x. The optional Cross-compiler word set

x.1 Introduction

The purpose of this optional wordset is to facilitate writing programs that may be compiled to run on targets other than the system performing the compilation. See “cross-compiler”, **Section x.2.1 Definitions of Terms**.

It is acknowledged that most cross-compiled programs have inherent dependencies upon hardware configuration and other issues that render true program portability impossible to achieve in this context. The intent of this wordset is to facilitate the porting of programs from one target to another by standardizing aspects of the relationship between host and target, as well as means of describing a target configuration.

x.2 Additional terms and notation

x.2.1 Definitions of terms

active section: the current section of the current section type.

allocation pointer: The address of the next available location in each defined memory section.

CData (constant data space): Data space that is read-only during execution of a target program. CData may also contain executable code.

compiling words: Words that modify a Forth dictionary by emplacing code or data there or performing a static memory allocation.

cross-compiler: A compiler whose target is assumed to be a different computer from the one on which the compiler is operating.

host: The computer on which a cross-compiler is run. It may or may not be or resemble the computer on which the compiled program will run.

IData (initialized data space): Space in RAM that is initialized by implementation-dependent means before a target program begins to run.

ROM: “Read-only-memory,” memory which may exist in the target and which cannot be changed by a running program. ROM may be used to contain code space and start-up initialization data. This term also includes memory that is writable only by special procedures, such as flash memory or EEPROM.

RAM: “Random-access memory,” memory in the target which may be read or written by a running program. Data space resides in RAM.

scope: The logical space in which a word is visible or can be executed.

searching words: Words that search a Forth dictionary.

section: A contiguous region of memory designated for particular treatment (according to its type, CData, IData, or UData) by the cross compiler.

target: the computer for which a cross-compiled program is being prepared. It may or may not resemble the computer on which the compiler executes.

target address: an address in target address space.

target dictionary: the image of the target program constructed by the host for execution on the target. It may include both executable code and data objects defined in initialized data space. The physical location and structure of the target dictionary is implementation-defined.

UData (uninitialized data space): Space in RAM that is not initialized to any specific value when a target program begins to run.

x.3 Additional usage requirements

x.3.1 Scopes

Scopes are used to distinguish specialized cross-compiler words from their counterparts in an underlying Forth system supporting the cross-compiler, or from target behaviors that are inappropriate in the host environment. Scopes may be (but are not required to be) implemented using the Search Order wordset; if this is the case, a scope may consist of a search order consisting of multiple wordlists.

A scope is selected by use of one of the following "scope selector" words: **HOST**, **INTERPRETER**, **COMPILER**, **TARGET**. Unless otherwise specified herein, a scope remains in effect until explicitly changed. All scope selector words must be executable in all scopes. Scope selector words are host-executable words and may not be referenced in target colon definitions.

An ambiguous condition exists if a program changes scope within a colon definition.

x.3.1.1 HOST scope

The **HOST** scope provides access to the underlying Forth system upon which the cross-compiler is built. The **HOST** scope shall include, at least, the CORE wordset. It may include additional wordsets and implementation-defined extensions for the purpose of cross-compiler support.

Words in the **HOST** scope are executable only on the host system in **HOST** or **INTERPRETER** scopes; **HOST** defining words construct additional host definitions and data objects; and **HOST** memory allocation words affect the host dictionary only.

x.3.1.2 INTERPRETER scope

The **INTERPRETER** scope enables the definition of words that may be executed on the host in **TARGET** scope to construct and manipulate the target dictionary. Generally speaking, this comprises cross-compiler versions of all defining words (e.g., **: ; CREATE VARIABLE**) and memory allocation and access words (e.g., **@**, **!**, **ALLOT**, **,**). Words in this scope behave analogously to their CORE counterparts except that they construct *target* definitions, allocate *target* memory, and access the *target's* initialized data space.

Cross-compiler-specific defining words such as **CVARIABLE** and **BUFFER:** are defined in **INTERPRETER** scope, as are any application-specific defining words or memory-allocation words. The portion of a defining word following **DOES>** is considered a target definition, and is subject to all specifications in this document that apply to target definitions.

Words defined in the **INTERPRETER** scope may be executed in **INTERPRETER** or **TARGET** scopes. **INTERPRETER** words may be referenced in other **INTERPRETER** definitions, but may not be referenced in **TARGET** colon definitions. They are not available in **HOST** or **COMPILER** scopes.

