This Draft document contains all the errata discovered to date in IEEE Std 1275, Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices. To provide a context for the changes, the entire numbered section of all sections containing changes is included. Change bars indicate the modified text lines. To facilitate discussion and verification of the changes, deleted text is formatted as strike-out and added text is underlined. In the final document, the deleted text will be hidden and the added text unmarked (the change bars will remain).

The IEEE editor will generate the rest of their boilerplate text, but we will have to provide the list of committee members at the time this document (P1275.7) is approved.

-John Rible, draft editor

Introduction

<Unchanged body of introduction>

The following individuals were members of the P1275 Working Group at the time IEEE Std 1275-1994 ~~this document~~ was produced:

William M. (Mitch) Bradley, *Chair*
David M. Kahn, *Vice Chair*
John Rible, *Draft Editor*
Luan D. Nguyen~~**Mike Williams**~~, *Secretary*

Paul Fischer	Thanos Mentzelopoulos	Paul Thomas
Ron Hochsprung	David Paktor	Mike Tuciarone
	Michael Saari	Mike Williams

3.6.4 Expansion bus device class template

A memory-mapped bus logically extends the processor's memory address space to include the *devices* on that bus, allowing the use of processor load and store cycles to directly address those devices. The details vary from bus to bus. This standard does not specify the adaptation of Open Firmware to any particular bus, but other related standards do so specify (see 2.1).

3.6.4.1 Bus-specific methods and properties

This subclause lists a set of *methods* that deal with requirements common to most memory-mapped buses. This subclause is intended as a suggested starting point for the development of complete sets of methods for particular buses; also see related standards, such as IEEE Std 1275.2-1994. The methods provide mapping services for establishing the correspondence between processor virtual and device *physical addresses,* allocation of DMA memory, and probing to locate *plug-in devices*.

map-in	(phys.lo ... phys.hi size -- virt)	Map the specified region; return a virtual address.
map-out	(virt size --)	Destroy mapping from previous **map-in**.
dma-alloc	(... size -- virt)	Allocate a memory region for later use.
dma-free	(virt size --)	Free memory allocated with **dma-alloc**.
dma-map-in	(... virt size cacheable? -- devaddr)	Convert virtual address to device bus DMA address.
dma-map-out	(virt devaddr size --)	Free DMA mapping set up with **dma-map-in**.
dma-sync	(virt devaddr size --)	Synchronize (flush) DMA memory caches.
probe-self	(arg-str arg-len reg-str reg-len fcode-str fcode-len --)	Evaluate FCode as a child of this node.

The following properties are specific to this class of *device node*:

"**ranges**"	Standard *property name* to define a bus-specific address translation. ~~device's physical address.~~
"**#address-cells**"	Standard *property name* to define the package's address format.
"**#size-cells**"	Standard *property name* to define the package's address *size* format.

3.6.4.2 Bus-specific register access words

Some expansion bus adaptors have characteristics that interfere with the semantics of the register-access words, **rb@**, **rb!**, **rl@**, **rl!**, **rw@**, and **rw!**. For example, some bus bridges swap bytes and others buffer their write operations. Bus packages for such devices shall substitute, as with **set-token**, conforming implementations of the register-access words during the evaluation of their childrens' FCode programs.

Such bus-specific register-access words can be written in terms of the "generic" register-access words, in the expectation that the parent bus will have substituted implementations of those words that handle any peculiar characteristics imposed by the parent bus. For example, suppose a bus adapter device supports a bus that reverses the ordering of doublets within quadlets. It can not predict the characteristics of its parent bus, for example, whether its parent bus buffers ordering of bytes within a doublet. The FCode program for such a bus adapter would include a definition of a bus-specific quadlet register-read word that might look something like this:

```
: my-rl@ ( qaddr -- quad )  rl@ lwflip ;
```

When the FCode for this word's definition is evaluated, the definition of **rl@** that will get compiled-in will be the one that was substituted by the node of the parent bus. The node of the present bus adapter would then, before evaluating its childrens' FCode programs, perform a substitution that might look something like this:

```
['] my-rl@ h# 234 set-token
```

Because such a substitution takes place at the time of evaluation of an FCode program, it becomes problematical to ensure that the semantics of the intended register-access words will be visible at the user interface level. For

this reason, it is recommended that device nodes for child-devices supply definitions for the register-access words that will bind the semantics that were substituted for the device's bus to the user interface name. Such a definition can be very simple, and might look something like this:

```
: rl@ ( qaddr -- quad )  rl@ ;
```

The net effect of such a definition would be that when the device is selected, and the user enters the name **rl@**, the user interface interpreter will find the name that occurs in the device's node; that name would be bound to the behavior that was installed in the FCode interpretation token-table at the time the device node's FCode was evaluated, causing the correct behavior to be executed.

3.6.5 Memory management device class template

An MMU is a *device* that performs address translation between a CPU's *virtual addresses* and the *physical addresses* of some bus, typically the bus represented by the *root node*. In general, the details are both processor-specific and bus-specific. This standard does not specify the adaptation of Open Firmware to any particular MMU, but other related standards may so specify (see 2.1). This standard does not require the presence of an MMU.

This subclause lists a set of *methods* that deal with requirements common to most MMUs. This subclause is intended as a suggested starting point for the development of complete sets of methods for particular MMUs. The methods provide services for fine-grained control of the allocation and mapping of virtual addresses, particularly intended for use by *client programs* through the **call-method** *client interface* service (see also the "**mmu**" *property* of the **/chosen** node). In general, the use of these methods makes a client program system-specific; nevertheless, they are useful in some circumstances. The arguments and results shown are intended as guidelines; particular MMUs might require additional arguments or changes to the arguments shown.

The presence of an MMU node does not imply that the Open Firmware is necessarily using virtual-to-physical address translation hardware.

The following methods, defined in the glossary, are recommended for MMU *packages*:

claim	([virt] size align -- base)	Allocate (claim) addressable resource
release	(virt size --)	Free (release) addressable resource.
map	(phys.lo ... phys.hi virt size mode --)	Create address translation
unmap	(virt size --)	Invalidate existing address translation.
modify	(virt size mode --)	Modify existing address translation.
translate	(virt -- false | phys.lo ... phys.hi mode true)	Translate virtual address to physical address.
dma-alloc	(size -- virt)	Allocate a memory region for later use.
dma-free	(virt size --)	Free memory allocated with **dma-alloc**.

NOTE—Although, in general, methods for some busses have optional parameters, the method for physical addresses that use the deblocker package shall not require optional parameters.

Additional requirements for the **claim** and **release** methods:
— The address format, *virt*, is a single-cell *virtual address*.
— The allocation length, *size*, is a single cell.
— The allocated resource is a region of virtual address space.

The following properties are recommended for MMU packages:

"**available**"	The property values are as defined for the standard "**reg**" format, with single-cell virtual addresses. The regions of virtual address space denote the virtual address space that is currently unallocated by the Open Firmware and is available for use by client programs.

"**existing**"	The value of this property defines the regions of virtual address space managed by the MMU in whose package this property is defined without regard to whether or not these regions are currently in use. The encodings of *virt* and *len* are MMU-specific.
"**page-size**"	The value of this property is the number of bytes in the smallest mappable region of virtual address space.

NOTE—Freeing virtual address space does not necessarily free any associated physical resource. The correct sequence of operations for freeing mapped memory is to first use **unmap**, thus destroying the translation. Then the physical memory and virtual address space can be freed with the **release** methods of the respective nodes.

4.3 Path resolution

This section defines the process of resolving a *device path* given by a device-specifier. There are three contexts in which this can occur:

— **find-device**. In this context, the intention is to locate the named *device node* and select it as the *active package* without any other side effects.

— **open-dev**. In this context, the intention is to *open* every node named in the path by executing its **open** method, thereby creating an instance chain, and to return the *ihandle* of the node at the tail end of the chain (the node farthest from the root node).

— **execute-device-method**. In this context, the intention is to open every node named in the path *except for the last node*. An instance chain is created, including an instance for the last node, but instead of executing that last node's **open** method, a different method, given as an argument, is executed. Then the open instances are *closed* and the instance chain is destroyed.

