Developing Applications With MSP430 CamelForth

As of version 0.4, CamelForth/MSP430 has the ability to save and autostart user-defined application programs. To fully understand this mechanism, you need to know how the Forth dictionary is implemented, and how it relies on User Variables.

The Forth "dictionary" is the list of all word names and definitions. The name of each word is sometimes referred to as the "header," and the definition of each word -- its code or data -- is sometimes referred to as the "body." When you define a new word, its header and body are added to the current end of the dictionary, in Flash ROM. [1] The current end of the ROM dictionary is always given by the Instruction Dictionary Pointer, **IDP**. IDP is a variable (specifically, a "user variable"). Its current value is returned by the word IHERE, which simply does **IDP @**

Some Forth words, such as VARIABLEs, also allocate some space in RAM. The current end of the allocated RAM is always given by the Dictionary Pointer, **DP**. DP is another user variable. Its current value is returned by the word HERE.

Finally, all of the headers of all the Forth words (including the kernel words) are joined together in a singly-linked list. This allows all the known names to be searched when you type or compile a word name. When you define a new Forth word, it is linked to the list of previous words, so the head of the linked list is always the last word in the Forth dictionary. The address of the last word in the dictionary is given by the user variable **LATEST**. Note that LATEST and IDP are very different! LATEST will point to the *start* of the last definition in the dictionary, while IDP will point past the *end* of the last definition (that is, to the first free location in ROM).

When you coldstart CamelForth, IDP DP and LATEST are set to their "cold" (or "factory") values: the first location in "application" ROM, the first location in "application" RAM, and the last word in the CamelForth kernel, respectively. As you compile new Forth words, IDP DP and LATEST will change.

As you compile new Forth words, they will be burned into Flash ROM, so they will still be present if you remove and restore power. However, the crucial user variables, IDP DP and LATEST, reside in RAM and will be lost. Without them, you cannot find your new words, and your data might get corrupted. [2] To preserve these variables, and restore them on a reset, you use the word **SAVE**.

SAVE

The word SAVE copies all of the Forth "user variables" (system variables) to the MSP430's "Information Memory" (a small dedicated area of Flash ROM). To be precise, it copies the system variables, the RAM interrupt vectors (if used), and the beginning of the application RAM to Information ROM. [3] You do *not* need to erase that ROM before using SAVE; SAVE will erase it automatically.

In order that all of your Forth words are included, you should use SAVE after the *last* word in your application has been defined. After compiling your application and using SAVE, you have

a) all of your application code in Flash ROM, and
b) the crucial system variables in Information ROM.

Autostart Process

When CamelForth/MSP430 (version 0.4 or later) is reset -- either a power-on reset or a hardware reset -- it does the following:

1. Basic CPU and peripheral registers are initialized.
2. The Information ROM is copied to the User Area (system variables), RAM interrupt vectors (if used), and the start of Application RAM.
3. If the restored RAM data is valid [4], the word at LATEST (the last word in your application) is examined.
4. If the word at LATEST is a valid colon definition, it is executed. This will normally be the MAIN word that starts your application. The name of this word is unimportant; what matters is that it is the last word in the dictionary (the very last word in your application).

However...

5a. If the word at LATEST is *not* a colon definition, **or**
5b. If the restored RAM data is *not* valid, **or**
5c. Regardless of the above, if the autostart bypass jumper is installed, then
6a. The word COLD is executed, which
6b. Copies "cold start" ("factory") values to the User Area (system variables), and
6c. Starts the Forth interpreter.

So, if you compile your application, and then do a SAVE, the next time you reset the CPU, your application will be automatically started.

If you compile your application and forget to do SAVE, the next time you reset the CPU, you will *appear* to have an basic Forth dictionary (WORDS will show only the CamelForth kernel words). You'll also get this if you install the autostart bypass jumper and do a reset.

Note that although conditions 5a-5c will restart CamelForth with just the kernel words visible, *your application is still in Flash ROM,* and if you did a SAVE, *your saved RAM data is still in Information ROM.* This is a feature, not a bug!

Suppose your application fails to autostart -- it crashes, or loops uncontrollably, or does nothing. You can install the autostart bypass jumper (see below) to "cold start" the CPU into the Forth interpreter, and then use Forth commands to examine your application in Flash ROM, or to examine your saved RAM data in Information ROM.

SCRUB and RESTORE

When you "cold start" CamelForth after compiling your application, LATEST points to the last word in the CamelForth kernel, and IDP is pointing to the start of the application ROM, but *your application is still present.* This means that although you can't see your application with WORDS, and you can't run any of your application words from the Forth interpreter, you can't compile new words, either! This is because the Flash ROM where new words would be added is not in an erased state -- it contains your application program.