An ambiguous condition exists if a word executed on the host attempts to access the target's uninitialized data space.

x.3.1.3 COMPILER scope

The **COMPILER** scope provides words that may be executed within a target colon definition (between the **:** or **:NONAME** and terminating **;**) to modify the target dictionary being constructed. Generally speaking, this comprises all flow-of-control structure words (e.g., **DO**, **IF**, etc.), **LITERAL**, and other words that are "immediate" in normal Forth usage.

Words defined in **COMPILER** scope will be executed automatically by the host when encountered in a target definition. The word **IMMEDIATE** (6.1.1710) may not be used with **COMPILER** words.

An ambiguous condition exists if a **COMPILER** word is referenced other than within a target colon definition while compiling.

x.3.1.4 TARGET scope

The **TARGET** scope contains all definitions that will be executable on the target system, including both colon definitions and data objects. An ambiguous condition results from an attempt to execute a **TARGET** definition on the host computer.

The following defining words have special behaviors when encountered in **TARGET** scope:

2VARIABLE, **:**, **:NONAME**, **CREATE**, **CVARIABLE**, **VALUE**, **VARIABLE**

Except for those words with additional requirements itemized in the glossaries below, the difference is that they create target definitions; in all other respects they behave analogously to their non-cross-compiling counterparts.

x.3.1.5 Use of IMMEDIATE

Since the primary function of **IMMEDIATE** (6.1.1710) is superceded in the cross compiler by the **COMPILER** scope, it is an ambiguous condition to use **IMMEDIATE** in **INTERPRETER**, **COMPILER**, or **TARGET** scopes.

x.3.2 Data space management

Target memory space can be divided into multiple sections of three types: CData, IData, and UData. These are discussed below.

x.3.2.1 Types of memory

There are three kinds of target memory:

CData contains data structures that are specified at compile time, and are read-only during execution of a target program. CData may reside in PROM or other read-only media. CData may also contain executable code, a program image, and/or the initialization tables for IData.

IData is initialized data space. IData may be allocated, read and written during the compilation process, and is also read/write during execution of the target program. The host maintains an image of Idat, built during compilation, which will be recorded with the target program in an implementation-dependent fashion. It is the responsibility of the target system to construct and initialize this space at run-time prior to launching any user programs.

UData is target data space whose initial content is undetermined at run-time. UData may be allocated, but may not be read or written during compilation. The target system is not required to perform any specific initialization of UData, and a program may not make any assumptions regarding its contents before storing into it. An ambiguous condition exists if the target program reads UData before writing to it.

Data objects reside in CData, IData or UData. The following standard data object defining words are specified as to which kind of space they reside in:

Table x.1: Data space assignments

Object	Data location
2VARIABLE	UData or IData, controlled by **VARIABLES** switch
CVARIABLE	UData or IData, controlled by **VARIABLES** switch
VARIABLE	UData or IData, controlled by **VARIABLES** switch
VALUE	IData

The default location for data in a user-defined defining word is IData, however the definition may explicitly select the memory space type in which its data space is allocated.

The words **CDATA**, **IDATA**, and **UDATA** select a current memory space type. The specific bounds of this space depend upon the currently selected section of the current memory type. **INTERPRETER** words that manage data space always act on the active section.

x.3.2.2 Memory sections

A program may define one or more sections of each section type, specifying the address range for each. The current section type controls data defining words, memory allocation, and memory accesses of various types, both during compilation and during interactive testing.

A section definition includes its starting and ending addresses, its type, and an allocation pointer. The allocation pointer for each section gives the address of the next available location in that section. When a section is initially defined, its allocation pointer points to its starting address.

The current state of all section definitions, including the current section of each section type and the current section type, but not the contents of the sections, is called the "section context." It may be saved and restored by **SAVE-SECTIONS** and **RESTORE-SECTIONS**.

x.3.2.2.1 Defining sections

At least one *instance* of each section type must be defined, with upper and lower target address boundaries, before it is used. Address ranges for instances of the same section type may not overlap. The syntax for defining a memory section is:

```
<low address> <high address> [<type>] SECTION <name>
```

This defines a section of type *type* named *name* with the address range specified.

Sections must be defined in **INTERPRETER** scope.

An instance becomes the *current section* of its type when its name is invoked. The compiler will maintain a set of allocation pointers for each section of each type. Only one section of each type is current at any time.

This standard does not attempt to guarantee the availability of memory at any specified target address range; the actual addresses available are intrinsically dependent upon the target hardware. An ambiguous condition exists if a program specifies a section in memory space incompatible with the target hardware.