The overall structure of the path resolution process is the same in all three contexts. This description shows it as one process with conditional tests at places where the details are context-dependent. However, it need not be implemented that way; for example, each context could be implemented separately.

The process is described in English as a set of procedures, each consisting of steps that are generally executed in order, with the scope of conditional tests shown by indentation and looping structures shown by labels and "go to" lines. It makes liberal use of variable names to identify intermediate data items. The scope of such variables is "global" with respect to the procedures. The use of these variable names does not imply that an implementation must or should use such variables; they are used solely for descriptive purposes. Similarly, the description of the process in terms of procedures does not imply that the implementation should be so structured; the separate procedures were used in the description so that the top-level description would not be unwieldy.

The following notation describes the parsing of pathnames into component parts:

left-split(*string*, "*x*") -> *initial*, *remainder*

String, *initial*, and *remainder* are the names of string variables, and "*x*" is a character.

Left-split divides *string* into two disjoint substrings, setting *initial* to the portion of *string* before the first occurrence of the character "*x*", and *remainder* to the portion of *string* following the first occurrence of the character "*x*". Neither *initial* nor *remainder* contains that first occurrence of "*x*", although *remainder* may contain other later occurrences of that character. If *string* does not contain the character "*x*", *initial* is set to *string* in its entirety, and *remainder* is set to the empty string.

~~right-split(*string*, "*x*") -> *initial*, *remainder*~~

~~Right-split is similar to left-split, except that the division of the string occurs around the last occurrence of the character "*x*", rather than the first.~~

split-before(*string*, "*x*") -> *initial, remainder*

Split-before is similar to left-split, except that if "*x*" occurs in *string*, *remainder* begins with the first occurrence of "*x*" (left-split removes the first occurrence of "*x*" from *remainder*)

split-after(*string*, "*x*") -> *initial, remainder*

Split-after is similar to split-before, except that the division of the string occurs after the last occurrence of the character "x". If "x" occurs in *string*, *initial* ends with "*x*", and *remainder* begins with the first character, if any, following that last occurrence of "*x*". If "*x*" does not occur in *string*, *initial* is set to *string* and *remainder* is set to the empty string.

The use of the preceding notation does not necessarily imply the existence of functions named left-split, split-before, and split-after ~~and right-split~~; it is simply a notational convention. (This standard does define a function **`left-parse-string`** whose semantics are very similar to left-split, but the details of returning the results are somewhat different.)

In searching for a matching node, the order in which the various *child nodes* are considered is unspecified. At the implementation's discretion, if no match is found among the children, the search may be widened to include the children's children, recursively to any depth.

In the following algorithmic description (4.3.1 through 4.3.5), the text enclosed in boxes is commentary describing the intention of the algorithm. The text outside of the boxes is definitive.

4.3.1 Path resolution procedure (top level procedure)

If the pathname *does not begin with "/", and its first* node name *component is an* alias*, replace the* alias *with its expansion.*

a) If PATH_NAME does not begin with the "/" character,

1) ~~Left-split~~Split-before(PATH_NAME, "/") -> HEAD, TAIL.

2) ~~Left-split~~Split-before(HEAD, ":") -> ALIAS_NAME, ALIAS_ARGS.

3) If ALIAS_NAME matches a defined alias,

i) Replace ALIAS_NAME with its alias value.

ii) If ALIAS_ARGS is not empty:

a) ~~Right-split~~Split-after(ALIAS_NAME, "/") -> ALIAS_HEAD, ALIAS_TAIL.

b) ~~Right-split~~Split-before(ALIAS_TAIL, ":") -> ALIAS_TAIL, DEAD_ARGS.

c) ~~If ALIAS_HEAD is not empty,~~
Concatenate(ALIAS_HEAD, ~~"/",~~ALIAS_TAIL, ALIAS_ARGS) -> ALIAS_~~TAIL~~NAME.

~~d) Concatenate(ALIAS_TAIL, ":", ALIAS_ARGS) -> ALIAS_NAME.~~

iii) ~~If TAIL is empty, replace PATH_NAME with ALIAS_NAME.~~

iv) ~~Otherwise (when TAIL is not empty),~~
Concatenate(ALIAS_NAME, ~~"/",~~ TAIL) -> PATH_NAME.

If the pathname*, after possible alias expansion, begins with "/", begin the search at the root node. Otherwise, begin at the* active package*.*

<<<The remainder of clause 4.3.1 is unchanged.>>>

5.2.2.2 FCode-offset

Either

byte	Encodes an 8-bit signed (two's complement) offset.

or

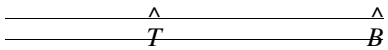
byte.high byte.low	Encodes a 16-bit signed (two's complement) offset.

A conditional or looping control transfer is represented by a pair of *FCode functions*. An *FCode-offset* specifies the number of bytes in the FCode program between two corresponding components of a control flow construct. The offset is calculated as the number of FCode bytes from the first byte of the offset to the byte just after the "target" of the control transfer. A positive offset corresponds to a transfer of control in the "forward" (towards the end of the FCode program) direction, and a negative offset corresponds to the "backward" (towards the beginning of the FCode program) direction.

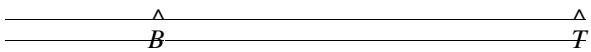
The following control transfer pairs are meaningful, with "**...**" representing an arbitrary sequence of FCode bytes:

Example source construct	**FCode control transfer pair**
begin ... again	**b(<mark) ... bbranch** *FCode-offset* *T*^ *B*^
begin ... until	**b(<mark) ... b?branch** *FCode-offset* *T*^ *B*^
if ... then	**b?branch** *FCode-offset* **...** **b(>resolve)** *B*^ *T*^
if ... else ... then	**b?branch** *FCode-offset1* **...** **bbranch** *FCode-offset2* **...** **b(>resolve)** *B1*^ *B2*^ *T1*^ *T2*^
do ... +loop	**b(do)** *Fcode-offset1* **...** **b(loop)** *FCode-offset2*
do ... loop	**b(do)** *Fcode-offset1* **...** **b(+loop)** *FCode-offset2*
?do ... loop	**?b(do)** *Fcode-offset1* **...** **b(loop)** *FCode-offset2*
?do ... +loop	**?b(do)** *Fcode-offset1* **...** **b(+loop)** *FCode-offset2* *B1*^ *T2*^ *B2*^ *T1*^
case ...	**b(case) ...**
... of ... endof	**... b(of)** *Fcode-offset1* **...** **b(endof)** *FCode-offset2*
...	*B1*^ *B2*^ *T1*^
... of ... endof	**... b(of)** *Fcode-offset3* **...** **b(endof)** *FCode-offset4*
...	*B3*^ *B4*^ *T3*^
endcase	**b(endcase)**
	T2^
	T4^

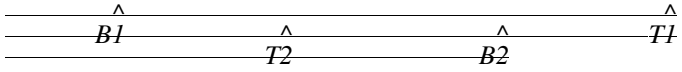
FCode control transfer pair	Example source construct
b(<mark) ... **bbranch** *FCode-offset*	**begin ... again**
b(<mark) ... **b?branch** *FCode-offset*	**begin ... until**



bbranch *FCode-offset* ... **b(>resolve)**	**... else ... then**
b?branch *FCode-offset* ... **b(>resolve)**	**if ... then**



b(do) *FCode-offset1* ... **b(loop)** *FCode-offset2*	**do ... loop**
b(do) *FCode-offset1* ... **b(+loop)** *FCode-offset2*	**do ... +loop**
b(?do) *FCode-offset1* ... **b(loop)** *FCode-offset2*	**?do ... loop**
b(?do) *FCode-offset1* ... **b(+loop)** *FCode-offset2*	**?do ... +loop**



The markers $B^{\wedge}$ and $T^{\wedge}$ show the "branch" and "target" locations used for the calculation of the value of *FCode-offset*. The value is the signed number of FCode bytes between *B* and *T* (positive if *B* is before *T*). *Bn/Tn~~B1/T1~~* ~~are for~~ refer to the corresponding *FCode-offsetn~~1~~* ~~and B2/T2 are for FCode-offset2~~.

