: EXAMPLE  [ 5 ] SLOW-LITERAL ;
```

You can always add these definitions to a system that doesn't have them. The definitions are very implementation dependent. For an old fig Forth system they would be the simplest:

```
: INLINE+  ( N --- )  ( ADDR --R-- ADDR+N )
    COMPILE R> COMPILE + COMPILE >R ; IMMEDIATE
```

```
: INLINE>  COMPILE R> ; IMMEDIATE
: >INLINE  COMPILE >R ; IMMEDIATE
: INLINE@  COMPILE R  ; IMMEDIATE
```

In JForth the definitions are slightly more complicated, and on some of the 32 bit segment threaded Forths, they are fairly ugly. But once written, all the rest of your code flies!

You may be confused by a word in JForth, INLINE , which has little to do with the inline words above. INLINE sets a flag in a words header, telling the compiler that this word must be compiled inline, it cannot be called (for instance >R) .

Don't Use JForth Internal Words

Avoid using JForth internal words unless you can reconstruct them from scratch.

Understand Unique Features Before Using Them

Unfortunately, you also have to avoid using some of the nice JForth utilities unless you can reconstruct them as well. At least we will keep the JForth utilities around. All of the JForth unique programs are copyrighted, but many are available for distribution without charge. If the file does not specifically state, that it may be freely distributed, than you must get our permission to distribute it elsewhere. We encourage you to take the CELL and INLINE concepts and programs anywhere you want. Many of the JForth utilities are available on other Forths as well, and we made every attempt at compatibility with other Forths where this is appropriate. Some programs that you need permission to distribute are: ODE ASM DISM MULTISTANDARD LOCALS

Delta Research is actively helping to establish better standards, particularly for high-end Forth systems.

Conditional Compilation.

Use the conditional compilation words, .IF .ELSE .THEN .NEED INCLUDE? , to adapt to different systems automatically. Usually the differences between systems are small enough to allow one source to deal with all target systems. Major sections that are different between targets, can be put into separate files, and conditionally loaded with INCLUDE? . Unfortunately, The Forth community has not standardized the names for the conditional compilation words. JForth uses the same names as LMI Forth. (The '83 standard include a SUGESTED set of names that we think are unacceptable: IFTRUE OTHERWISE IFEND). The words we have used are easy to remember, since they work like the familiar IF ELSE THEN words all Forth programmers know so well. The "." sets them apart from the ordinary conditionals as well. We would have preferred [IF] [ELSE] [THEN] to emphasize the fact that these words work in a different state, but we felt it was better to use an existing acceptable set of words.

Multistandards

If you need to compile code that conforms to the FIG, Forth'79, or Forth'83 standard, then INCLUDE the file JU:MULTISTANDARDS.

Apendix F
Words by Function

Address Conversion

```
>ABS >REL IF>ABS IF>REL CALL>ABS
```

AMIGA Library Access

```
ARGS CALL CALL>ABS CALLVOID CALLVOID>ABS CLOSEALLLIBS CLOSELIB
DCALL LIB? LIBRARY LIBVERSION OPENLIB
```

Arithmetic

```
* */ */MOD + +! +- - / /MOD 1+ 1- 2* 2+ 2- 2/ 4* 4+ 4- 4/ <<
@BITS ABS ASHIFT BIT-SET? BYTE-SWAP CLR-BIT COMP D+ D- D2* D2/
DABS DNEGATE DU2* DU2/ EVEN-UP FIG-NOT M* M/ M/MOD MAX MIN MOD
NEGATE SQRT U* U/ U2* U2/ W/
```

Conditional Compiling

```
.ELSE .IF .NEED .THEN EXISTS? INCLUDE?
```

Conditional Operators

```
0< 0= 0> < <= = > >= D< D= DU< FALSE NOT TRUE U< U> WITHIN?
```

'C' Structures from JU:CSTRUCT

```
..! ..@ :STRUCT ;STRUCT APTR ARRAYOF BYTE BYTES DST GETMODULE
LONG OB.STATS? S@ S! SHORT SIZEOF() STRUCT UBYTE ULONG
UNION{ USHORT }UNION }UNION{
```

Compiler Support