If the target system will place IData initialization tables and executable code (or its program image) in CData, the programmer is responsible for configuring sufficient CData space to contain these things in addition to explicitly defined CData objects.

x.3.2.2.2 Memory access

The following memory access words have special behaviors when executed in **TARGET** scope or in an **INTERPRETER** definition:

! , @ BLANK C! C, C@ CMOVE CMOVE> CMOVEC ERASE FILL MOVE

Except for those words with additional requirements itemized in the glossaries below, the difference is that these words accept addresses in target address space. An ambiguous condition exists if that address does not fall within a defined CData or IData section when these words are executed by the host.

The following memory allocation and address management words have special behaviors when executed in **TARGET** scope or in an **INTERPRETER** definition:

, (comma) **ALIGN ALIGNED ALLOT C, HERE UNUSED**

Except for those words with additional requirements itemized in the glossaries below, the difference is that these words accept addresses in target address space. An ambiguous condition exists if that address does not fall within a defined CData or IData section when these words are executed by the host.

x.3.2.6 Environmental queries

A Cross-compiler supports two environments, the host and target. Table x.2 identifies which of the environmental query strings defined in **3.2.6 Environmental queries** applies to the host and which to the target.

Table x.2 – Environmental Query Strings

String	Applies to:
/COUNTED-STRING	target
/HOLD	target
/PAD	target
ADDRESS-UNIT-BITS	target
CORE	true if complete core word set present in the programming environment presented by the host and target as described in section x.5.1.
CORE-EXT	true if core extensions word set present in the programming environment presented by the host and target as described in section x.5.1.
CROSS	true if complete cross-compiler word set present.
CROSS-EXT	true if complete cross-compiler extension word set present.
FLOORED	target
MAX-CHAR	target
MAX-D	target
MAX-N	target
MAX-U	target
MAX-UD	target
RETURN-STACK-CELLS	target
STACK-CELLS	target

x.3.3.3.2 Contiguous regions

A cross compiler guarantees that a region of data space allocated using **ALLOT**, **,** (comma), **C,** (c-comma), and **ALIGN** shall be contiguous with the last region allocated with one of the above words within the same section of the same section type. The data-space pointer **HERE** always identifies the beginning of the next data-space region in the active section to be allocated. As successive allocations are made, the data-space pointer increases. A program may perform address arithmetic within contiguously allocated regions. The last region of data space allocated in the

current section using the above operators may be released by allocating a corresponding negatively-sized region using **ALLOT**.

CREATE establishes the beginning of a contiguous region of data space in the active section, whose starting address is returned by the **CREATE**d definition. This region is terminated by defining the next object in that section type.

The contiguity of regions allocated by **BUFFER:** and **RESERVE** is unspecified.

x.3.4 Effects of scopes on data object defining words

Defining words other than **:** (colon) are used to build data structures with characteristic behaviors.

Normally, the primary role of a cross-compiler is building definitions and data structures for the target system; therefore, the dominant use of defining words is in the **TARGET** scope while interpreting. A program may also build data objects in **HOST** that may be used in all scopes *except* **TARGET**; such objects might, for example, be used to control the compiling process.

Target data objects fall into three classes:

- *CData or IData objects* in initialized data memory—e.g., words defined by **CREATE**, etc., including most user-defined words made with **CREATE … DOES>**.
- *UData objects* in uninitialized data memory—e.g., words defined by the use of **VARIABLE**, **BUFFER:**, etc.
- *Constants*—words defined by **VALUE**, **CONSTANT** or **2CONSTANT**.

Unlike target colon definitions, target data objects defined with standard defining words may be invoked in **TARGET** scope while interpreting. Words defined by **CREATE**, **VARIABLE**, **2VARIABLE**, **CVARIABLE**, and **BUFFER:** will return the address of their target data space. A target data object defined with custom behaviors using **CREATE … DOES>** may *not* be invoked while interpreting, because its behavior is not executable on the host. To obtain the address of the target data space of such an object, you may use the form:

```
' <name> >BODY.
```

Target data objects defined by **VALUE**, **CONSTANT**, or **2CONSTANT** return their values when executed in **TARGET** scope. Target **VALUE**s may be altered interpretively in **TARGET** scope using **TO**. **CONSTANT**s and **2CONSTANT**s may not be altered at any time once they have been defined.

IData objects may be given compiled, initial values with **,** (comma) and **C,** (c-comma), and you may also use **@** and **!** (etc.) with them interpretively on the host. However, there is no way to initialize UData objects at compile time, and an ambiguous condition results from any attempt to access UData objects from the host.

x.3.5 Ambiguous conditions

– a word executed on the host attempts to access the target's uninitialized data space.