NOTE—On some devices, FCode programs are stored with "gaps" between successive FCode bytes. For example, each FCode byte might be stored in the least significant byte of a separate quadlet, in which case it might be necessary to add four to the address to advance to the next FCode byte. This does not affect the calculation of an *FCode-offset*—the offset is in terms of the number of FCode bytes, independent of how those bytes are addressed.

The offset size (whether of 8 bits or 16 bits) is established at the beginning of the FCode program by the particular start code that begins the FCode program. **version1** sets the offset size to 8 bits, and the other start codes (**start0**, **start1**, **start2**, and **start4**) set the offset size to 16 bits. The offset size may be changed from 8 bits to 16 bits by executing **offset16**.

In most cases (the exceptions are **bbranch** and **b?branch** in interpretation state), the *FCode evaluator* needs only the sign of the offset, not its numerical value. In these cases, the value of the offset is essentially redundant because control transfers are represented by pairs of FCode functions (a branching function and its target). The offset indicates the distance between the branch and its target, but that information can be derived during the FCode evaluation process without needing the offset value. However, standard FCode programs are required to have numerically correct offsets (as described in the above paragraph) for compatibility with existing practice.

6.3.2.5 Control transfer

```
boot
        IN:   [string] bootspec
        OUT:  none
```

Exits the *client program*, resets the system (as with the command **reset-all**), and reboots the system with the device and arguments given by the null-terminated string *bootspec*. The string *bootspec* is interpreted in the same manner as the arguments of the command **boot**.

`enter`

`IN: none`
`OUT: none`

If an IEEE std 1275 user interface exists, exit the client program and e~~E~~nter~~s~~ the Open Firmware command interpreter (e.g., called by the operating system after a console input device abort). The client program may be resumed if the user continues execution with the **`go`** command.

Otherwise, if another user interface exists, transfer control to that interface.

When no user interface exists, return control to the client program.

`exit`

`IN: none`
`OUT: none`

If an IEEE std 1275 user interface exists, e~~E~~xit~~s from~~ the client program and enter the Open Firmware command interpreter. The execution of the client program may not be resumed.

Otherwise, if another user interface exists, transfer control to that interface.

When no user interface exists, idle.

`chain`

`IN: [address] virt, size, [address] entry, [address] args, len`
`OUT: none`

Frees *size* bytes of memory starting at virtual address *virt*, then executes another client program beginning at address *entry*. The argument buffer *args, len* is copied into the Open Firmware memory and passed to the other program. The address of the arguments in the Open Firmware memory is the client program's second argument, and their length is its third argument. `chain` is used to free any remaining memory for a secondary boot program and begin executing the booted program.

NOTE—The behavior of the `chain` *client interface service* includes the functions of **`init-program`** and **`go`** on behalf of the new client program, but does not include the functions of reading the client program into memory, parsing its header, or allocating its memory.

6.3.2.6 User interface

`interpret`

`IN: [string] cmd, stack-arg1, ..., stack-argP`
`OUT: catch-result, stack-result1, ..., stack-resultQ`

Pushes one less than *N-args* items, *stack-arg1*, ..., *stack-argP*, onto the Forth data stack, with *stack-arg1* on top of the stack; executes the null-terminated string *cmd* as a Forth command line guarded by **`catch`**. Pops the result returned by **`catch`** into *catch-result*. If that result is nonzero, restore the depth of the Forth data stack to its depth prior to the execution of `interpret`. If that result is zero, pops up to one less than *N-returns* items, *stack-result1*, ..., *stack-resultQ*, from the Forth data stack into the returned values portion of the argument array, with *stack-result1* corresponding to the top of the stack.

N-args and *N-returns* are stored in the argument array and may be different for different calls to `interpret`. If the number of items *X* left on the Forth data stack as a result of the execution of *cmd* is less than *N-returns*, only *stack-result1, ..., stack-resultX* are modified; other elements of the returned values portion of the argument array are unaffected. If *X* is more than *N-returns*, additional items are popped from the Forth data stack after setting *stack-result1, ..., stack-resultQ* so that, in all cases, the execution of `interpret` results in no net change to the depth of the Forth data stack.

An implementation shall allow at least six *stack-arg* and six *stack-result* items.

`interpret` is optional; it need be present only if the Open Firmware user interface is present.

```
set-callback
      IN:   [address] newfunc
      OUT:  [address] oldfunc
```

Client programs may define a routine for handling the Open Firmware routines **`callback`** and **`sync`**. *Newfunc* is the address of the entry point of the callback routine. This service sets the callback handler to *newfunc* and returns as *oldfunc* the address of the entry point of the previously installed callback handler.

The Open Firmware shall use the same calling conventions specified in 6.3.1 for *client interface services* when calling the callback handler function. See **`callback`** and **`$callback`** glossary entries for details.

A client program callback handler shall return either a nonzero error code in the *ret1* cell of the argument array if the service indicated by the service argument is unavailable, or zero otherwise. The client program callback handler shall return any additional results in the *ret2 ... retN* cells, setting *N-returns* to the total number of return values including the error code (or zero) that is in the *ret1* cell. The handler shall not store more than *M* results, where *M* is the value that was in the *N-returns* cell when the handler was called, nor shall the returned value of *N-returns* exceed *M*.

```
set-symbol-lookup
      IN:   [address] sym-to-value, [address] value-to-sym
      OUT:  none
```

Sets the symbol table resolution **`defer`** words **`sym>value`** and **`value>sym`** so that they execute the client program callbacks whose addresses are given by the arguments *sym-to-value* and *value-to-sym*, respectively. If either argument is zero, the corresponding **`defer`** word is set to the action of **`false`**.

sym-to-value is called as follows:

```
      IN:   [string] symname
      OUT:  error, symvalue
```

The `service` name string in the argument array is a pointer to a null-terminated string containing "`sym-to-value`".

Searches for a symbol whose name is *symname*. If such a symbol is found, returns zero in *error* and the symbol's value in *symvalue*. If no such symbol is found, returns –1 in *error* and zero in *symvalue*.

value-to-sym is called as follows:

```
      IN:   symvalue
      OUT:  offset, [string] symname
```

The `service` name string in the argument array is a pointer to a null-terminated string containing "`value-to-sym`".

Locates the symbol whose value is closest to but not greater than *symvalue* and returns *offset*, the non-negative offset from the value of that symbol to *symvalue*, and *symname*, the symbol name. If *symvalue* is less than the value of any known symbol, or is insufficiently close to any symbol value according to an implementation-dependent criterion, returns –1 in *offset* and the empty string in *symname*.

`set-symbol-lookup` is optional; it need be present only if the Open Firmware user interface is present and the Client Program Debugging *command group* (see 7.6) is implemented.

Annex A
Open Firmware glossary

bljoin (b1.lo b2 b3 b4.hi -- quad) F 0x7F

Join four bytes to form a quadlet.

~~The high bits of each of the four bytes must be zero.~~

Combine the eight least significant bits of each operand to form a quadlet. Other operand bits shall be zero.

b(<mark) (F: --) F 0xB1

Target of backward branch implemented with **bbranch** or **b?branch**.

FCode evaluation: (F: --)

Perform the interpretation or compilation semantics of **begin**.

FCODE ONLY (Tokenized by **begin**)

b(>resolve) (--) F 0xB2

(F: --)

Target of forward branch implemented with **bbranch** or **b?branch**.

FCode evaluation: (F: --)

If in interpretation state:

Do nothing

If in compilation state:

Perform the compilation semantics of **then**. Then, if the current definition is temporary and the depth of the control flow stack is the same as its depth when the temporary current definition was initiated, perform the FCode evaluation semantics of **b(;)** and execute the temporary current definition.

FCODE ONLY (Tokenized by **else**, **then**, and **repeat**)

bwjoin (b.lo b.hi -- w) F 0xB0

Join two bytes to form a doublet.

~~The high bits of each of the two bytes must be zero.~~

Combine the eight least significant bits of each operand to form a doublet. Other operand bits shall be zero.

encode-int (quad ~~n~~ -- prop-addr 4 ~~prop-len~~) F 0x111

Encode a number into a *prop-encoded-array*.

The *property encoding* of a (quadlet) number is a sequence of 4 bytes, with the most significant byte first (i.e., at the smallest address).

No alignment is implied; the sequence of 4 bytes begins at the first available location.