```
!CSP >INLINE >PARENT ?COMP ?CSP ?EXEC ?PAIRS BOTH BSR-CODE
CALLADR, CARRAY CFA, COMPILE COMPILING? CREATECHAR DLIT DLITERAL
DO-DOES-SIZE DOES> IMMEDIATE INLINE INLINEOK? INTERPRETING? JSR-
CODE LIT LITERAL LONGCFA, RECOGNIZE RTS-CODE SMUDGE STATE
UNSMUDGE [ [COMPILE] ]
```

Debugging Aids

```
&CHARS .FROM .FROM-OR.  .VOC 1010-EMULATE 1111-EMULATE }DEBUG
BEST-GUESS DEBUG DEBUG{ DEBUG.ON DEF DO-TRACE DST DUMP FILE?
FIND-DATA FIND-WDATA FREE-TRAP FROM-ADR FROM-VOC FROM-WORD GET-
TCB GET-TRAP H.  IF-DEBUG IF-TESTING MAP MEMCELLS? NO-EMULATION
NOTRAPS STACK-HOLD STACK.CHECK STACK.MARK START.UNRAVEL.QUIT
STOP.UNRAVEL.QUIT SYSUSER# TIB.DUMP TRAPS UNRAVEL VALID-NAME?
WORD.DUMP
```

Dictionary Address Conversion

```
'>BODY '>NAME >BODY >LINK >NAME BODY> LINK> N>LINK NAME>
```

Dictionary Management

```
' , ALIGN ALLOT C, CRID.  DP DPLIMIT HERE ID.  IMMEDIATE?
INITVOCS MAX-INLINE PAD ROMABLE ROOT W, ['] 
```

DOS Commands from JU:DOSCOMMANDS

```
+DOS >DOS ASSIGN AVAIL CD COPY DATE DELETE DIR DOS DOS0
DOSCOMMAND DOSSTRING INFO LIST NEWCLI PATH RENAME RUN STATUS
```

Disassembler from JU:DISM

```
ADISM DEF DISM DISM-CYCLES DISM-NAMES DISM-WORD? INIT-DISM RISM
SELECT-DISM-DEFAULTS SEE
```

Defining Words

```
#VOCS : :CREATE ; ARRAY CONSTANT CREATE REVERTVOC USER VARIABLE
VALUE { }
```

Error Reporting

```
.ERR ?ABORT" ?ERROR ABORT ABORT" ERROR MSG QUIT WARNING WARNING"
```

Forth Screen Support

```
BLK BLKERR LIST LOAD R# SCRED
```

Flow of Control

```
+LOOP -DO -LOOP 0BRANCH ?EXIT ?LEAVE ?RETURN ?STAY AGAIN BACK
BEGIN BRANCH CASE CASE_FLAG CONDITION DO DO-LOOP-NEST ELSE END
END-SELECT ENDCASE ENDCOND ENDOF EXIT FORWARD I IF IF-NOT IK J
LEAVE LOOP LOOP-BACK LOOP-FORWARD NOT0BRANCH OF REPEAT RETURN
SELECT THEN TIMES UNTIL UNTIL-NOT WHILE WHILE-NOT
```

FILE I/O

```
$FOPEN +DOS @&CLOSEFILES ACCESS_READ ACCESS_WRITE BLOCK
CLOSEFILES EOF EOL F@, F@,? FCLOSE FCLOSEATBYE FCLOSEVAR FEMIT
FERROR FILEMODE FKEY FDUMP FOPEN FREAD FREAD? FSEEK FSEEK? FTYPE
FWRITE MARKFCLOSE MODE_NEWFILE MODE_OLDFILE NEW OFFSET_BEGINNING
OFFSET_CURRENT OFFSET_END OLD TEMPBUFF TEMPF, TEMPFILE TYPEFILE
UNMARKFCLOSE
```

Floating Point

```
F.  F* FLOAT FNUMBER FNUMBER? FSIN see chapter on Floating Point
```

Forth System