– the current scope is changed within a colon definition.

– a **COMPILER** word is referenced other than within a target colon definition while in compiling state.

– the word **IMMEDIATE** (6.1.1710) is used in **COMPILER**, **INTERPRETER**, or **TARGET** scopes.

– an attempt to execute a **TARGET** definition on the host computer.

– a program specifies a section in memory space incompatible with the target hardware.

– *addr* passed to **ORG** is not within the active section.

– a target program attempts to write to CData.

– a program attempts to define a **VARIABLE**, **CVARIABLE**, or **2VARIABLE** without previously executing **VARIABLES**.

– A target program reads a UData location before it has been written.

– **EQU** is executed in **HOST** scope.

x.4 Additional documentation requirements

x.4.1 System documentation

x.4.1.1 Implementation-defined options

x.4.1.1.1 Host options

The implementation-defined items in the following list represent host characteristics and choices left to the discretion of the implementor, provided that the requirements of this Standard are met. A system shall document the values for, or behaviors of, each item.

– conditions under which control characters match a space delimiter (**3.4.1.1 Delimiters**);
– input line terminator (**3.2.4.1 User input device**);
– maximum size of a parsed string (**3.4.1 Parsing**);
– maximum size of a definition name, in characters (**3.3.1.2 Definition names**);
– maximum string length for **6.1.1345 ENVIRONMENT?**, in characters;
– methods of dictionary compilation (**3.3 The Forth dictionary**);
– size of buffer at **6.1.2450 WORD** (**3.3.3.6 Other transient regions**);
– system case-sensitivity characteristics (**3.4.2 Finding definition names**);
– system prompt (**3.4 The Forth text interpreter**, **6.1.2050 QUIT**);
– values of **6.1.2250 STATE** when true;
– whether the current definition can be found after **6.1.1250 DOES>** (**6.1.0450 :**).

x.4.1.1.2 Target options

The implementation-defined items in the following list represent target characteristics and choices left to the discretion of the implementor, provided that the requirements of this Standard are met. A system shall document the values for, or behaviors of, each item.

– aligned address requirements (**3.1.3.3 Addresses**);
– behavior of **6.1.1320 EMIT** for non-graphic characters;
– character editing of **6.1.0695 ACCEPT** and **6.2.1390 EXPECT**;
– character set (**3.1.2 Character types**, **6.1.1320 EMIT**, **6.1.1750 KEY**);
– character-aligned address requirements (**3.1.3.3 Addresses**);
– character-set-extensions matching characteristics (**3.4.2 Finding definition names**);
– format of the control-flow stack (**3.2.3.2 Control-flow stack**);
– conversion of digits larger than decimal thirty-five (**3.2.1.2 Digit conversion**);
– display after input terminates in **6.1.0695 ACCEPT** and **6.2.1390 EXPECT**;
– exception abort sequence (as in **6.1.0680 ABORT"**);
– input line terminator (**3.2.4.1 User input device**);
– maximum size of a counted string, in characters (**3.1.3.4 Counted strings**, **6.1.2450 WORD**);
– maximum size of a parsed string (**3.4.1 Parsing**);
– method of selecting **3.2.4.1 User input device**;
– method of selecting **3.2.4.2 User output device**;

- number of bits in one address unit (**3.1.3.3 Addresses**);
- number representation and arithmetic (**3.2.1.1 Internal number representation**);
- ranges for *n*, *+n*, *u*, *d*, *+d*, and *ud* (**3.1.3 Single-cell types**, **3.1.4 Cell-pair types**);
- read-only data-space regions (**3.3.3 Data space**);
- size of one cell in address units (**3.1.3 Single-cell types**);
- size of one character in address units (**3.1.2 Character types**);
- size of the keyboard terminal input buffer (**3.3.3.5 Input buffers**);
- size of the pictured numeric output string buffer (**3.3.3.6 Other transient regions**);
- size of the scratch area whose address is returned by **6.2.2000 PAD** (**3.3.3.6 Other transient regions**);
- type of division rounding (**3.2.2.1 Integer division**, **6.1.0100 */**, **6.1.0110 */MOD**, **6.1.0230 /**, **6.1.0240 /MOD**, **6.1.1890 MOD**);
- values of **6.1.2250 STATE** when true;
- values returned after arithmetic overflow (**3.2.2.2 Other integer operations**);
- whether the current definition can be found after **6.1.1250 DOES>** (**6.1.0450 :**).