Used as: `5000 encode-int ( prop-addr prop-len )`

find-package (name-str name-len -- false | phandle true) F 0x204

Locate the support package named by *name string*.

If the package can be located, return its *phandle* and **true**; otherwise, return **false**.

Interpret the name in *name string* relative to the "**/packages**" device node. If there are multiple packages with the same name (within the "**/packages**" node), return the *phandle* for the most recently created one.

is-install (xt --) F 0x11C

Create **open**, other methods for this display device.

Create methods for accessing the display device driver in the *active package*. *xt* is the execution token of a routine to initialize the display device, whose stack diagram is (--).

Used as: `['] my-open-routine is-install`

Create the following methods:

open
When later called, execute the display driver's "my-open-routine" (whose execution token is *xt*) guarded by **catch**. If the execution of *xt* results in a **throw**, return **false**. Otherwise, ~~and~~ initialize the *terminal emulator* and return **true**.

write
When later called, pass its argument string to the *terminal emulator* for interpretation.

draw-logo
When later called, execute the display driver's "my-draw-logo" procedure which was installed into the **defer** word **draw-logo** by the driver's "my-open-routine".

restore
When later called, execute the display driver's "my-reset-screen" procedure which was installed into the **defer** word **reset-screen** by the driver's "my-open-routine".

lbsplit (quad -- b.lo b2 b3 b4.hi) F 0x7E

Split a quadlet into 4 bytes.

~~The high bits of the 4 bytes are zero.~~
All but the least significant eight bits of each result value shall be zero.

lwsplit (quad -- w1.lo w2.hi) F 0x7C

Split a quadlet into two doublets.

~~The high bits of the two doublets are zero.~~
All but the least significant sixteen bits of each result value shall be zero.

"**network**" S

Packet-oriented network device type.

Standard string value of the "**device_type**" property for network devices with IEEE 802 packet formats.

A standard package with this "**device_type**" property value shall implement the following methods.

open, **close**, **read**, **write**, **load**

Additional requirements for the **open** method:

Execute **mac-address** and create a "**mac-address**" property with that value.

Additional requirements for the **read** method:

Receive (non-blocking) a network packet, placing at most the first *len* bytes of that packet into memory starting at *addr*. Return the number of bytes actually received (not the number copied into memory), or -2 ~~zero~~ if no packet is currently available.

Discard packets containing hardware-detected errors, as though they were not received.

NOTE—In general, -2 indicates no data was available at the time **read** was done and -1 indicates that an error occurred. Zero is generally used only for devices where data arrives in records, packets, or other such container, and indicates that a valid but empty container was received.

Additional requirements for the **write** method:

Transmit the network packet of length *len* bytes from the memory buffer beginning at *addr*. Return the number of bytes actually transmitted.

The packet to be transmitted begins with an IEEE 802 Media Access Control (MAC) header.

Usage restriction: The caller must supply the complete header; the source hardware address will not necessarily be "automatically inserted" into the outgoing packet.

Additional requirements for the **load** method:

Read the default client program into memory, starting at *addr*, using the default network booting protocol.

A standard package with this "**device_type**" property value may implement additional device-specific methods.

A standard package with this "**device_type**" property value shall implement the following property if the associated device has a preassigned IEEE 802.3-style MAC (network) address:

"**local-mac-address**"

NOTE—Such packages often use the "**obp-tftp**" support package to implement the "**load**" method.

See also: "**address-bits**", "**max-frame-size**"

next-property (previous-str previous-len phandle -- false | name-str name-len true) F 0x23D

Return the *name* of the property following *previous* of *phandle*.

~~*Name* is a null-terminated string that is the name of the property following *previous* in the property list for device *phandle*. If *previous* is zero or points to a zero-length string, *name* is the name of the *phandle's* first property. If there are no more properties after *previous* or if *previous* is invalid (i.e., names a property that does not exist in *phandle*), *name* is a pointer to a zero-length string.~~

Locate, within the property list of the package identified by *phandle*, the first property if *previous-len* is zero, or the property following the property named by the text string *previous-str previous-len* otherwise. Return *name-str name-len* and **true** if such a property exists, or **false** otherwise (i.e. if there are no more properties, or if there is no property in that package with the name given by *property-str property-len*).

A sequence of invocations of **next-property** with the same *phandle* value, beginning with *previous-len* equal to zero, and passing the *name-str name-len* result of the previous invocation as the *previous-str previous-len* argument to the next invocation, continuing until **false** is returned, shall enumerate the names of all properties of that package. The order in which those individual properties are returned is undefined (e.g. the first property returned by **next-property** is not necessarily the one that was created first). If a new property is created within that package between invocations of **next-property** in such a sequence, the new property name may, but need not, be returned as a result of one of the following invocations of **next-property** within that same sequence.

"**page-size**" S

MMU package property name to define the virtual address space page size.

prop-encoded-array:

Integer encoded as with **encode-int**.

The value of this property is the number of bytes in the smallest mappable region of virtual address space.

"**ranges**" S

Standard *property name* to define a bus-specific address translation. ~~device's physical address.~~

Buses such as SBus and VMEbus, whose children can be accessed with CPU load-and-store operations (as opposed to buses such as SCSI or IPI, whose children are accessed with a command protocol), require a way to define the relationship between the physical address spaces of the parent and child nodes. The "**ranges**" property provides this capability.

The value of the "**ranges**" property describes the correspondence between the physical address space defined by a bus node (the "child address space") and the physical address space of that bus node's parent (the "parent address space"). The "**ranges**" property value is a sequence of

```
child-phys  parent-phys  size
```

specifications. *Child-phys* is an address, encoded as with **encode-phys**, in the child address space. *Parent-phys* is an address (likewise encoded as with **encode-phys**) in the parent address space. *Size* is a list of integers, each encoded as with **encode-int**, denoting the length of the child's address range. The number of integers in each *size* entry is determined by the value of the **#size-cells** property of *this* node (the node in which the "**ranges**" property appears) or 1 if the **#size-cells** property is absent. The interpretation of *size* is bus-dependent.

Each specification defines a one-to-one correspondence between the child addresses in the range *child-phys..child-phys+size*–1 and the parent addresses in the range *parent-phys..parent-phys+size*–1. The address ranges thus described might (and often will) define a sparse address space, i.e., the address ranges need not be consecutive in either the child address space or the parent address space. Successive specifications are encoded one after another. It is recommended, but not required, that the specifications be sorted in ascending order of *child-phys*.

If a "**ranges**" property exists but has a zero-length property value, the child address space is identical to the parent address space.

The absence of a "**ranges**" property for a bus node indicates that there is no direct correspondence between the child address space and the parent address space, e.g., the bus is a command protocol bus such as SCSI.

Example: Suppose that a 4-slot, 28-bit SBus is attached to a machine whose physical address space consists of a 32-bit memory space (*phys.hi*=0) and a 32-bit i/o space (*phys.hi*=1). The SBus slots are numbered 0, 1, 2, and 3, and appear in i/o space at addresses 0x80000000, 0x90000000, 0xA0000000, and 0xB0000000, respectively. The "**ranges**" property for this "**sbus**" device node would contain the encoded form of the following sequence of numbers:

Child Address		Parent Address		Size
phys.hi	phys.lo	phys.hi	phys.lo	
0	0	1	8000.0000	1000.0000
1	0	1	9000.0000	1000.0000
2	0	1	A000.0000	1000.0000
3	0	1	B0000000	1000.0000

See also: "**#size-cells**".

rb! (FCode function) (byte addr --) F 0x231

Store a byte to device register at *addr*.

Data is stored with a single access operation and flushes any intervening write buffers, so that the data reaches its final destination before the next FCode function is executed.

Register is stored with identical bit ordering as the input stack item.

NOTE—A bus device can substitute (as with **set-token**) a bus-specific implementation of **rb!** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. See clause 3.6.3.1 for further details and for user interface semantics.