At this point you have three options.

1.You can use *interpretive commands only* to examine the application ROM and the Information ROM. These commands would include DUMP, @, and C@. For example, if your application completely failed to start, you can use DUMP to examine the saved RAM data in the Information ROM, to see if BASE is valid and if LATEST points to the correct application word.

2. In versions 0.41 and later [5], the word RESTORE will reset the system variables from the saved Information ROM. This will reset IDP, DP, and LATEST to their SAVE state, and your application will reappear in the dictionary! So you can run specific application words from the Forth interpreter, to see what they're doing (and perhaps what they're doing wrong). You can even type MAIN (or whatever you called it) to try restarting the application. If you had some unused flash ROM at the end of your application, you can add new Forth definitions.

Even if the application program is faulty, this will work as long as the dictionary links are intact and the saved system variables are valid. (Warning: if you forgot to do a SAVE, and then do a RESTORE, the interpreter will certainly crash!) If, however, the dictionary links or SAVE data is corrupted -- or you forgot to do SAVE -- your only recourse is option #3.

3. The word SCRUB will erase the entire application flash ROM, and reset the system variables to their "cold start" values. This essentially returns the CPU to its "factory fresh" condition, with only the CamelForth kernel installed, ready to accept a new application program. (One small difference is that SCRUB does not erase the Information ROM.)

To summarize: after an autostart bypass, you can
1. Use *interpretive commands only* (such as DUMP) to examine memory.
2. Use RESTORE to attempt to return to the "fully programmed" state, or
3. Use SCRUB to return to the "empty and ready to be programmed" state.

Note: RESTORE is provided for development and debugging purposes. There should never be a need for your application program to call RESTORE.

Autostart Bypass

The autostart bypass will always reset the CPU into the CamelForth interpreter. The application Flash ROM and information ROM may contain your code and data, but the CamelForth interpreter is intact and the kernel dictionary is valid. If an autostart bypass does *not* display a startup prompt on the terminal, and provide a working Forth interpreter, then the CamelForth kernel has been corrupted, and your only option is to reflash the MSP430 part.

To bypass autostart on a CPU reset:
:
MSP430G2553: jumper P1.3 to ground. On the TI Launchpad board, you can do this by holding S2 pressed while resetting the CPU.

MSP430F1611: jumper P3.3 to ground. On the NMI Tini430 board, you can do this by jumpering pins 2 and 3 of J3. NOTE: if you are not using a Tini430 board, P3.3 must have an external pullup resistor to Vcc. As this pin is used for the I2C interface, such a pullup will normally be present.

Note that the autostart bypass does not erase either program ROM or Information ROM. This means that you can install the autostart bypass jumper to get access to the Forth interpreter, use interpretive commands to examine memory -- and then remove the bypass jumper and reset to restart your application.

Summary of behaviors

	Application ROM	**System variables IDP, DP, LATEST**	**Information ROM**
initial "factory" state	erased	initialized to "COLD" state: IDP = start of App ROM DP = start of App RAM LATEST = last kernel word	erased
compile new words	user-defined words (user's application)	IDP, DP, LATEST are updated to reflect the space allocated to the application	no change
SAVE	no change	no change	copied from user area and application RAM
reset, with valid SAVE data	no change	copied from Information ROM	no change
reset, with invalid SAVE data	no change	initialized to "COLD" state	no change
reset, with autostart bypass	no change	initialized to "COLD" state	no change
COLD	no change	initialized to "COLD" state	no change
SCRUB	erased	initialized to "COLD" state	no change
RESTORE (ver. 0.41 and later)	no change	copied from Information ROM	no change

Remember: if you have reset the system variables to the "COLD" state, but you have not erased your application from ROM, you *cannot* compile new Forth definitions.

Creating an "Autostartable" Application

First, and most important: all of your application must be compiled to Flash ROM. From version 0.4 onwards, this is the default behavior for CamelForth/MSP430, so you shouldn't have to worry about this. But CamelForth also allows you to compile code to RAM -- for debugging purposes -- and if you have done this, that code will be lost the next time you reset the CPU. Application programs must reside entirely in ROM!

As mentioned above, the word which starts your application must be a colon definition, and must be the last word you compile. This is normally always the case. If not, it can always be *made* to be the case by adding one more colon definition at the end of your application. Suppose, for example, that your application is to be started by running a CODE (machine code) word. You can then call that word from a colon definition:

```
CODE MAIN ... END-CODE
: STARTUP MAIN ;
```

Similarly, suppose for some reason you need to add some debugging words that appear *after* your MAIN word (perhaps they need to access an address within your MAIN word). You can then do:

```
: MAIN  ... ;
: DEBUG1 ... ;
: DEBUG2 ... ;
: STARTUP  MAIN ;
```

What matters to CamelForth is that your autostart word is the *last* word in the dictionary, and that it be a colon definition. Its name is not important. Most of the time, your application word *will* be the last word in the dictionary, and you don't have to worry about this. But if you're a person who likes to append some debugging words at the end of your application source file, you need to be aware of this.