x.4.1.2 Ambiguous conditions

- system response to each of the conditions listed in Section x.3.5.

x.4.1.3 Other system documentation

A Cross-compiler shall provide the following information:

- list of non-standard target words using **6.2.2000 PAD** (**3.3.3.6 Other transient regions**);
- target's terminal facilities available;
- target program data space available, in address units;
- target return stack space available, in cells;
- target stack space available, in cells;
- target CData space required, in address units.

x.4.2 Program documentation

- memory requirements for CData, IData, and UData.

x.5 Compliance and labeling

x.5.1 ANS Forth cross-compilers

A system may be described as "An ANS Forth Cross-Compiler" when the programming environment presented by the host and target in combination complies with all the system requirements in **3. Usage requirements** and **4. System documentation** not superceded by requirements in this section X, and all additional requirements in this section X. The following sections document specific requirements for the host and target considered individually.

x.5.1.1 Host system requirements

The programming environment presented by the host system shall include at a minimum all compiling, defining, and searching words defined in section **x.6.1 Cross compiler words**.

The host must also provide in its **HOST** scope all CORE words with their standard meanings. Those CORE words not given extended meanings in section **x.6.1 Cross-compiler words** shall be available for use in **INTERPRETER** scope.

x.5.1.2 Target system requirements

The target system may, but is not required to, provide the following compiling, defining, and searching words:

```
'  (  ,  :  ;  >IN  ALLOT  ALIGN  C,  CHAR  CONSTANT
CREATE  FIND  ENVIRONMENT?  EVALUATE  IMMEDIATE  QUIT
SOURCE  STATE  VARIABLE  WORD  [  ]
HOST  INTERPRETER  COMPILER  TARGET
CDATA  IDATA  UDATA
```

x.5.1.3 Extension labeling requirements

The phrase “Providing *name*(*s*) from the Cross-compiler Extensions word set” shall be appended to the label of any Cross Compiler that provides portions of the Cross-compiler Extensions word set.

The phrase “Providing the Cross-compiler Extensions word set” shall be appended to the label of any Cross Compiler that provides all of the Cross-compiler and Cross-compiler Extensions word sets.

The target system is not required to provide the following Cross-compiler Extensions words:

```
2VARIABLE  BUFFER:  CVARIABLE  EQU  ORG  VALUE
```

Additional optional words and wordsets may be provided, however the Cross-compiler must specify what features in any such additions are supported by the host, by the target, or both.

x.5.2 ANS Forth programs

The phrase “Requiring the Cross-compiler word set” shall be appended to the label of Standard Programs that require the system to provide the Cross-compiler word set.

The phrase “Requiring *name*(*s*) from the Cross-compiler Extensions word set” shall be appended to the label of Standard Programs that require the system to provide portions of the Cross-compiler Extensions word set.

The phrase “Requiring the Cross-compiler Extensions word set” shall be appended to the label of Standard Programs that require the system to provide all of the Cross-compiler and Cross-compiler Extensions word sets.

x.6 Glossary

x.6.1 Cross-compiler words

x.6.1.0010 ! “store” CROSS

(*x a-addr* —)

Extend the definition of **!** when executed in **TARGET** scope or in an **INTERPRETER** definition to store into the host’s image of the memory section (CData or IData only) containing *a-addr*. An ambiguous condition exists if *addr* doesn’t reside in a defined CData or IData section.

x.6.1.0070 ' “tick” CROSS

(“<*spaces*>*name*” — *xt*)

Extend the definition of **'** when executed in **TARGET** scope or in an **INTERPRETER** definition to search only **TARGET** definitions and return a target-executable *xt*.

x.6.1.0150 **,** "comma" CROSS

(x —)

Extend the definition of **,** when executed in **TARGET** scope or in an **INTERPRETER** definition to compile *x* at the next available location in the active section (CData or IData only).

x.6.1.0650 **@** "fetch" CROSS

(*a-addr — x*)

Extend the definition of **@** when executed in **TARGET** scope or in an **INTERPRETER** definition to fetch from the host's image of the memory section (CData or IData) containing *a-addr*.

x.6.1.0705 **ALIGN** CROSS

(—)

Extend the definition of **ALIGN** when executed in **TARGET** scope or in an **INTERPRETER** definition to force the space allocation pointer for the active section to be aligned according to requirements of the target system.

See: **3.3.3.1 Address alignment**.

x.6.1.0706 **ALIGNED** CROSS

(*addr — a-addr*)

Extend the definition of **ALIGN** when executed in **TARGET** scope or in an **INTERPRETER** definition to align *addr* according to requirements of the target system.