~~**rb!** (user interface) (byte addr --)~~

~~Store a byte to device register at *addr*.~~

~~**Compilation:** (--)~~

~~Perform the equivalent of the phrase:~~

```
h# 231 get-token if execute else compile, then
```

~~**Interpretation:** (byte addr --)~~

~~Perform the equivalent of the phrase:~~

```
h# 231 get-token drop execute
```

~~**NOTE—**A bus device can substitute (see **set-token**) a bus-specific implementation of **rb!** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. The given user interface semantics of **rb!** ensure that such substitutions are visible at the user interface level.~~

rb@ (FCode function) (addr -- byte) F 0x230

Fetch a byte from device register at *addr*.

Data is read with a single access operation.

Result has identical bit ordering as the original register data.

NOTE—A bus device can substitute (as with **set-token**) a bus-specific implementation of **rb@** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. See clause 3.6.3.1 for further details and for user interface semantics.

~~**rb@** (user interface) (addr -- byte)~~

~~Fetch a byte from device register at *addr*.~~

~~**Compilation:** (--)~~

~~Perform the equivalent of the phrase:~~

```
h# 230 get-token if execute else compile, then
```

~~**Interpretation:** (addr -- byte)~~

~~Perform the equivalent of the phrase:~~

```
h# 230 get-token drop execute
```

~~**NOTE—**A bus device can substitute (see **set-token**) a bus-specific implementation of **rb@** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. The given user interface semantics of **rb@** ensure that such substitutions are visible at the user interface level.~~

rl! (FCode function) (quad qaddr --) F 0x235

Store a quadlet to device register at *qaddr*.

Data is stored with a single access operation and flushes any intervening write buffers, so that the data reaches its final destination before the next FCode function is executed.

Register is stored with identical bit ordering as the input stack item.

NOTE—A bus device can substitute (as with **set-token**) a bus-specific implementation of **rl!** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. See clause 3.6.3.1 for further details and for user interface semantics.

~~**rl!** (user interface) (quad qaddr --)~~

~~Store a quadlet to device register at *qaddr*.~~

~~**Compilation:** (--)~~

~~Perform the equivalent of the phrase:~~

```
h# 235 get-token if execute else compile, then
```

~~**Interpretation:** (quad qaddr --)~~

~~Perform the equivalent of the phrase:~~

```
h# 235 get-token drop execute
```

~~**NOTE—**A bus device can substitute (see **set-token**) a bus-specific implementation of **rl!** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. The given user interface semantics of **rl!** ensure that such substitutions are visible at the user interface level.~~

rl@ (FCode function) (qaddr -- quad) F 0x234

Fetch a quadlet from device register at *qaddr*.

Data is read with a single access operation.

Result has identical bit ordering as the original register data.

NOTE—A bus device can substitute (as with **set-token**) a bus-specific implementation of **rl@** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. See clause 3.6.3.1 for further details and for user interface semantics.

~~**rl@** (user interface) (qaddr -- quad)~~

~~Fetch a quadlet from device register at *qaddr*.~~

~~**Compilation:** (--)~~

~~Perform the equivalent of the phrase:~~

```
h# 234 get-token if execute else compile, then
```

~~**Interpretation:** (qaddr -- quad)~~

~~Perform the equivalent of the phrase:~~

```
h# 234 get-token drop execute
```

~~**NOTE—**A bus device can substitute (see **set-token**) a bus-specific implementation of **rl@** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. The given user interface semantics of **rl@** ensure that such substitutions are visible at the user interface level.~~

rw! (FCode function) (w waddr --) F 0x233

Store a doublet *w* to device register at *waddr*.

Data is stored with a single access operation and flushes any intervening write buffers, so that the data reaches its final destination before the next FCode function is executed.

Register is stored with identical bit ordering as the input stack item.

NOTE—A bus device can substitute (as with **set-token**) a bus-specific implementation of **rw!** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. See clause 3.6.3.1 for further details and for user interface semantics.

~~**rw!** (user interface) (w waddr --)~~

~~Store a doublet *w* to device register at *waddr*.~~

~~**Compilation:** (--)~~

~~Perform the equivalent of the phrase:~~

```
~~h# 233 get-token if execute else compile, then~~
```

~~**Interpretation:** (w waddr --)~~

~~Perform the equivalent of the phrase:~~

```
~~h# 233 get-token drop execute~~
```

~~**NOTE—**A bus device can substitute (see **set-token**) a bus-specific implementation of **rb!** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. The given user interface semantics of **rb!** ensure that such substitutions are visible at the user interface level.~~

rw@ (FCode function) (waddr -- w) F 0x232

Fetch a doublet *w* from device register at *waddr*.

Data is read with a single access operation.

Result has identical bit ordering as the original register data.

NOTE—A bus device can substitute (as with **set-token**) a bus-specific implementation of **rw@** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. See clause 3.6.3.1 for further details and for user interface semantics.

~~**rw@** (user interface) (waddr -- w)~~

~~Fetch a doublet *w* from device register at *waddr*.~~

~~**Compilation:** (--)~~

~~Perform the equivalent of the phrase:~~

```
~~h# 232 get-token if execute else compile, then~~
```

~~**Interpretation:** (waddr -- w)~~

~~Perform the equivalent of the phrase:~~

```
~~h# 232 get-token drop execute~~
```

~~**NOTE—**A bus device can substitute (see **set-token**) a bus-specific implementation of **rw@** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. The given user interface semantics of **rw@** ensure that such substitutions are visible at the user interface level.~~

wbsplit (w -- b1.lo b2.hi) F 0xAF

Split a doublet *w* into two bytes.

~~The high bits of each of the two bytes are zero.~~

All but the least significant eight bits of each result value shall be zero.

wljoin (w.lo w.hi -- quad) F 0x7D

Join two doublets to form a quadlet.

~~The high bits of each of the two doublets must be zero.~~

Combine the sixteen least significant bits of each operand to form a quadlet. Other operand bits shall be zero.

Annex C
The tokenizer

C.3.3 Fcode start and end

These commands create an FCode header for the Open Firmware source between them.

`fcode-version2` (--) T

Begin tokenizing an Fcode program.

Prepare the tokenizer to tokenize subsequent source text. Output the Fcode# for **`start1`** followed by an Fcode header. Some of the fields of the Fcode header are filled-in later by **`fcode-end`**.

FCODE ONLY command.

`fcode-end` (--) T

Finish tokenizing an Fcode program.

Output the Fcode# for end0. Stop tokenizing the current Fcode program. Replace the checksum and length fields of the Fcode header with the program's checksum and length.

FCODE ONLY command.

Annex E
SCSI host adapter package class

E.6.1 overall.fth

```
\ FCode driver for hypothetical SCSI host adapter

fcode-version2
hex

" XYZI,scsi"          name          \ Name of device node
" XYZI,12346-01"      model         \ Manufacturer's model number
" scsi-2"             device-type   \ Device implements SCSI-2 method set
3 0                   intr          \ Device interrupts on level 3, no vector

external

\ These routines may be called by the children of this device.
\ This card has no local buffer memory for the SCSI device, so it
\ depends on its parent to supply DMA memory.  For a device with
\ local buffer memory, these routines would probably allocate from
\ that local memory.

: dma-alloc    ( n -- vaddr )  " dma-alloc" $call-parent  ;
: dma-free     ( vaddr n -- )  " dma-free" $call-parent  ;
: dma-sync     ( vaddr devaddr n -- )  " dma-sync" $call-parent  ;
: dma-map-in   ( vaddr n cache? -- devaddr )  " dma-map-in" $call-parent  ;
: dma-map-out  ( vaddr devaddr n -- )  " dma-map-out" $call-parent  ;
: max-transfer ( -- n )
   " max-transfer"  ['] $call-parent catch  if  2drop h# 7fff.ffff  then
   \ The device imposes no size limitations of its own; if it did, those
   \ limitations could be described here, perhaps by executing:
   \    my-max-transfer min
;

```

```
fload scsiha.fth

fload hacom.fth

   new-device
      fload scsidisk.fth  \ scsidisk.fth also loads scsicom.fth
   finish-device

   new-device
      fload scsitape.fth  \ scsitape.fth also loads scsicom.fth
   finish-device

fcode-end
end0
```