Finally, your MAIN application word (whatever you call it) *must never return.* You have two choices:

1. Your application word can contain an infinite loop (such as BEGIN...AGAIN). This is typically the case for embedded applications.

2. Your application word can end by calling ABORT, which will start the Forth interpreter. This can be useful for debugging, or if you need to provide a "back door" for system administration. (Once you're in the Forth interpreter, you can always type MAIN to restart the application.)

It's good practice to include an ABORT after your application in case of a programming error. For example:

```
: MAIN   APP-INIT APP-LOOP ABORT ;
```

If for some reason APP-LOOP terminates, ABORT will be executed instead of the system just crashing. And if you have a terminal connected, you can see that the application has terminated because it will begin responding to Forth commands. (You may also want to send a text prompt before calling ABORT, because ABORT just "quietly" starts the Forth interpreter.)

MARKER

During the development process, you may wish to erase *part* of your application so that you can recompile it. Suppose that your lower-level support words are fully debugged, but you need to keep modifying and reloading the higher-level words to get them to work. If you just keep reloading the source code, you'll fill up the application ROM. What you need is some way to erase *part* of your application from ROM, so you can re-use that ROM space. This is the function of MARKER.

MARKER is a defining word, whcih creates a "self-erasing" word. It's easiest to explain by example. At the start of your "developmental" code, you define a MARKER word:

```
MARKER DEVSTUFF
... your application code...
```

... more application code...

If you then use WORDS, before your application code you will see a word named DEVSTUFF. The name is not important -- you can use any name you want for the "marker" word. Now you can test your application code. When you want to erase your application code, you simply type

DEVSTUFF

The word DEVSTUFF (which was created by MARKER) will erase all of the application code you loaded after DEVSTUFF was defined -- and will erase itself! It backs up the dictionary pointers (IDP DP and LATEST), *and* it erases the flash ROM that was used by the code it removes. Basically it restores the state of the dictionary as it was before the MARKER word was defined.

If you're downloading code from a file, a common practice is to put a MARKER definition as the first definition in that file. Then you can type DEVSTUFF (or whatever you called it) whenever you want to reload the file, to erase the previous version of whatever you're reloading.

Note also that you can have multiple MARKER words. You can place one wherever you want a "checkpoint" of the dictionary state, so you can back up to different positions in your compile. However, there is one caution you should bear in mind before using lots of MARKERs:

Because flash ROM can only be erased a "page" at a time, MARKER skips ahead to a page boundary before compiling the marker word. This ensures that the marker word and everything following can always be completely erased, without affecting any previous words. But it also means that MARKER can allocate up to *a full page of flash ROM* whenever it is used. On the MSP430G2553, for example, a page of flash ROM is 512 bytes, so four MARKERs in a row will use up 2K of ROM.

This problem does not occur when you compile the same MARKER over and over again, because when the marker word erases itself, it also "backs up" over the space it allocated.

If you use MARKER, normally you'll just use one, so this isn't a major problem. After you've finished debugging, and are ready to compile your completed application, you should remove the MARKER from your source file.

Notes

1. With CamelForth/MSP430, new definitions may be added to Flash ROM or to RAM. In version 0.3, RAM was used by default, and you had to reset the dictionary pointer to compile to ROM. In versions 0.4 and later, Flash ROM is used by default.

2. The Forth kernel uses HERE as a temporary buffer. If you have defined some VARIABLEs, and then HERE gets reset to its "cold" state, the kernel will overwrite your variables.

3. You can use SAVE to save a few key variables in your application that need to be automatically restored when the CPU is reset. This is not important to understanding the autostart process; it is an incidental benefit. The amount of application RAM which is saved varies among different members of the MSP430 family. The MSP430G2553, for example, currently saves the first 96 bytes of application RAM.

4. Currently (version 0.41), the saved system variables are considered valid if BASE is either 2, 8, 10, or 16. To be precise, if any bit *other* than bits 4, 3, and 1 ($10, $08, $02) is set in BASE, then BASE is considered invalid.

5. In version 0.4, which was only released for the MSP430G2553, the word RESTORE is not included. You can get the identical effect by typing **HEX 1000 U0 80 CMOVE** to copy the Information ROM to the User Area. Note that you must type this as an interpretive command, because you can't add it as a Forth definition until you've reset the dictionary pointers.