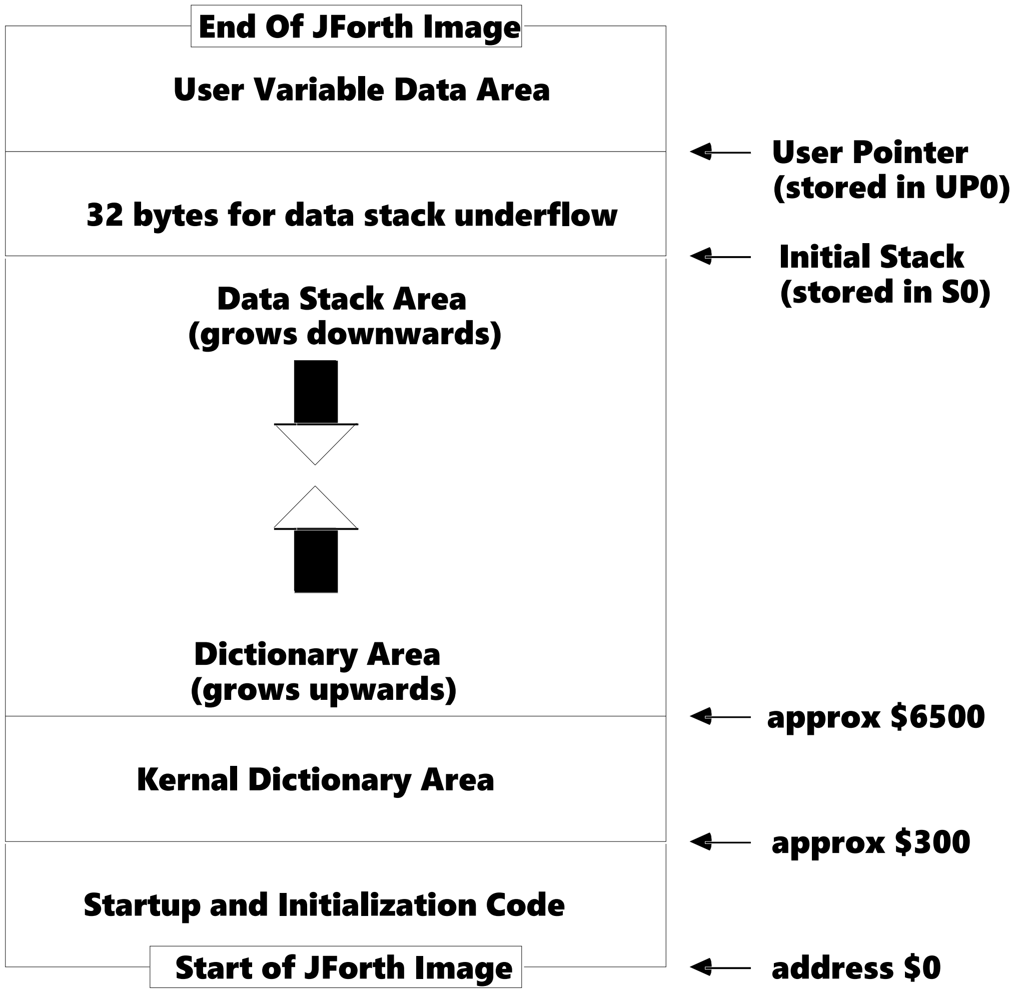
Chapter 12
System Internals

The following memory map shows the layout of the JForth image in memory.


End Of JForth Image
User Variable Data Area
User Pointer (stored in UP0)
32 bytes for data stack underflow
Initial Stack (stored in S0)
Data Stack Area (grows downwards)
Dictionary Area (grows upwards)
approx $6500
Kernal Dictionary Area
approx $300
Startup and Initialization Code
Start of JForth Image
address $0

JForth is stored on a disk device in Amiga Binary Format, and when loaded into ram and executed, occupies one contiguous block of memory (with a few specific exeptions, as explained later).

As the image may exist in either position-independent or relocatable form, it may reside at any location in either Chip or Fast ram (uses FAST, if available). For this reason, all addresses mentioned here (and indeed, within JForth itself) are relative to the base of the JForth image, with location 0 being the first (bottommost) byte of the executable program.

