# Chapter 7
# Clone - The JForth Target Compiler

Historically, Forth has provided a strong development environment; fast to write code in and easy to debug. Its natural interactivity strongly contributed to both features. However, Forth has been somewhat limited in its usefulness to the personal computer user, as it does not lend itself well to creating small standalone programs.

The main reason is that Forth has to carry around its dictionary, and usually much of it is never used by the application. This includes the Forth compiler itself, which is almost always disabled by vendor-provided turnkey programs! Other language components that standalone programs often tote along without using include debugging aids, INTERPRET-related words, assemblers, disassemblers, etc.

This is where CLONE comes in, and it really is something quite unique in the Forth world! We are very proud of CLONE and pleased to provide you this sophisticated development tool.

CLONE creates for you *another executable copy* of your compiled JForth program, but does so *outside* of the JForth dictionary. And as it builds this "clone", it *includes only the parts of JForth that are needed by your program!* This greatly reduces the size of your final program.

Also, executables created by CLONE are *YOURS!* You may do with them as you wish...*ROYALTY FREE!* This is an important consideration for commercial projects, as you are not allowed to distribute a JForth image (those with a dictionary) unless it has been processed by JU:TURNKEY to no longer be interactive.

## How To Use Clone

1. Compile the CLONE program...
   a. From CLI or Shell, enter: Run Com:JForth
   b. From JForth, enter: INCLUDE CL:TOPFILE

Clone consumes approximately 30K of dictionary. (We recommend SAVE-FORTHing an image with CLONE compiled so that this step need not be repeated each time.)

2. Compile the program to be CLONEd.
   a. You may have to increase the dictionary size (via #K) and SAVE-FORTH to have enough room.

3. Enter: CLONE <NAME>

   where: <NAME> is the name of the main entry point of your program. Wait for "ok".

4. Enter: SAVE-IMAGE <NAME> <FILENAME> [-s -m -icon]

   where: <NAME> is the name of the main entry point of your program.

   <FILENAME> is the path/name of the executable file

options:

| | |
|---|---|
| -s | Include a debug symbol table for Wack |
| -m | Write a "filename.map" file (this can be big) |
| -icon | Create a "filename.info" icon file |

Notes:

Executables created by CLONE may be launched from CLI or Workbench.

The above mentioned main entry point should *not* accept parameters on the stack, but rather in the command line, like CLI commands. These are available immediately when your program is started; the usual parsing words like WORD function normally to retrieve this information; the next WORD or LWORD operation moves the next argument on the command line to HERE.

## Technical Information About Clone

CLONE is an expert at analyzing compiled JForth code. Given a main entry point, it derives a complete call dependency tree sufficient to rebuild a separate executable image of minimum size, including only those words necessary to support the run time environment of the main program.

In the process, the NAME-FIELD, LINK-FIELD, and SIZE-FIELD components are also filtered out, also reducing the size of the resultant executable. The standard JForth Demos image (loadable from the JD: directory of your release disks) shrank from about 160K, in the JForth dictionary, to less than 28K after passing through CLONE. No demo source file was altered from the original V1.2 release.

MINIMUM TARGET SIZE

The minimum size of a CLONEd JForth program is about 3 Kbytes. This is the code required to support both launching from CLI and Workbench environments. This can be verified by CLONEing the resident JForth word NOOP with the NOCONSOLE variable (described below) turned ON.

SPECIAL NOTE ON JFORTH "STATE"...

Since CLONE depends heavily on the resident dictionary to function as a model for the final program, the state of the JForth environment during CLONE is crucial. For example, if you type the JForth word SLOW before CLONEing (causing JForth to use unbuffered input/output to the console window) the CLONEd program will also exhibit this behavior.

Another example of how the state of JForth directly affects the output file is in the use of the compiler *MAX-INLINE* variable; whatever its setting when the program was compiled will affect the size of the created target.

## Clone Glossary

These words form the user interface for the CLONE program. They are usually typed at the keyboard to create your standalone program...

**`CLONE ( -- , <name> )`**

A two-pass program which first creates a call dependency tree for *<name>*, then uses the tree to build a separate program image in memory, of minimum size.

**`STATS ( -- )`**

Prints out a status report for the previous CLONE operation. This includes the name of each included routine, its address in both the normal dictionary and the new image, and the number of times it is referenced by other routines in the new image. (This same information is saved in the '.map' file if the *-m* option is included on the CLONE command line.)

**`SAVE-IMAGE ( -- , <name> <filename> [<-s -m -icon>] )`**

Creates an executable file on the disk for <name>, which must exist somewhere within the current call dependency tree. Accepts optional parameters. For more details, see *HOW TO USE CLONE,* above.

**`SHOWME ( -- , <name> )`**

Once CLONE has been performed, disassembles the CLONEd code for <name>, using target-image-relative addressing.