```
BYE COLD INTERPRET QUIT
```

Host Portability

```
'C CELL CELL* CELL+ CELL- CELL/ CELLS CFA->PFA HO.FIND.CFA
HO.FIND.PFA HOST" HO_MAX_INT HO_MIN_INT INLINE+ INLINE> INLINE@
PFA->NFA PICK79 PICK83 REL->USE USE->REL
```

INPUT

```
#TIB >IN ?PAUSE ?TERMINAL BSIN EXPECT FILEWORD HISTORY KEY PARSE
PARSE-WORD QUERY SKIP-WORD? SOURCE SPAN TIB TIB0 TIBEND UNWORD
WORD Y/N
```

I/O General

```
#CHARS ?LETTER ?VISIBLE ASCII BL CONSOLE EMPTYCHAR HICASE
LINELIMIT LOCASE LPLACE LWORD NOCASE OFFSET PLACE
```

Logging to a File from JU:LOGTO

```
$LOGTO LOGEND LOGGED? LOGSTART LOGSTOP LOGTO
```

Logical

```
!BITS +SHIFT -SHIFT AND OR SET-BIT SHIFT WORD-SWAP XOR |
```

Memory Access

```
! +! -> 2! 2@ ? @ ABS! ABS@ ABSC! ABSC@ ABSW! ABSW@ C! C@ CMOVE
CMOVE> D! D@ ERASE FILL MOVE ODD! ODD@ ODDD! ODDD@ ODDW! OFF ON
SPARE TOGGLE W! W@ WMOVE X! X@
```

Memory Allocation

```
+STACK -STACK @&FREEBLOCKS ALLOCBLOCK ALLOCBLOCK? ALLOCSTRUCT
FREEATBYE FREEBLOCK FREEBLOCKS FREEBYTE FREEBYTEA FREECELL
FREEMEM MARKFREEBLOCK MEMF_CHIP MEMF_CLEAR MEMF_FAST MEMF_LARGEST
MEMF_PUBLIC NEXTMEM NEXTMEMA POP PUSH SIZEMEM UNMARKFREEBLOCK
XALLOCBLK XFREEBLK
```

Numeric Conversion

```
# #> #DIGS #DIGITS #S $ (NUMBER) .  .HEX .HX .R <# BASE BINARY
COMMAS CONVERT D.  D.R DECIMAL DIGIT DIGS/, DPL HEX HLD HOLD
N>TEXT NO-COMMAS NUMBER NUMBER? SIGN U.
```

Output

```
+OUT ." .S <FASTEMIT> <FASTKEY> <FLUSHEMIT> >NEWLINE ?MORE ?WRAP
BELL BSOUT CLRCHAR CLS CR CR? EMIT EMIT-TO-COLUMN FAST FLUSHEMIT
MAX-TYPE OUT OUTBUF OUTPUT-CASE SLOW SPACE SPACES TAB TAB-WIDTH
TYPE TYPE-HERE WTYPE
```

Random Numbers from JU:RANDOM

```
CHOOSE RAND-SEED RANDOM WCHOOSE
```

Return Stack

```
.RS .RSTACK 0RP >R DUP>R MYRP R R0 R> R@ RDEPTH RDROP RP! RP@
RP+! RPICK SET-RP X>R XR> XRDROP
```

Saving and Loading

```
#K #RELOCS #U .IMAGE .RELOCS ?FORGOTTEN ABSRELOCS CLONE FBLK
FILEHEADERS HEADTAIL IFOPTIMIZE INCLUDE INCLUDE? LOAD-FILE
PUSHRELOC REDEF? RELOCSADR SAVE-FORTH SAVE-IMAGE SCR-FILE
SCREDING TURNKEY TURNKEYING?
```

Stack Manipulation

```
-2SORT -DUP -ROT 0SP 2DROP 2DUP 2OVER 2SORT 2SWAP ?DUP ?STACK CSP
DDROP DDUP DEPTH DROP DSWAP DUP MYDSP MYTOS NIP OVER PICK ROT S0
SET-SP SP! SP@ SWAP TUCK XDROP XDUP XPICK
```

Strings