USER Variable Data Area

This area holds the actual memory used for storage of USER variables, while the names and executable code used to generate the address of this location reside in the normal dictionary space.

In future versions of JForth, which may support a 'Forth multitasking' environment, each Forth task spawned will receive a local copy of this area.   In single-task versions (3.0 and earlier), there is functionally no difference from global VARIABLEs, in fact, the CLONE program will convert USER variables into global VARIABLEs in the standalone image.

The size of the USER area determines the maximum number of USER variables which may be defined; each USER variable taking up 4 bytes. Note that this area may be increased by altering the value stored in #U and subsequently saving the image (via SAVE-FORTH). The saved image, when booted, will have the larger USER area. Once increased, the USER area cannot be made smaller for a given image.

The MAP command will display the number of USER variables defined, as well as the number yet available in the current image

Data Stack Area

Immediately below the USER area is a 32 byte 'buffer-zone', protecting the USER area from programs which underflow the stack by as much as 8 items. No protection is provided for more errant programs.

The Data Stack is assigned just below this zone, starting at the highest location and growing downward as numbers are 'pushed' on to it.

Note that this area is not related to the Return (or 'program') Stack which will be discussed later. This area is solely for the manipulation of data items by Forth operatives such as +, -, etc.

As data is placed on the Data Stack, the Data Stack Pointer (register A6) is decremented, resulting in a data area which grows downward toward the Dictionary Area. Consequently, items removed or consumed from the Data Stack cause the Data Stack Pointer to approach its original, higher value. Since debugged programs, over the course of running, give back any Data Stack consumed, the normal transient usage of this area does not interfere with dictionary expansion. This is insured by maintaining sufficient Dictionary Area available.

Extensible Dictionary Area

As the JForth vocabulary is extended, new definitions are added serially, consuming the memory immediately above the preceeding one. The act of compilation, then, causes the Dictionary Area to grow upward toward the Data Stack, resulting in an overall lowering of the available Dictionary Space.

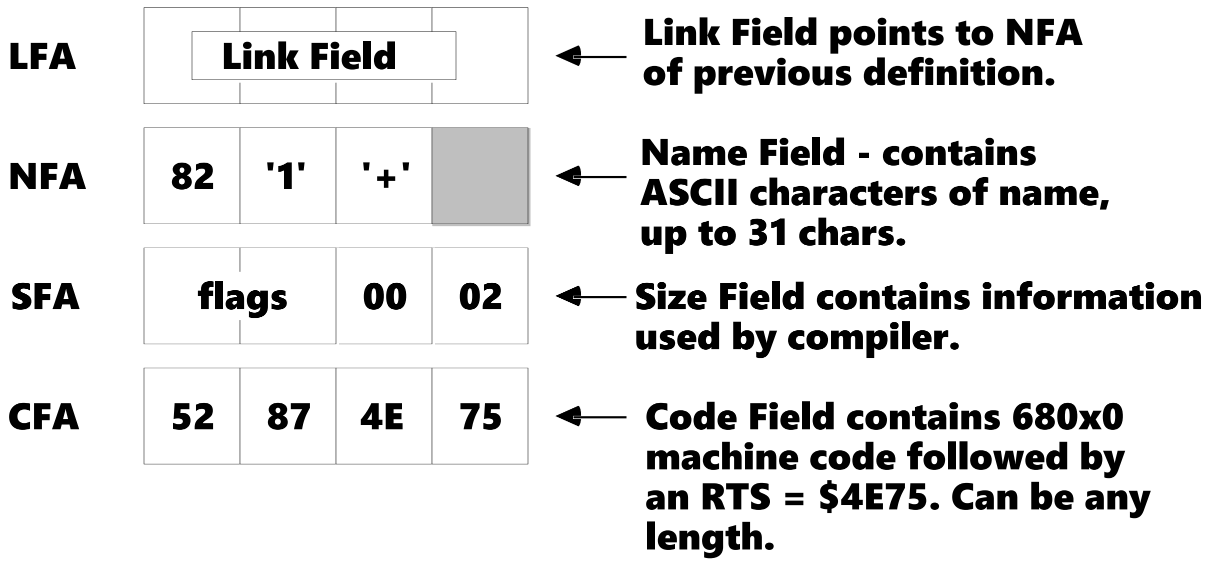
As mentioned, the amount of memory needed for the transient Data Stack is relatively small (less than 100 bytes, typically), so Data Stack utilization is not usually a consideration.   However, the process of compiling causes the Dictionary to grow toward the Data Stack by an amount directly proportional to the size of the compiled program; without sufficient Dictionary Area available, the Dictionary and Data Stack would meet with undesirable effects.