E.6.2 scsiha.fth

```
\ Example FCode driver for a hypothetical SCSI bus interface device

hex

\ The following structure defines the registers for the SCSI device.
\ This hypothetical device is designed for ease of programming.  It
\ has a separate register for each function (no bit packing).  All
\ registers are both readable and writeable.  The device has a random-
\ access buffer large enough for a maximum-length SCSI command block.

\ To execute a SCSI command with this device, write the appropriate
\ information into the registers named ">cmd-adr" through ">input?", write
\ a 1 to the ">start" register, and wait for the ">start" register to
\ change to 0.  Then read the ">phase" register to determine whether or
\ not the command completed all phases (">phase" reports 0 on success,
\ h# fd for incoming reset, h# ff for other hardware error).
\ If so, ">status" contains the SCSI status byte, and ">message-in"
\ contains the command-complete message byte.

struct  ( scsi-registers )
  0c field >cmd-adr              \ Up to 12 command bytes
   4 field >cmd-len              \ Length of command block

   4 field >data-adr             \ Base address of DMA data area
   4 field >data-len             \ Length of data area

   1 field >host-selectid        \ Host's selection ID
   1 field >target-selectid      \ Target's selection ID
   1 field >input?               \ 1 for data output; 0 for data input
   1 field >message-out          \ Outgoing message byte

   1 field >start                \ Write 1 to start.  Reads as 0 when done.
   1 field >phase                \ Reports the last transaction phase
   1 field >status               \ Returned status byte
   1 field >message-in           \ Incoming message byte

   1 field >intena               \ Write 1 to enable interrupts.
   1 field >reset-bus            \ Write 1 to reset the SCSI bus.
   1 field >reset-board          \ Write 1 to reset the board.
constant /scsi-regs


\ Now that we have a symbolic name for the size of the register block,
\ we can declare the "reg" property.
```

```
\ Registers begin at offset 800000 and continue for "/scsi-regs" bytes.

my-address 80.0000 +  my-space  /scsi-regs  reg


-1 instance value regs            \ Virtual base address of device registers

0 instance value my-id            \ host adapter's selection ID
0 instance value his-id           \ target's selection ID
0 instance value his-lun          \ target's unit number

\ Map device registers

: map  ( -- )
   my-address 80.0000 +  my-space  /scsi-regs  ( addr-low addr-high size )
   " map-in" $call-parent   to regs            ( )
;
: unmap  ( -- )
   regs /scsi-regs  " map-out" $call-parent  -1 to regs
;

create reset-done-time 0 ,
create resetting false ,

\ 5 seconds appears to be about the right length of time to wait after
\ a reset, considering a variety of disparate devices.
d# 5000 value scsi-reset-delay

: reset-wait  ( -- )
   resetting @  if
      begin  get-msecs reset-done-time @ -  0>=  until
      resetting off
   then
;

: reset-scsi-bus  ( -- )
   1 regs >reset-board rb!        \ Reset the controller board.
   0 regs >intena      rb!        \ Turn off interrupts.
   1 regs >reset-bus   rb!        \ Reset the SCSI bus.

   \ After resetting the SCSI bus, we have to give the target devices
   \ some time to initialize their microcode.  Otherwise the first command
   \ may hang, as with some older controllers.  We note the time when it
   \ is okay to access the bus (now plus some delay), and "execute-command"
   \ will delay until that time is reached, if necessary.
   \ This allows us to overlap the delay with other work in many cases.

   get-msecs scsi-reset-delay + reset-done-time !  resetting on
;

0 value scsi-time       \ Maximum command time in milliseconds
0 value time-limit      \ Ending time for command

: set-timeout  ( msecs -- )   to scsi-time ;

0 value devaddr

\ Returns true if select failed
: (exec)  ( dma-adr,len dir cmd-adr,len -- hwresult )
   reset-wait           \ Delay until any prior reset operation is done.

   his-lun h# 80 or  regs >message-out rb! \ Set unit number; no disconnect.
   my-id      regs >host-selectid   rb!    \ Set the selection IDs.
   his-id     regs >target-selectid rb!
```

```
   \ Write the command block into the host adapter's command register

   dup 0  ?do                        ( data-adr,len dir cmd-adr,len )
      over i + c@                    ( data-adr,len dir cmd-adr,len cmd-byte )
      regs >cmd-adr i ca+ rb!        ( data-adr,len dir cmd-adr,len )
   loop                              ( data-adr,len dir cmd-adr,len )

   regs >cmd-len rl!   drop          ( data-adr,len dir )

   my-id 7 + to my-id                ( initialize for this instance )

   \ Set the data transfer parameters.

   ( .. dir ) regs >input?   rb!     ( data-adr,len )  \ Direction
   ( .. len ) regs >data-len rl!     ( data-adr )      \ Length
   ( .. adr ) regs >data-adr rl!     ( )               \ DMA Address

   \ Now we're ready to execute the command.

   1  regs >start  rb!                    \ Tell board to start the command.

   get-msecs scsi-time +  to time-limit   \ Set the time limit.

   begin  regs >start rb@  while          \ Wait until command finished.

      scsi-time  if                       \ If timeout is enabled, and
         get-msecs time-limit -  0>=  if  \ the time-limit has been reached,
            reset-scsi-bus  true  exit    \ reset the bus and return error.
         then
      then

   repeat

   \ Nonzero phase means that the command didn't finish.

   regs >phase rb@
;

\ Returns true if select failed
: execute-command ( data-adr,len dir cmd-adr,len -- hwresult | statbyte false)
   \ Temporarily put dir and cmd-adr,len on the return stack to get them
   \ out of the way so we can work on the DMA data buffer.

   >r >r >r                     ( data-adr,len )

   dup  if                      ( data-adr,len )

      \ If the data transfer has a nonzero length, we have to map it in.

      2dup  false  dma-map-in   ( data-adr,len dma )
      2dup swap  r> r> r>       ( data-adr,len dma dma,len dir cmd-adr,len)

      (exec)                    ( data-adr,len phys hwres)

      >r swap dma-map-out  r>   ( hwresult )
   else                         ( data-adr,len )
      r> r> r>  (exec)          ( hwresult )
   then                         ( hwresult )

   ?dup  0=  if                                    ( hwresult | )
      regs >status rb@  false \ Command finished; return status byte and false.
   then                                            ( hwresult | statbyte 0 )
;
```

```
external

: reset  ( -- )  map  reset-scsi-bus  unmap  ;
reset   \ Reset the SCSI bus when we are probed.

: open-hardware  ( -- okay? )
   map

   \ Should perform a quick "sanity check" selftest here,
   \ returning true if the test succeeds.

   true
;
: reopen-hardware  ( -- okay? )  true  ;

: close-hardware  ( -- )  unmap  ;
: reclose-hardware  ( -- )  ;

: selftest  ( -- 0 | error-code )
   \ Perform reasonably extensive selftest here, displaying
   \ a message and returning an error code if the
   \ test fails and returning 0 if the test succeeds.
   0
;
: set-address  ( unit target -- )
   to his-id  to his-lun
;
```