```
" $" $, $- $.  $= $ACCUM $APPEND $CLR $CONCAT $LEN $MOVE $N>S
$SIZE $TYPE $VARIABLE -TRAILING /STRING 0" COMPARE COUNT
MAKEUCASE MATCH? MCASE-SENSITIVE SCAN SKIP SKIP-WORD SKIPWORD
TEXT=? UPPER UPPERC@
```

Timers

```
BENCH BENCH.WITH MEASURE MSEC PROFILE
```

Unused Words Analysis from JU:UNUSED

```
CLEAR.MARKS MARK.UNUSED MARK.USED START.MARKING.WORDS
STOP.MARKING.WORDS UNUSED.WORDS USED.WORDS USED_BIT
```

User Stack

```
>US 'USP> INIT-USP US-DEPTH US-PICK US> US@ USP! USP@ UP0 USP
```

Vectored Execution

```
>IS @EXECUTE COLDEXEC DEFER DEFER-EXECUTE EXECUTE GLOBAL-DEFER IS
ISDEFUSER? WHAT'S
```

Virtual File I/O

```
BUFFERADR CLOSEFVREAD CLOSEFVWRITE F, FFLUSH? LINESFILLV OPENFV
READLINE VIRTBUFFSIZE
```

Vocabulary Management

```
ANEW CONTEXT CURRENT FENCE FIND FORGET FREEZE LATEST MAXVOCS
SCAN-ALL-VOCS SCAN-VOC SCAN-WORDS VALLOT VLATEST>VLINK VLINK>'
VLINK>VLATEST VLIST VOC-LINK VOCABULARY WHEN-SCANNED WHEN-VOC-
SCANNED WORDS WORDS-LIKE
```

Word Sizing

```
B->S S->D W->S
```

Appendix G
Incompatibilities

Between V2.0 and V3.0

Error Handling in IFF and Picture words.

Numerous changes have been made in the way that these words handle errors. IFF.WRITE? has a new stack diagram. These changes are described in detail in a special section on incompatibilities in chapter 21.

Local Variables

The local variable used in V2.0 have been completely replaced. We tried to make the new ones compatible with the old ones but there are some differences. The new locals are kept on the return stack instead of the data stack. The only code that should be affected was code that illegally used internal aspects of the old locals or that illegally relied on the parameters being on the data stack.

Between V1.2 and V2.0

The changes between JForth 1.2 and 2.0 mainly involve the addition of new code. We tried to avoid making any changes that would cause code written using version 1.2 to fail under 2.0. In some cases, however, we felt that the definition of a word in 1.2 was faulty and was worth correcting. Luckily the changes are minor.

>BODY and BODY>

In JForth 1.2, these were defined as NOOP. This was because JForth is subroutine threaded does not have a traditional CFA and PFA. We felt, however, that >BODY was mainly used to get the address of data in a CREATE DOES> word. For JForth, this meant we had to change the definition from NOOP to:

```
: >BODY  ( cfa -- body ) DO-DOES-SIZE + ;
```

Please correct for this in your code if you use >BODY or BODY>.

CONSOLE

This variable contained the address of the file used for I/O by JForth V1.2. Since I/O is a two way street, Input AND Output, we actually need **two** separate variables. This is required to make I/O redirection work properly in Cloned programs. We, therefore, split CONSOLE into CONSOLEIN and CONSOLEOUT. The single word CONSOLE! was added that will set both of these variables. We also added CONSOLE@ for symmetry. CONSOLE@ fetches the value of CONSOLEOUT.

In JForth V1.2:

```
CONSOLE   !
```

In JForth V2.0:

```
CONSOLE!  ( one word )
```

Totally New Floating Point

The old floating point system was not worth keeping after we got a better version from Dave Sirag. We recommend that you switch to this new system. If you absolutely need the old system, take it off of

the old disks.

Bugs Fixed

Since bugs were fixed in these words, they will behave differently. You will only have to change your code if you had made a patch for these words. There were other bugs fixed but they should not affect stack diagrams.