**INITCLONE ( -- )**

Re-Initializes CLONE, freeing all CLONE-related resources and allowing another CLONE operation to be performed. Normally done between CLONE evolutions.

Two words are available to your program for special "cancel" behavior (applicable only if run fromCLI or Shell):

**CANCELKEY? ( -- n1 )**

Returns an ascii C, D, E, or F if the corresponding control key has been pressed at the keyboard; 0 otherwise. See ENABLE_CANCEL in the section "Customizing the CLONEd Image".

Prior to compiling the CLONE program, CANCELKEY? will unconditionally return 0.

**CANCELNOW? ( -- )**

If ENABLE_CANCEL has been set non-zero, calls CANCELKEY? to check if CTRL-C, D, E, or F has been pressed at the keyboard. If so, the image is exited; otherwise, no action occurs. See ENABLE_CANCEL in the section *Customizing the CLONEd Image*.

Prior to compiling the CLONE program, CANCELNOW? executes NOOP.

## Customizing the CLONEd Image

Several words, mostly state variables, are available to the programmer to control CLONE's key run time behavior. Default values are employed that satisfy most programs, but for best results, each should be set or verified just prior to a CLONE operation. Refer ahead to the section *Clone Configuration File*, for an example of how this is done conveniently.

**TRACKING ( -- addr )**

This is a variable, used as a boolean. If non-zero, CLONE will include all the normal memory and file pointer tracking mechanisms that are used in the JForth development image; this feature automatically returns still-allocated memory, and closes still-opened files and standard Amiga libraries when the image is exited, but can increase the size of the executable and affect the run-time performance of those sections of the program which are memory allocation/deallocation intensive.

If zero, CLONE will not include any of this support, leaving the total responsibility for memory, files and libraries to the application. Well-behaved, debugged programs are always meticulous in these respects; such extra housekeeping is unnecessary (and arguably undesirable).

The default state for TRACKING is OFF.

**NOCONSOLE ( -- addr )**

Another boolean variable, this controls whether I/O support for a normal AmigaDOS CLI window should be included. If your program relies on the standard Forth KEY, EMIT, ?TERMINAL, and/or TYPE words (including ."), you should set this variable OFF. This is the configuration for CLI-oriented commands like DIR, regardless of what kind of console window (RAW:, CON: or NEWCON:) is used.

On the other hand, if your program does not directly call any of the Forth I/O routines (such as a graphics-intensive game), NOCONSOLE should be turned ON. This will exclude the run-time code for the normal I/O primitives and yield a smaller executable.

The default state for NOCONSOLE is OFF.

**ENABLE_CANCEL ( -- addr )**

This boolean variable matters only to CLI-based programs and, when set non-zero, causes CLONEd programs to automatically exit via QUIT on the first call to KEY after CTRL-C, D, E or F has been typed by the user. See the description of CANCELNOW? in the Clone Glossary section

ENABLE_CANCEL must be set non-zero BEFORE CLONEing if this ability is desired in the CLONEd program.

The default state for ENABLE_CANCEL is OFF.

**RAWEXPECTECHO ( -- addr )**

Another boolean variable, used to adjust for differences in behavior between the 2 types of CLI-based Amiga windows, RAW: and CON: (also NEWCON:, after WorkBench 1.2). Should be set ON if your program opens a RAW: window; this causes EXPECT to echo typed characters. Characters are automatically echoed by a CON: or NEWCON: window, so for these two types, RAWEXPECTECHO should be turned OFF.

The default state for RAWEXPECTECHO is OFF.

**IFLEAVELONG ( -- addr )**

This variable should be set ON if you expect your CLONEd image to exceed 128K, OFF otherwise. If an image is larger than 128K it may need to use long relocatable JSRs which need four bytes instead of the normal two bytes for short register relative JSRs. If CLONE finds that this variable needs to be set, it will abort the current CLONE operation and inform you. To recover from this error, enter:

```
INITCLONE
IFLEAVELONG ON
```

Then try the CLONE operation again.

The default state for IFLEAVELONG is OFF.

**IFLONGBRANCH ( -- addr )**

This variable should be set ON if CLONE aborts and tells you to set it. To recover from this error, enter:

```
INITCLONE
IFLONGBRANCH ON
```

Then try the CLONE operation again.

The default state for IFLONGBRANCH is OFF.

**STACKSIZE ( -- addr )**

This variable is examined by CLONE to determine the initial size for the DATA stack, allocated when the image is run. Forth programs are normally conservative in the use of the data stack; 4K is usually quite sufficient.

The default setting for STACKSIZE is 4096.

NOTE: This should not be confused with the setting of the "stack" in the WorkBench 'info' menu command. That parameter determines the size of the JForth return stack (AmigaDOS defaults this value to 4096, also; this satisfies most JForth programs here, too).