E.6.4 scsicom.fth

```
\ This file contains some words that are useful for both
\ SCSI disk and SCSI tape device drivers.

\ The SCSI disk and SCSI tape packages need to export dma-alloc and dma-free
\ methods so the deblocker can allocate DMA-capable buffer memory.

external
: dma-alloc  ( n -- vaddr )  " dma-alloc" $call-parent  ;
: dma-free   ( vaddr n -- )  " dma-free"  $call-parent  ;
headers

: parent-max-transfer  ( -- n )  " max-transfer"  $call-parent  ;


\ Calls the parent device's "retry-command" method.  The parent device is
\ assumed to be a driver for a SCSI host adapter (device-type = "scsi").

: retry-command  ( dma-addr dma-len dma-dir cmd-addr cmd-len #retries -- ... )
           ( ... -- false )               \ No error
           ( ... -- true true )           \ Hardware error
           ( ... -- sensebuf false true )  \ Fatal error with extended status
   " retry-command" $call-parent
;


\ Simplified command execution routines for common simple command forms

: no-data-command  ( cmdbuf -- error? )  " no-data-command" $call-parent  ;

: short-data-command  ( data-len cmdbuf cmdlen -- true | buffer false )
   " short-data-command" $call-parent
;
```

```
\ Some tools for reading and writing 2-, 3-, and 4-byte numbers to and from
\ SCSI command and data buffers.  The ones defined below are used both in
\ the SCSI disk and the SCSI tape packages.  Other variations that are
\ used only by one of the packages are defined in the package where they
\ are used.

: +c!  ( n addr -- addr' )  tuck c! 1+  ;
: 3c!  ( n addr -- )  >r lbsplit drop  r> +c! +c! c!  ;

: -c@  ( addr -- n addr' )  dup c@  swap 1-  ;
: 3c@  ( addr -- n )  2 +  -c@ -c@  c@       0  bljoin  ;
: 4c@  ( addr -- n )  3 +  -c@ -c@ -c@  c@      bljoin  ;


\ "Scratch" command buffer useful for construction of read and write commands

create cmdbuf  0 c, 0 c, 0 c, 0 c, 0 c, 0 c, 0 c, 0 c, 0 c, 0 c, 0 c, 0 c,
: cb!  ( byte index -- )  cmdbuf + c!  ;        \ Write byte to command buffer


\ The deblocker converts a block/record-oriented interface to a byte-oriented
\ interface, using internal buffering.  Disk and tape devices are usually
\ block- or record-oriented, but the OBP external interface is byte-oriented,
\ in order to be independent of particular device block sizes.

0 instance value deblocker
: init-deblocker  ( -- okay? )
   " "  " deblocker"  $open-package  to deblocker
   deblocker if
      true
   else
      ." Can't open deblocker package"  cr  false
   then
;

headerless
: selftest  ( -- error? )
   fcode-revision h# 3.0000 >=  if
      my-unit " set-address" $call-parent
      " diagnose" $call-parent
   else
      0
   then
;

headers
```

Annex G
Summary lists

G.2 Assigned FCode numbers

0x00	**end0**	0x44	**between**	0x7B	**noop**
0x01– 0x0F	Beginning codes of 2-byte FCode sequences	0x45	**within**	0x7C	**lwsplit**
		0x46	**drop**	0x7D	**wljoin**
0x10	**b(lit)**	0x47	**dup**	0x7E	**lbsplit**
0x11	**b(')**	0x48	**over**	0x7F	**bljoin**
0x12	**b(")**	0x49	**swap**	0x80	**wbflip**
0x13	**bbranch**	0x4A	**rot**	0x81	**upc**
0x14	**b?branch**	0x4B	**-rot**	0x82	**lcc**
0x15	**b(loop)**	0x4C	**tuck**	0x83	**pack**
0x16	**b(+loop)**	0x4D	**nip**	0x84	**count**
0x17	**b(do)**	0x4E	**pick**	0x85	**body>**
0x18	**b(?do)**	0x4F	**roll**	0x86	**>body**
0x19	**i**	0x50	**?dup**	0x87	**fcode-revision**
0x1A	**j**	0x51	**depth**	0x88	**span**
0x1B	**b(leave)**	0x52	**2drop**	0x89	**unloop**
0x1C	**b(of)**	0x53	**2dup**	0x8A	**expect**
0x1D	**execute**	0x54	**2over**	0x8B	**alloc-mem**
0x1E	**+**	0x55	**2swap**	0x8C	**free-mem**
0x1F	**-**	0x56	**2rot**	0x8D	**key?**
0x20	*****	0x57	**2/**	0x8E	**key**
0x21	**/**	0x58	**u2/**	0x8F	**emit**
0x22	**mod**	0x59	**2***	0x90	**type**
0x23	**and**	0x5A	**/c**	0x91	**(cr**
0x24	**or**	0x5B	**/w**	0x92	**cr**
0x25	**xor**	0x5C	**/L**	0x93	**#out**
0x26	**invert**	0x5D	**/n**	0x94	**#line**
0x27	**lshift**	0x5E	**cat**	0x95	**hold**
0x28	**rshift**	0x5F	**wa+**	0x96	**<#**
0x29	**>>a**	0x60	**la+**	0x97	**u#>**
0x2A	**/mod**	0x61	**na+**	0x98	**sign**
0x2B	**u/mod**	0x62	**char+**	0x99	**u#**
0x2C	**negate**	0x63	**wa1+**	0x9A	**u#s**
0x2D	**abs**	0x64	**la1+**	0x9B	**u.**
0x2E	**min**	0x65	**cell+**	0x9C	**u.r**
0x2F	**max**	0x66	**chars**	0x9D	**.**
0x30	**>r**	0x67	**/w***	0x9E	**.r**
0x31	**r>**	0x68	**/l***	0x9F	**.s**
0x32	**r@**	0x69	**cells**	0xA0	**base**
0x33	**exit**	0x6A	**on**	0xA1*	**convert**
0x34	**0=**	0x6B	**off**	0xA2	**$number**
0x35	**0<>**	0x6C	**+!**	0xA3	**digit**
0x36	**0<**	0x6D	**@**	0xA4	**-1**
0x37	**0<=**	0x6E	**l@**	0xA5	**0**
0x38	**0>**	0x6F	**w@**	0xA6	**1**
0x39	**0>=**	0x70	**<w@**	0xA7	**2**
0x3A	**<**	0x71	**c@**	0xA8	**3**
0x3B	**>**	0x72	**!**	0xA9	**bl**
0x3C	**=**	0x73	**l!**	0xAA	**bs**
0x3D	**<>**	0x74	**w!**	0xAB	**bell**
0x3E	**u>**	0x75	**c!**	0xAC	**bounds**
0x3F	**u<=**	0x76	**2@**	0xAD	**here**
0x40	**u<**	0x77	**2!**	0xAE	**aligned**
0x41	**u>=**	0x78	**move**	0xAF	**wbsplit**
0x42	**>=**	0x79	**fill**	0xB0	**bwjoin**
0x43	**<=**	0x7A	**comp**	0xB1	**b(<mark)**