JForth prevents this by monitoring the distance between the Data Stack and the Dictionary very closely during compiles. The function ?STACK is used for this purpose, aborting the compile if this area becomes less than 256 bytes. Note that you may call ?STACK anytime within your own programs.

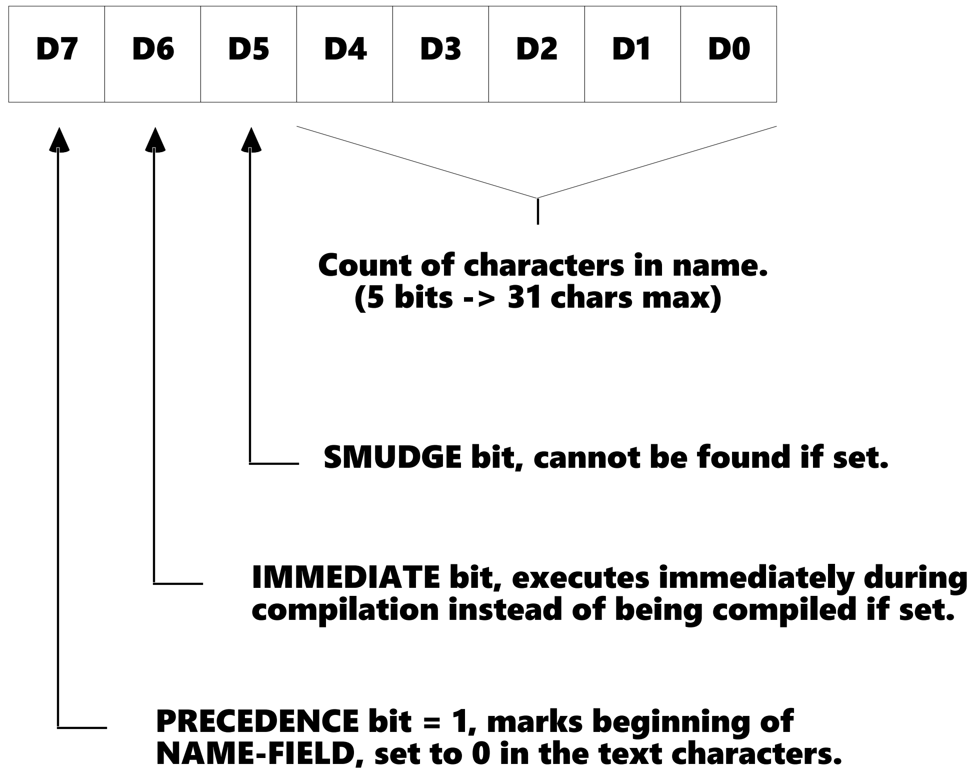
Note that the Extensible Dictionary Area may be increased by altering the value stored in #K and subsequently saving the image (via SAVE-FORTH). The saved image, when booted, will have the desired larger Dictionary area.

Each definition added to the Dictionary consumes Dictionary Space; how much is determined by the function type and size. However, regardless of the type and size, each function generates a Dictionary Header made up of a Link Field, Name Field, Size Field and Code Field. The following diagram show the layout of a dictionary header in memory, in order of increasing address:

Memory Layout of JForth Dictionary Header


LFA
Link Field
Link Field points to NFA of previous definition.
NFA
82
'1'
'+'
Name Field - contains ASCII characters of name, up to 31 chars.
SFA
flags
00
02
Size Field contains information used by compiler.
CFA
52
87
4E
75
Code Field contains 680x0 machine code followed by an RTS = $4E75. Can be any length.

1. The LINK-FIELD ... 32 bits holding the address (JForth relative) of the preceeding definition's NAME-FIELD. The LINK-FIELD address may be derived from the CODE-FIELD address (see below) with >LINK .
2. The NAME-FIELD ... contains the text of the defined name; the first byte of this field is used as an 'attribute' byte as follows:


D7
D6
D5
D4
D3
D2
D1
D0
Count of characters in name. (5 bits -> 31 chars max)
SMUDGE bit, cannot be found if set.
IMMEDIATE bit, executes immediately during compilation instead of being compiled if set.
PRECEDENCE bit = 1, marks beginning of NAME-FIELD, set to 0 in the text characters.