```
ROLL  ( was totally messed up, now OK )
U/    ( gave wrong answers sometimes for big numbers )
D!    ( used to swap halves incorrectly, not anymore )
```

Signed AND Unsigned Structure Members

JForth 1.2 treated all structure members as unsigned. The Amiga makes a distinction between signed and unsigned so JForth 2.0 does so as well. The difference is that if you FETCH an 8 or 16 bit signed member, it will be sign extended if it is declared as SHORT or BYTE. Members declared as USHORT or UBYTE will not be sign extended. If you store a -3 in a signed BYTE using ..! then get it back using ..! , you will have -3 on the stack. In an UNSIGNED BYTE you would end up with hex FD.

We have given you an option here. If you do not want automatic sign extension in your program, compile your code with the variable SIGNED-MEMBERS set to FALSE. The default is TRUE.

Appendix G
Incompatibilities

Between V2.0 and V3.0

Error Handling in IFF and Picture words.

Numerous changes have been made in the way that these words handle errors. IFF.WRITE? has a new stack diagram. These changes are described in detail in a special section on incompatibilities in chapter 21.

Local Variables

The local variable used in V2.0 have been completely replaced. We tried to make the new ones compatible with the old ones but there are some differences. The new locals are kept on the return stack instead of the data stack. The only code that should be affected was code that illegally used internal aspects of the old locals or that illegally relied on the parameters being on the data stack.

Between V1.2 and V2.0

The changes between JForth 1.2 and 2.0 mainly involve the addition of new code. We tried to avoid making any changes that would cause code written using version 1.2 to fail under 2.0. In some cases, however, we felt that the definition of a word in 1.2 was faulty and was worth correcting. Luckily the changes are minor.

>BODY and BODY>

In JForth 1.2, these were defined as NOOP. This was because JForth is subroutine threaded does not have a traditional CFA and PFA. We felt, however, that >BODY was mainly used to get the address of data in a CREATE DOES> word. For JForth, this meant we had to change the definition from NOOP to:

```
: >BODY  ( cfa -- body ) DO-DOES-SIZE + ;
```

Please correct for this in your code if you use >BODY or BODY>.

CONSOLE

This variable contained the address of the file used for I/O by JForth V1.2. Since I/O is a two way street, Input AND Output, we actually need **two** separate variables. This is required to make I/O redirection work properly in Cloned programs. We, therefore, split CONSOLE into CONSOLEIN and CONSOLEOUT. The single word CONSOLE! was added that will set both of these variables. We also added CONSOLE@ for symmetry. CONSOLE@ fetches the value of CONSOLEOUT.

In JForth V1.2:

```
CONSOLE   !
```

In JForth V2.0:

```
CONSOLE!  ( one word )
```

Totally New Floating Point

The old floating point system was not worth keeping after we got a better version from Dave Sirag. We recommend that you switch to this new system. If you absolutely need the old system, take it off of the old disks.

Bugs Fixed

Since bugs were fixed in these words, they will behave differently. You will only have to change your code if you had made a patch for these words. There were other bugs fixed but they should not affect stack diagrams.