**DICTIONARYSIZE ( -- addr )**

This variable is examined to determine the size of the workspace at HERE in the target image. As in most Forth memory maps, HERE returns the address where compiled code ends, and marks the first available free address of the dictionary. Even though the concept of a dictionary does not exist in CLONEd programs, the area is commonly referenced as a scratch string area, so SAVE-IMAGE adds "DictionarySize @'" number of bytes to the code segment as a small workspace. It should be noted that the specified area actually resides in the saved executable, so programs should allocate larger workspaces rather than using the area above HERE.

The default setting for DICTIONARYSIZE is 256, allowing 128 bytes for PAD, and another 128 bytes below that, shared by HERE and the number formatter. This default size represents a minimum.

**INITIALIMAGESIZE ( -- addr )**

This variable becomes important when you are CLONEing with a minimum of free memory available. Before CLONEing, if you set INITIALIMAGESIZE to slightly (1 or 2 kbytes) more than the final image size, the amount of memory needed for CLONE to complete will be reduced by as much as 50%.

For example, if your CLONEd application size normally ends up at about 33k, and your last CLONE failed due to insufficient memory, enter:

```
INITCLONE  \ to free up memory and allow CLONE to re-init
DECIMAL 35 1024 * INITIALIMAGESIZE ! \ slightly more than needed
```

Try CLONEing again; much less memory should be needed this time. Of course, you may be just plain too low or fragmented for this to help (if so, reboot the Amiga to unfragment).

Please note that one of the first things CLONE does is to read INITIALIMAGESIZE and try to allocate that much memory. Therefore, if you set it to something prohibitively high, CLONE will seem to fail immediately due to insufficient memory.

By default, INITIALIMAGESIZE is set to 4096.

**ERRORCLEANUP ( -- )**

An optional DEFERed word allows the programmer to specify one cleanup word that will be executed IF the CLONEd program terminates via **QUIT** or **ABORT** (usually due to a fatal error). If used, it should return all Amiga resources that have been allocated (close files, free memory etc.), but *should take care not to free something already freed.* The stack diagram for a word placed into ERRORCLEANUP should be ( -- ).

ERRORCLEANUP is not called if the application terminates normally.

By default, ERRORCLEANUP is set up to execute NOOP. *THE APPLICATION MUST SET THIS VECTOR AT RUNTIME* by including a line similar to the following in its initialization code...

```
' MyErrorCleanup is ErrorCleanup
```

**USERCLEANUP ( -- )**

An optional DEFERed word allows the programmer to specify one cleanup word that will be executed when the CLONEd program terminates **normally**. If used, it should return all Amiga resources that have been allocated (close files, free memory etc.), but *should take care not to free something already freed.* The stack diagram for a word placed into USERCLEANUP should be ( -- ).

USERCLEANUP is not called if the application terminates via QUIT or by ABORTing.

By default, USERCLEANUP is set up to execute NOOP. *THE APPLICATION MUST SET THIS VECTOR AT RUNTIME* by including a line similar to the following in its initialization code...

```
' MyUserCleanup is UserCleanup
```

## Clone Configuration File

One convenient, transparent method of configuring the clone variables is to include something like the following at the end of the main load file for your program:

```
EXISTS? clone    \ are we compiling on top of CLONE?
.IF              \ YES, set CLONE how I want it.
    include CLONE.CONFIG
.THEN
```

A sample CLONE.CONFIG follows:

```
\ Sample CLONE.CONFIG file  (set CLONE run-time behavior)
\
\ Keep in working directory, conditionally INCLUDE
\ from load file

decimal

Tracking      off
NoConsole     off
Enable_Cancel off
RawExpectEcho off
IfLeaveLong   off
4096 StackSize !
 256 DictionarySize !
4096 InitialImageSize !
' noop is ErrorCleanUp
' noop is UserCleanUp
```

## Word Redefinitions under Clone

Some JForth words functionally change due to the standalone nature of the targeted program. Some are described here; see the file CL:REDEFS.F for a complete listing.

QUIT - terminates the image.

INTERPRET - a null word, does nothing. (No Dictionary possible)

?PAUSE - again, a null word...the CLI provides 'pause'.

WORD - preserves case, like LWORD.

Also, since there are no name fields in a CLONEd image, there can be niether dictionary nor reason to CLONE words that operate on such. Examples of such words include FIND VOCABULARY ORDER DEF DISM WORDS VLIST, etc. Most have been rendered ineffectual in CLONEd images, others could cause problems. Don't worry, though. It is actually quite hard to accidentally include these in your CLONEd program.

Note: you are *not permitted* to include any JForth *compiler primitive* or other such *code generating utility or program* in your standalone image, via CLONE or any other method or tool. This includes words such as CFA, CREATE : ASM CODE etc. Actually, these have been specifically written to not be CLONE-compatible, so they will not function correctly in a standalone image anyway.