See: **3.3.3.1 Address alignment**.

x.6.1.0710 **ALLOT** CROSS

(n —)

Extend the definition of **ALLOT** when executed in **TARGET** scope or in an **INTERPRETER** definition to allocate *n* bytes at the next available location in the active section.

x.6.1.0850 **C!** "c-store" CROSS

(*char c-addr —*)

Extend the definition of **C!** when executed in **TARGET** scope or in an **INTERPRETER** definition to store into the host's image of the section (CData or IData only) containing *c-addr*. An ambiguous condition exists if *c-addr* doesn't reside in a defined CData or IData section.

x.6.1.0860 C, "c-comma" CROSS

(char —)

Extend the definition of **C,** when executed in **TARGET** scope or in an **INTERPRETER** definition to compile *char* at next available location in the active section (CData or IData only).

x.6.1.0870 C@ "c-fetch" CROSS

(*c-addr – char*)

Extend the definition of **C@** when executed in **TARGET** scope or in an **INTERPRETER** definition to fetch a character from the host's image of the memory section (CData or IData only) containing *c-addr*. An ambiguous condition exists if *c-addr* doesn't reside in a defined CData or IData section.

x.6.1.nnnn CDATA "c-data" CROSS

(—)

Select CData as the current section type. This word may be executed in **TARGET** scope, or used in **INTERPRETER** definitions.

x.6.1.0880 CELL+ "cell-plus" CROSS

($a\text{-}addr_1$ — $a\text{-}addr_2$)

Extend the definition of **CELL+** when executed in **TARGET** scope or in an **INTERPRETER** definition to apply the size of a *target* cell.

See: **3.3.3.1 Address alignment**.

x.6.1.0890 CELLS CROSS

(n_1 — n_2)

Extend the definition of **CELLS** when executed in **TARGET** scope or in an **INTERPRETER** definition to apply the size a *target* cells.

x.6.1.0897 CHAR+ "char-plus" CROSS

($c\text{-}addr_1$ — $c\text{-}addr_2$)

Extend the definition of **CHAR+** when executed in **TARGET** scope or in an **INTERPRETER** definition to apply the size of a *target* character.

See: **3.3.3.1 Address alignment**.

x.6.1.0898 CHARS “chars” CROSS

(n_1 — n_2)

Extend the definition of **CHARS** when executed in **TARGET** scope or in an **INTERPRETER** definition to apply the size of a *target* character.

x.6.1.nnnn COMPILER CROSS

(—)

Select the scope in which words are defined that will be executed when encountered within **TARGET** colon definitions at compile time, to modify the target dictionary entry being constructed.

See: **x.3.1.3 COMPILER scope**.

x.6.1.1000 CREATE CROSS

(“<spaces>name” —)

Extend the definition of **CREATE** when executed in **TARGET** scope to define a named reference to the next available location in the active section. Does not allocate any data space.

When encountered while compiling a word being defined in **INTERPRETER** scope, it compiles a reference to the version of **CREATE** described above. This is the appropriate usage for creating new target defining words, used in combination with **x.6.1.1250 DOES>**.

x.6.1.1250 DOES> “does” CROSS

(—)

When encountered while compiling a word being defined in **INTERPRETER** scope, extend the definition of **DOES>** to cause the compiler to use only **TARGET** and **COMPILER** words until the end of the definition in which **DOES>** is found.

x.6.1.1540 FILL CROSS

(c-addr u char —)

Extend the definition of **FILL** when executed in **TARGET** scope or in an **INTERPRETER** definition to write to the host’s image of CData or IData.

x.6.1.1650 **HERE** CROSS

(— addr)

Extend the definition of **HERE** when executed in **TARGET** scope or in an **INTERPRETER** definition to return the address of the next available location in the active section.

x.6.1.nnnn **HOST** CROSS

(—)

Select the scope that provides access to the words in the underlying Forth system upon which the cross compiler is built.

See: **x.3.1.1 HOST scope**.

x.6.1.nnnn **IDATA** "i-data" CROSS

(—)

Select IData as the current section type. This word may be executed in **TARGET** scope, or used in **INTERPRETER** definitions.

x.6.1.nnnn **INTERPRETER** CROSS

(—)

Select the scope in which words used to construct and manage target definitions and data objects are defined.

See: **x.3.1.2 INTERPRETER scope**.

x.6.1.1900 **MOVE** CROSS

($addr_1$ $addr_2$ *u* —)

Extend the definition of **MOVE** when executed in **TARGET** scope or in an **INTERPRETER** definition to write to the host's image of IData or CData. An ambiguous condition exists if $addr_1$ or $addr_2$ doesn't reside in a defined IData or CData section.