The text of the name follows this byte; the text may be followed by a 'filler' byte to insure that the next field is word-aligned. It is the address of the 'attribute' byte that is returned by >NAME , calculated from the CODE-FIELD address (see below).

3. The SIZE-FIELD ... The JForth compiler examines this field to find out how the word is to be compiled (stored in bits 30 & 31) and what the size of the CODE-FIELD is, minus the RTS (stored in the lower word, bits 0-15).

```
D31  D30 === Controls Compilation As Follows:
---  ---

 0    0 .... will be CALLED if size > MAX-INLINE ,
                  otherwise INLINE, referred to as "BOTH".
 0    1 .... must be CALLED.

 1    0 .... must be compiled INLINE.

(NOTE: bits 16-29 are reserved for future use).
```

4. The CODE-FIELD ... this field contains the executable 68000 assembly language and completely determines the runtime behavior of the word. It is this address that is returned by ' (tick). The CFA of a word is its *code field address.*

5. The PARAMETER-FIELD ... (optional) In JForth, this refers to the data portion of a CREATE DOES> word ONLY. The word >BODY will convert a CFA to a PFA of such a word. The PFA of a word is its *parameter field address.*

Kernel Dictionary Area

This area is functionally identical to the Extensible Dictionary area in that it is comprised of function definitions which follow the Dictionary Header format described above.

One difference, however is that the source code for this area is not provided in the JForth package, and therefore cannot be recompiled by the programmer (the Kernal is generated by a conventional 68000 Assembler).

The Kernel Dictionary Area is, however, supplied as an assembled package and may be used to regenerate the JForth system for various special cases...

1. The programmer has altered the JForth system files (jf:).

2. Regenerate with a different MAX-INLINE (JForth is normally generated with a MAX-INLINE value of 8).

3. Generate an absolute-minimum system for TURNKEYING (by booting COM:JKERNAL and stating TURNKEYING ON before regenerating).

4. Any other reason the programmer may think of.

For more information on regenerating or customizing the JForth system, refer to the section "How to Generate a New JForth System", later in this chapter.

The lastcompiled word in this area has, for historical reasons, been named TASK .

Other memory Allocation / Utilization...

In the process of using JForth, certain functions will allocate memory for systems use, and do so quite transparently to the user.

Relocations Table

As compiled images approach 140k in overall size, they will begin to assemble JSR operations in the

68000 long absolute mode of addressing.

To the JForth system, this means that a table of information must be kept about these references, so that the Amiga Binary Loader can correctly 'relocate' them when the image is loaded into a different part of memory.

JForth automatically generates and manages this table, completely transparent to the programmer. Note, however, that this table is not maintained within the dictionary; but is allocated from the general memory pool provided by the Amiga.

When present, the ABSOLUTE ADDRESS of the table will be contained in a USER variable called ABSRELOCS .

Each increase of 256 Long Relocations results in an additional 1k of memory allocated from the Amiga OS; 'FORGET'ing programs that contain Relocations will automatically return memory when appropriate.

Files and Memory Housekeeping...

Provided the programmer uses the JForth-provided utilities to allocate memory and open files, JForth will keep track of the requests for these resources, returning them in BYE should the programmer forget.

The data areas used to 'remember' memory and file usage is also allocated outside of the contiguous memory area occupied by the executing JForth image.

JForth Compiler

CFA, is the JForth compiler. It is a defered word set to (CFA,) . The JForth compiler is very flexible. You can define a word to be INLINE , CALLED or BOTH (depending on the system settings when the defined word is compiled). The Forth function + is a BOTH type of word. It is so small, 2 bytes , that it will always be compiled inline, since a call would be at least 4 bytes.

```
DEF +  ( will display )
    ADD.L   (DSP)+,TOS
    RTS
```

Compiling + in another word such as...

```
: USE-OF-PLUS  + + + ;
```

Results in the compiler creating...

```
DEF USE-OF-PLUS
    ADD.L   (DSP)+,TOS
    ADD.L   (DSP)+,TOS
    ADD.L   (DSP)+,TOS
    RTS
```