0xB2	**b(>resolve)**
0xB3*	**set-token-table**
0xB4*	**set-table**
0xB5	**new-token**
0xB6	**named-token**
0xB7	**b(:)**
0xB8	**b(value)**
0xB9	**b(variable)**
0xBA	**b(constant)**
0xBB	**b(create)**
0xBC	**b(defer)**
0xBD	**b(buffer:)**
0xBE	**b(field)**
0xBF*	**b(code)**
0xC0	**instance**
0xC1	Reserved
0xC2	**b(;)**
0xC3	**b(to)**
0xC4	**b(case)**
0xC5	**b(endcase)**
0xC6	**b(endof)**
0xC7	**#**
0xC8	**#s**
0xC9	**#>**
0xCA	**external-token**
0xCB	**$find**
0xCC	**offset16**
0xCD	**evaluate**
0xCE–	
0xCF	Reserved
0xD0	**c,**
0xD1	**w,**
0xD2	**l,**
0xD3	**,**
0xD4	**um***
0xD5	**um/mod**
0xD6–	
0xD7	Reserved
0xD8	**d+**
0xD9	**d-**
0xDA	**get-token**
0xDB	**set-token**
0xDC	**state**
0xDD	**compile,**
0xDE	**behavior**
0xDF–	
0xEF	Reserved
0xF0	**start0**
0xF1	**start1**
0xF2	**start2**
0xF3	**start4**
0xF4–	
0xFB	Reserved
0xFC	**ferror**
0xFD	**version1**
0xFE*	**4-byte-id**
0xFF	**end1**
0x100	Reserved
0x101*	**dma-alloc**
0x102	**my-address**
0x103	**my-space**
0x104*	**memmap**
0x105	**free-virtual**
0x106*	**>physical**
0x107–	
0x10E	Reserved
0x10F*	**my-params**
0x110	**property**
0x111	**encode-int**
0x112	**encode+**
0x113	**encode-phys**
0x114	**encode-string**
0x115	**encode-bytes**
0x116	**reg**
0x117†	**intr**
0x118*	**driver**
0x119	**model**
0x11A	**device-type**
0x11B	**parse-2int**
0x11C	**is-install**
0x11D	**is-remove**
0x11E	**is-selftest**
0x11F	**new-device**
0x120	**diagnostic-mode?**
0x121‡	**display-status**
0x122	**memory-test-suite**
0x123*	**group-code**
0x124	**mask**
0x125	**get-msecs**
0x126	**ms**
0x127	**finish-device**
0x128	**decode-phys**
0x129–	
0x12F	Reserved
0x130	**map-low**
0x131	**sbus-intr>cpu**
0x131–	
0x14F	Reserved
0x150	**#lines**
0x151	**#columns**
0x152	**line#**
0x153	**column#**
0x154	**inverse?**
0x155	**inverse-screen?**
0x156*	**frame-buffer-busy?**
0x157	**draw-character**
0x158	**reset-screen**
0x159	**toggle-cursor**
0x15A	**erase-screen**
0x15B	**blink-screen**
0x15C	**invert-screen**
0x15D	**insert-characters**
0x15E	**delete-characters**
0x15F	**insert-lines**
0x160	**delete-lines**
0x161	**draw-logo**
0x162	**frame-buffer-adr**
0x163	**screen-height**
0x164	**screen-width**
0x165	**window-top**
0x166	**window-left**
0x167–	
0x169	Reserved
0x16A	**default-font**
0x16B	**set-font**
0x16C	**char-height**
0x16D	**char-width**
0x16E	**>font**
0x16F	**fontbytes**
0x170–	
0x17C*	**fb1-** routines
0x17D–	
0x17F	Reserved
0x180	**fb8-draw-character**
0x181	**fb8-reset-screen**
0x182	**fb8-toggle-cursor**
0x183	**fb8-erase-screen**
0x184	**fb8-blink-screen**
0x185	**fb8-invert-screen**
0x186	**fb8-insert-characters**
0x187	**fb8-delete-characters**
0x188	**fb8-insert-lines**
0x189	**fb8-delete-lines**
0x18A	**fb8-draw-logo**
0x18B	**fb8-install**
0x18C–	
0x18F	Reserved
0x190–	
0x196†	VME-bus support
0x197–	
0x19F	Reserved
0x1A0*	**return-buffer**
0x1A1*	**xmit-packet**
0x1A2*	**poll-packet**
0x1A3	Reserved
0x1A4	**mac-address**
0x1A5–	
0x200	Reserved
0x201	**device-name**
0x202	**my-args**
0x203	**my-self**
0x204	**find-package**
0x205	**open-package**
0x206	**close-package**
0x207	**find-method**
0x208	**call-package**
0x209	**$call-parent**
0x20A	**my-parent**
0x20B	**ihandle>phandle**
0x20C	Reserved
0x20D	**my-unit**
0x20E	**$call-method**
0x20F	**$open-package**
0x210*	**processor-type**
0x211*	**firmware-version**
0x212*	**fcode-version**
0x213	**alarm**
0x214	**(is-user-word)**
0x215	**suspend-fcode**
0x216	**abort**
0x217	**catch**
0x218	**throw**

0x219	**user-abort**	0x226	**lwflip**	0x239*	**probe-virtual**
0x21A	**get-my-property**	0x227	**lbflip**	0x23A	Reserved
0x21B	**decode-int**	0x228	**lbflips**	0x23B	**child**
0x21C	**decode-string**	0x229*	**adr-mask**	0x23C	**peer**
0x21D	**get-inherited-property**	0x22A-0x22F	Reserved	0x23D	**next-property**
0x21E	**delete-property**	0x230	**rb@**	0x23E	**byte-load**
0x21F	**get-package-property**	0x231	**rb!**	0x23F	**set-args**
0x220	**cpeek**	0x232	**rw@**	0x240	**left-parse-string**
0x221	**wpeek**	0x233	**rw!**	0x241-0x5FF	Reserved
0x222	**lpeek**	0x234	**rl@**	0x600-0x7FF	Vendor FCodes
0x223	**cpoke**	0x235	**rl!**	0x800-0xFFF	Local FCodes
0x224	**wpoke**	0x236	**wbflips**		
0x225	**lpoke**	0x237	**lwflips**		
		0x238*	**probe**		

_____

* These are historical FCodes.

† These are obsolete FCodes.

Annex H
Historical notes

H.4 New FCodes and methods

Most pre-Open Firmware systems do not implement the following FCodes and *methods*:

FCode#	Name	Comments
0xC7	**#**	Not the same as old **#** (now called **u#**).
0xC9	**#>**	Not the same as old **#>** (now called **u#>**).
0xC8	**#s**	Not the same as old **#s** (now called **u#s**).
0xDE	**behavior**	
0x23E	**byte-load**	On pre-Open Firmware, **" byte-load" $find** could be used.
0xDD	**compile,**	On pre-Open Firmware, **" (compile)" $find** could be used.
0x128	**decode-phys**	
0xDA	**get-token**	
method	**encode-unit**	
0x227	**lbflip**	
0x228	**lbflips**	
0x2269	**lwflip**	On pre-Open Firmware, the "**wflip**" tokenizer macro was used.
0x23D	**next-property**	
0x23F	**set-args**	On pre-Open Firmware, **" set-args" $find** could be used.
0xDB	**set-token**	
0xDC	**state**	On pre-Open Firmware, **" state" $find** could be used.
0x89	**unloop**	

H.6 New user interface commands

Most pre-Open Firmware systems do not implement the following user interface commands.

apply	(... "method-name< >device-specifier< >" -- ???)	Execute named method in the specified package.
char	("text< >" -- char)	Generate numeric code for next character from input buffer.
[char]	(C: [text< >] --) (-- char)	Generate numeric code for next character from input buffer.
~~**close-dev**~~	~~(ihandle --)~~	~~Close device and all of its parents.~~

$create	(E: -- a-addr) (name-str name-len --)	Call **create**; new name specified by *name string*.
environment?	(str len -- false | value true)	Return system information based on input keyword.
fm/mod	(d n -- rem quot)	Divide *d* by *n*.
noshowstack	(--)	Turn off **showstack** (automatic stack display).
parse	(delim "text<delim>" -- str len)	Parse text from the input buffer, delimited by *delim*.
parse-word	("text< >" -- str len)	Parse text from the input buffer, delimited by white space.
~~**(patch)**~~	~~(new-n1 num1? old-n2 num2? xt --)~~	~~Change contents of command indicated by *xt*.~~
postpone	(C: [old-name< >] --) (... -- ???)	Delay execution of the immediately following command.
recurse	(... -- ???)	Compile recursive call to the command being compiled.
s"	([text<">] -- text-str text-len)	Gather the immediately following string.
s>d	(n1 -- d1)	Convert a number to a double number.
sm/rem	(d n -- rem quot)	Divide *d* by *n*, symmetric division.
~~**status**~~	~~(--)~~	~~**defer** word that can be used to modify the user interface prompt.~~

H.7 FCode name changes

The following FCodes names have changed from their pre-Open Firmware versions for clarity and consistency. While this can affect the *tokenizer* and/or user interface behavior, the actual behavior of the function associated with that FCode number has not changed. Existing (already-tokenized) FCode programs that use these FCodes will be unaffected.

Items marked with a * have retained the old name, as a synonym.

Old Name	New Name
#	**u#**
#>	**u#>**
#s	**u#s**
<<	**lshift** *
>>	**rshift** *
attribute	**property**
b(is)	**b(to)**
/c*	**chars** *
ca1+	**char+** *
decode-2int	**parse-2int**
delete-attribute	**delete-property**
eval	**evaluate** *
flip	**wbflip**
get-inherited-attribute	**get-inherited-property**
get-my-attribute	**get-my-property**
get-package-attribute	**get-package-property**

Old Name	New Name
is	**to**
lflips	**lwflips**
map-sbus	**map-low**
na1+	**cell+** *
/n*	**cells** *
not	**invert** *
u*x	**um***
version	**fcode-revision**
wflips	**wbflips**
x+	**d+**
x-	**d-**
xdr+	**encode+**
xdrbytes	**encode-bytes**
xdrint	**encode-int**
xdrphys	**encode-phys**
xdrstring	**encode-string**
xdrtoint	**decode-int**
xdrtostring	**decode-string**
xu/mod	**um/mod**

H.8 User interface name changes

The following user interface command names have changed from their pre-Open Firmware versions, with no change in behavior.

Old name	New name
.attributes	**.properties**
cd	**dev**
reset	**reset-all**
~~**select-dev**~~	~~**open-dev**~~
~~**unselect-dev**~~	~~**device-end**~~