See: **x.6.2.0910 CMOVE**, **x.6.2.0920 CMOVE>**.

x.6.1.2033 **POSTPONE** CROSS

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("<*spaces*>*name*" —)

Extend the semantics of **6.1.2033 POSTPONE** when encountered in a **COMPILER** definition to search for *name* only among **COMPILER** and **TARGET** definitions, and append the compilation behavior of *name* to the current **COMPILER** definition.

An ambiguous condition exists if **POSTPONE** is used in a **TARGET** definition.

See: **3.4.1 Parsing**.

x.6.1.nnnn RESTORE-SECTIONS CROSS

*(n*x n —)*

Restore the entire section context previously saved by **SAVE-SECTIONS**.

x.6.1.nnnn SAVE-SECTIONS CROSS

*(— n*x n)*

Save the entire current section context in an implementation-dependent form.

x.6.1.nnnn SECTION CROSS

($addr_1$ $addr_2$ "<spaces>name" —)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition *name* in the host only, with the execution semantics defined below. The definition of *name* identifies a target memory section of the section type that is current when **SECTION** is executed, whose memory range is $addr_1$ to $addr_2$ (inclusive).

SECTION shall be executed only in **INTERPRETER** scope. Defined sections may be referenced in **INTERPRETER**, **COMPILER**, and **TARGET** scopes.

Immediately following use of **SECTION**, *name* is the current section of that section type, and $addr_1$ is the initial value of the section's allocation pointer.

name Execution: *(—)*

Make *name* the current section of *name*'s section type.

x.6.1.nnnn TARGET CROSS

(—)

Select the scope in which target-executable definitions and data objects are defined.

See: **x.3.1.1 TARGET scope**.

x.6.1.nnnn UDATA "u-data" CROSS

(—)

Select UData as the current section type. This word may be executed in **TARGET** scope, or used in **INTERPRETER** definitions.

x.6.1.2410 VARIABLE CROSS

(“<spaces>name” —)

Extend the definition of **VARIABLE** executed in **TARGET** scope or in an **INTERPRETER** definition to create a definition for *name* in the current section of the section type specified by **x.6.1.nnnn VARIABLES**.

name Execution: (— *a-addr*)

a-addr is the address of the cell reserved by **VARIABLE** when it defined *name*. A program is responsible for initializing the contents.

x.6.1.nnnn VARIABLES CROSS

Configures the cross-compiler to assign data space for **VARIABLE**s (as well as **CVARIABLE**s and **2VARIABLE**s, if they exist) in the current section of the section type (either IData or UData) that is current when **VARIABLES** is executed.

An ambiguous condition exists if a program attempts to define a **VARIABLE**, **CVARIABLE**, or **2VARIABLE** without previously executing **VARIABLES**.

x.6.1.2510 ['] “bracket-tick” CROSS

Compilation: (“<*spaces*>*name*” —)

Extend the compilation semantics of **[']** when encountered in a **TARGET** definition to search only **TARGET** definitions and compile a reference to a target-executable *xt*.

x.6.2 Cross-compiler extension words

x.6.2.nnnn @C “fetch-c” CROSS EXT

(*a-addr* — *x*)

x is the value stored at *a-addr* in CData.

x.6.2.0440 2VARIABLE CROSS EXT

(“<spaces>name” —)

Extend the definition of **2VARIABLE** when executed in **TARGET** scope to create a definition for *name* in the current section of the section type specified by **x.6.1.nnnn VARIABLES**.

name Execution: (— *a-addr*)

a-addr is the address of the first (lowest address) cell of two consecutive cells reserved by **2VARIABLE** when it defined *name*. A program is responsible for initializing the contents.

x.6.2.0780 BLANK CROSS EXT

(*c-addr u* —)

Extend the definition of **BLANK** when encountered while interpreting in **INTERPRETER** or **TARGET** scopes to write to the host's image of CData or IData.

x.6.2.nnnn BUFFER: CROSS EXT

(n "<spaces>name" —)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name*, with the execution semantics defined below. Reserve *n* address units starting at the next location in the current UData section.