## How To Be Clone Compatible

Programs destined to be CLONEd should follow these guidelines:

**ODE**

An ODE program that is to be cloned must call OB.INIT at the beginning to set up the object stack and to initialize the dynamic object tracking. It must also load the file JO:CLONE_SUPPORT if not already loaded.  See the chapter on ODE for more information.

**Use Supported Data Structures**

Use the system tools wherever possible. VARIABLE, ARRAY, CREATE, and DOES> are all supported data structures that will be faithfully reproduced in the final image both in data content and functionality.

The DEFER and GLOBAL-DEFER words are the only currently supported means of vectoring execution.  An example CLONE-able execution array is provided below.

**Runtime Initialization of Data-Storage Elements which contain Addresses**

CLONE will automatically relocate the contents of DEFERed and GLOBAL-DEFERed words, as well as compiled address references created with  ' <pronounced tick> or ALITERAL.  Use of these tools insures CLONE-compatibility.

Similarly, the addresses returned by CREATE-DOES> children (such as VARIABLEs or ARRAYs) will also reflect the new addresses of their data in the CLONEd image, however, the contents of their data area will be unmodified (since they often do not contain addresses).  This means that if you save addresses in such elements, your program must initialize them when it runs, before using them.  This is accomplished simply by compiling in a statement like the following (assuming a VARIABLE is being used):

```
   ' MyWord MyVariable !  \ initialize MyVariable to point to
   MyWord's CFA
```

For more example code, see the sample execution array later in this document.

Do not assign addresses as CONSTANTs.  CONSTANT values are not changeable and therefore can't be re-initialized later by your program.  Of course, if the address can never change (such as the address of a hardware device), then use of CONSTANT is appropriate as CLONE should not relocate them.

All STRUCT elements are considered by CLONE to be data-storage and those that hold addresses (even if defined as APTRs) should be initialized by the program.

**Assembly Code**

Be especially careful when using either assembler format when writing CLONE-compatible programs; the address generation problem noted above is easy to create.  Always use the function as would the compiler to generate its CFA or data address at run time.  For example, if you want to load the relative address of a variable called MYVARIABLE  into A0:

*WRONG*, MyVariable's data addr is built at "assembly-time" and "hard-coded"...

```
   ( RPN Example )
       MyVariable #   ar0 an   long   move
   ( Forward ASM example )
       move.l  #[MyVariable],a0
```

*CORRECT*, 'MyVariable' is "asked" at run-time for its addr, then moved to a0..

```
   ( RPN Example )
       ] MyVariable [              \ pushes tos, puts addr in tos
       tos dn   0ar an  move       \ move it to A0
       dsp a@+  tos dn  move       \ restore original TOS
   ( Forward ASM example )
       callcfa  MyVariable          \ pushes TOS, puts addr into TOS
       move.l   tos,a0              \ move it to A0
       move.l   (dsp)+,tos          \ restore original TOS
```

Since the compiler is temporarily invoked to build the reference to MYVARIABLE, CLONE will later be able to relocate that code segment.

**Run Time Initialized Example Execution Array**

An example execution array:

```
3 ARRAY EXECARRAY        \ 3 possible executable words

: INIT-ARRAY   ( -- , set up addresses in exec array )
  ' NOOP       0 EXECARRAY !    \ put 'NOOP' cfa in 1st elmnt
  ' OPEN.ALL   1 EXECARRAY !    \ put 'OPEN.ALL' cfa in 2nd elmnt
  ' CLOSE.ALL  2 EXECARRAY !    \ put 'CLOSE.ALL' cfa in 3rd
elmnt
;

: EXECUTE-FROM-ARRAY  ( #element -- , fetch and execute )
  EXECARRAY @EXECUTE
;
```

This program will work as long as INIT-ARRAY is called in the main program before the array is used.

## Differences Between Original and Cloned Code

Small but distinct differences may be noted between the dictionary and CLONEd versions of programs, primarily due to optimizations that CLONE performs.

1. While VARIABLE references were compiled inline in JForth V1.2, the compiler in later versions will create CALLs to the VARIABLE's cfa in the normal dictionary. CLONE, however, converts them to inline in the generated image, reclaiming the speed advantage.
2. All occurences of USER variables are converted to the VARIABLE data structure. Functionally, there is no difference in JForth programs, and the VARIABLE construct is faster.
3. The target compiler will alter the particular manner that another function was called as appropriate based on the new target address. For example, if TEST1 called TEST2 via a JSR absolute in the original dictionary, CLONE will change it to a more efficient and smaller relative reference if possible in the new image.
4. All VERIFY-LIBS error checking is stripped from calls to Amiga libraries.
5. If the IFLONGBRANCH variable is set TRUE, all short ,8 bit, branch displacements are converted to long, 16 bit.