```
ROLL  ( was totally messed up, now OK )
U/    ( gave wrong answers sometimes for big numbers )
D!    ( used to swap halves incorrectly, not anymore )
```

Signed AND Unsigned Structure Members

JForth 1.2 treated all structure members as unsigned. The Amiga makes a distinction between signed and unsigned so JForth 2.0 does so as well. The difference is that if you FETCH an 8 or 16 bit signed member, it will be sign extended if it is declared as SHORT or BYTE. Members declared as USHORT or UBYTE will not be sign extended. If you store a -3 in a signed BYTE using ..! then get it back using ..! , you will have -3 on the stack. In an UNSIGNED BYTE you would end up with hex FD.

We have given you an option here. If you do not want automatic sign extension in your program, compile your code with the variable SIGNED-MEMBERS set to FALSE. The default is TRUE.

Appendix H
Products Written in JForth

This chapter describes commercial products that have been written in JForth. This is a partial list and contains only the ones we know about. If you have any JForth programs that you are selling, send us information and hopefully a sample copy, and we will, if possible, include a description for free in our next manual. If you read about something here that you like, please help your fellow JForth programmers by buying it.

B.A.D. from MV Micros

This "best selling utility" optimizes disk access time up to 500% by analyzing and restructuring any Amiga DOS disk. B.A.D. can reduce "disk gronking", speedup window opening under Workbench, and speed up DIR and other disk operations. B.A.D. also includes a disk structure test to detect errors. Works on both floppy and hard disks. B.A.D. is available from many mail order outlets. It is distributed by Centaur Software, Inc.

My Diary from MV Micros

My Diary is a full featured Personal Information Manager (PIM). My Diary provides a pop-up notepad for managing names, phone numbers, addresses, appointments, "to do" lists, due dates, diaries, birthdays, etc. Notes are date stamped and can be attached to the calendar. Notes can be searched for keywords. Timers can be used to provide *reminders*. Information can be imported from and exported to hand held electronic organizers like the Sharp Wizard. ARexx support and Auto Dialer also provided.

HMSL, the Hierarchical Music Specification Language

HMSL is a programming language for experimental music composition and performance. It is an extension to JForth that adds MIDI capability, local sound support, and an extensive set of object-oriented ODE classes for organizing, manipulating and playing musical data. HMSL also includes a graphics toolbox for building interactive screens, a score entry dialect, and a simple sequencer. HMSL was developed by Phil Burk, Larry Polansky, and David Rosenboom. For information contact:

```
Frog Peak Music
P.O. Box 151051
San Rafael, CA
94915-1051
(415) 461-1442
```

Copyist Companion by Nick Didkovsky

Copyist Companion converts Deluxe Music Construction Set (DMCS) score files to Dr. T's The Copyist files. All the features of the DMCS score are meticulously converted, including time and key signature changes, clefs, ties, beaming, stem direction, slurs, dots, chords, triplets, quintuplets, accidentals, dynamics, etc. The result is a beautifully laid out Copyist score, ready for professional printing. Copyist Companion was written in JForth (of course) and runs on any Amiga. The great quantity of reverse engineering required to parse the DMCS file format made JForth and ODE the only programming environment I would have considered using for this project. You can get more info

about Copyist Companion from:

```
Nick Didkovsky
171 East 99th Street #20
New York, NY 10029
(212) 369-1733
email didkovsk@dorsai.com
```

Or contact Dr. T's Music Software, who are distributing Copyist Companion.

Nick is also developing a DMCS to MIDIfile conversion utility. Call or write for details on its progress.

Doctor Nerve with Nick Didkovsky

Doctor Nerve's latest two CD's are "Beta 14 ok" and "Did Sprinting Die?". Both projects include compositions generated by HMSL (thus the JForth connection). These compositions are performed by the band on "Beta" and directly by Amiga/HMSL on "Sprinting". Both CD's also include non-HMSL-composed material. See if you can tell the difference!

Doctor Nerve LP's and CD's can be ordered from:

```
Wayside Music
PO Box 6517
Wheaton, MD 20906.
```

Ask for their catalog. Or contact Nick Didkovsky as above.

IntuiEZ by Curtis Stanton

IntuiEZ is a shareware program that allows the user to interactively design screens and windows with gadgets using the mouse. You can then output JForth compatible source code to a file. IntuEZ also has the ability to capture the Intuition structure from other applications screens and window for study or modification. This can create gadgets with the version 2.0 style 3D gadgets but it.works with Amiga DOS 1.3. At press time it was not known how this program would be distributed so call Delta Research for the latest information, or look for it on a BBS.

XL by Martin Kees

XL is a cel animation program by the author of JForth's ANIM support routines. It uses an *onion-skin* display that lets you see several cels at once. It also features a very powerful bezier curve editor. A demonstration version of XL can be found on Fred Fish disk #516 which, by the way, is entirely filled with programs written in JForth. On the same disk check out Enigma, a beautiful puzzle that models a programmable machine.

JGoodies_1 from various.

Check out Fred Fish disk #239. It is also all JForth code. Includes a fast Mandelbrot generator, audio tools, FFT code, HeadClean, and other goodies.