BUFFER: shall be executed only in **TARGET** scope. When words defined by **BUFFER:** are referenced on the host while interpreting, their behavior is to return the address of the start of the reserved space.

name Execution: (— *a-addr*)

a-addr is the address of the space reserved by **BUFFER:** when it defined *name*. A program is responsible for initializing the contents. When words defined by **BUFFER:** are referenced on the host while interpreting, their behavior is to return the address of the start of the reserved space.

x.6.2.nnnn C@C "c-fetch-c" CROSS EXT

(*c-addr — char*)

Fetch the character stored at *c-addr* in CData. When the cell size is greater than character size, the unused high-order bits are all zeroes.

x.6.2.0945 COMPILE, "compile-comma" CROSS EXT

Interpretation: Interpretation semantics for this word are undefined.

Execution: (*xt* —)

Extend the definition of **6.2.0945 COMPILE,** when encountered in a **COMPILER** definition, to append the execution semantics of the target definition represented by *xt* to the execution semantics of the current target definition.

An ambiguous condition exists if **COMPILE,** is used in a **TARGET** definition.

x.6.2.0910 CMOVE "c-move" CROSS EXT

($c\text{-}addr_1$ $c\text{-}addr_2$ *u* —)

Extend the definition of **CMOVE** when executed in **TARGET** scope or in an **INTERPRETER** definition to write to the host's image of IData or CData. An ambiguous condition exists if $c\text{-}addr_1$ or $c\text{-}addr_2$ doesn't reside in a defined IData or CData section.

See: **17.1.0910 CMOVE**.

x.6.2.0920 CMOVE> “c-move-up” CROSS EXT

(*c-addr*$_1$ *c-addr*$_2$ *u* —)

Extend the definition of **CMOVE>** when executed in **TARGET** scope or in an **INTERPRETER** definition to write to the host’s image of IData or CData. An ambiguous condition exists if *c-addr*$_1$ or *c-addr*$_2$ doesn’t reside in a defined IData or CData section.

See: **17.1.0920 CMOVE>**.

x.6.2.nnnn CMOVEC “c-move-c” CROSS EXT

(*c-addr*$_1$ *c-addr*$_2$ *u* —)

Copy *u* consecutive characters from *c-addr*$_1$ in CData space to *c-addr*$_2$ in IData or (when used in a target definition) UData space, proceeding character-by-character from lower addresses to higher addresses.

x.6.2.nnnn CVARIABLE CROSS EXT

(“<spaces>name” —)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* in the host only, with the execution semantics defined below. Reserve one character at the next location in the current section of the section type specified by **x.6.1.nnnn VARIABLES**.

name Execution: (— *c-addr*)

c-addr is the address of the space reserved by **CVARIABLE** when it defined *name*. A program is responsible for initializing the contents.

See: **3.3.3.1 Address alignment**.

x.6.2.1350 ERASE CROSS EXT

(*addr u* —)

Extend the definition of **ERASE** when executed in **TARGET** scope or in an **INTERPRETER** definition to write to the host’s image of CData or IData.

x.6.2.nnnn EQU CROSS EXT

(x “<spaces>name” —)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* in the host dictionary only, with the execution semantics defined below.

An ambiguous condition exists if **EQU** is executed in **HOST** scope.

name Execution: *(— x)*

Returns the value *x*. If an **EQU** is referenced inside a target colon definition, its value will be compiled as a literal.

x.6.2.nnnn ORG CROSS EXT

(addr —)

Set the address of the next available location in the active section to *addr*. An ambiguous condition exists if *addr* is not within the active section.

x.6.2.nnnn RESERVE CROSS EXT

(+n — a-addr)

Allocate *+n* bytes of UData, starting at a-*addr*.

x.6.2.2295 TO CORE EXT

Interpretation: (*x* "<*spaces*>*name*" —)

Extend the interpretation semantics of **TO** when executed in **TARGET** scope to affect a **VALUE** defined in **TARGET** scope. An ambiguous condition exists if *name* was not defined by a **TARGET VALUE**.

Compilation: ("<*spaces*>*name*" —)

Extend the compilation semantics of **TO** when invoked in a **TARGET** definition to reference a **TARGET VALUE**. An ambiguous condition exists if *name* was not defined by a **TARGET VALUE**.

Run-time: (*x* —)

Store *x* in *name*.

See: **x.6.2.2405 VALUE**, **13.6.1.2295 TO**.

x.6.2.2395 UNUSED CORE EXT

(— *u*)

Extend the definition of **UNUSED** when encountered in **TARGET** scope to return the amount of space remaining in the active section in address units.

x.6.2.2405 VALUE CROSS EXT

(*x* "<*spaces*>*name*" —)

Extend the definition of **VALUE** when encountered in **TARGET** scope to create a definition for *name* in IData, initialized to *x*.