The default setting to new definitions is CALLED . This is for maximum compatiblity with threaded Forths. So the above USE-OF-PLUS will always be called. Note that USE-OF-PLUS is compiled without any threading of any kind. The code for + is simply laid end-to-end, inline in the dictionary. This eliminates the 36 or more cycles that would be required for a JSR ... RTS between each of the only 10 cycles required for the actual addition. This is one of the main reasons JForth is at least 3 times faster than any threaded Forths. Now look at this word:

```
: EITHER    + + + + BOTH ;
```

The BOTH before the ; tells the compiler to mark this word so that when used by another word, it may be compiled inline or called depending on the value of the MAX-INLINE user variable at compile time. EITHER is 8 bytes long, the RTS is not included in the size. If MAX-INLINE is set to anything less than 8 , a call will be compiled to EITHER . If MAX-INLINE is greater than 8 , the code for

EITHER will be laid inline in whatever definition it's used in. So,

```
1 MAX-INLINE !
: EXAMPLE   EITHER  [ 9 MAX-INLINE ! ] EITHER ;
DEF EXAMPLE
    JSR     EITHER
    ADD.L   (DSP)+,TOS
    ADD.L   (DSP)+,TOS
    ADD.L   (DSP)+,TOS
    ADD.L   (DSP)+,TOS
    RTS
```

Note the first EITHER was compiled as a call. After the MAX-INLINE is changed to 9 , EITHER was placed inline.   This dynamic sizing feature of JForth allows you to put speed where you need it, and save space where you don't need as much speed.

Some words must be compiled INLINE , like R> >R (LOOP) and others.

How to Generate a New JForth System

These steps guide the programmer in creating a custom JForth development system. This becomes necessary if the programmer has modified any of the 'jf:' (Extras:JForth) or any 'ju:' (JForth:util) files that are part of the normal JForth image.

NOTE: Modifying the JForth System such that this procedure is necessary can seriously affect our ability to render Technical Support, should you request it. You may be asked to furnish a copy of your changes in this event, which can result in delays, if only for shipping purposes. For this reason, we do not recommend modifications in this area unless suggested by Delta Research or otherwise if absolutely necessary.

1) From CLI, type: run com:jkernal

NOTE: number of bytes available will appear in HEX and the I/O will be in SLOW mode. This will change as compilation progresses.

2) When the JForth window appears you can optionally select a tradeoff between speed and memory utilization. If you have memory to burn and want a slightly faster Forth enter:

```
64 MAX-INLINE !
```

For a smaller but somewhat slower version, enter:

```
6 MAX-INLINE !
```

The default is 8 because JForth is plenty fast already and some systems may be short on memory. You can set MAX-INLINE to anything you want but the practical range is 6 to 256.   If you set MAX-INLINE too high you may run out of memory. You may want to selectively set MAX-INLINE high just for particular words that you want to optimize.

You are now ready to compile the first stage. Enter:

```
INCLUDE JF:BUILDSYS
```

When the compilation has completed, you need to save this first stage somewhere temporarily. If you have plenty of RAM, you could save it in RAM:MINIMUM. You could also save it on a blank floppy disk or on your hard disk. Assuming you have an area reserved for temporary files called TMP:, enter:

```
SAVE-FORTH TMP:MINIMUM    \ or wherever you want
```

3) Enter: BYE

4) From CLI, enter:

```
RUN   TMP:MINIMUM     \ or whatever you called it
```

5) When the window appears, Enter:

```
INCLUDE   JF:LOADJFORTH
```

The loader will ask if you want to INCLUDE the MODULE facility, the HASHed vocabulary search and History. We recommend answering yes, 'Y', to all these questions.   If you do not use Module support then the code will probably not fit in this image. In this case, increase #K then SAVE-FORTH to get a larger image.Modules that are created will be written to the MOD: directory.

6) When the compile has stopped, the programmer may load in any additional files he desires (we recommend adding your files to the list in JF:LOADJFORTH).

7) Type:

```
SAVE-FORTH COM:JForth
```

The just-created COM:JForth image will parallel the released version, reflecting, of course, any changes you have made.   You should now re-compile any applications you have written on top of this new JForth image.