 Table of Contents

A.11 The optional File-Access word set

Many Forth systems support access to a host file system, and many of these support interpretation of Forth from source text files. The Forth-83 Standard did not address host OS files. Nevertheless, a degree of similarity exists among modern implementations.

For example, files must be opened and closed, created and deleted. Forth file-system implementations differ mostly in the treatment and disposition of the exception codes, and in the format of the file-identification strings. The underlying mechanism for creating file-control blocks might or might not be visible. We have chosen to keep it invisible.

Files must also be read and written. Text files, if supported, must be read and written one line at a time. Interpretation of text files implies that they are somehow integrated into the text interpreter input mechanism. These and other requirements have shaped the file-access extensions word set.

Most of the existing implementations studied use simple English words for common host file functions: OPEN, CLOSE, READ, etc. Although we would have preferred to do likewise, there were so many minor variations in implementation of these words that adopting any particular meaning would have broken much existing code. We have used names with a suffix -FILE for most of these words. We encourage implementors to conform their single-word primitives to the ANS behaviors, and hope that if this is done on a widespread basis we can adopt better definition names in a future standard.

Specific rationales for members of this word set follow.

A.11.3 Additional usage requirements

A.11.3.2 Blocks in files

Many systems reuse file identifiers; when a file is closed, a subsequently opened file may be given the same identifier. If the original file has blocks still in block buffers, these will be incorrectly associated with the newly opened file with disastrous results. The block buffer system must be flushed to avoid this.

A.11.6 Glossary

A.11.6.1.0765 BIN

Some operating systems require that files be opened in a different mode to access their contents as an unstructured stream of binary data rather than as a sequence of lines.

The arguments to READ-FILE and WRITE-FILE are arrays of character storage elements, each element consisting of at least 8 bits. The Technical Committee intends that, in BIN mode, the contents of these storage elements can be written to a file and later read back without alteration. The Technical Committee has declined to address issues regarding the impact of **wide** characters on the File and Block word sets.

A.11.6.1.1010 CREATE-FILE

Typical use:

```
: X .. S" TEST.FTH" R/W CREATE-FILE  ABORT" CREATE-FILE FAILED" ... ;
```

A.11.6.1.1717 INCLUDE-FILE

Here are two implementation alternatives for saving the input source specification in the presence of text file input:

1) Save the file position (as returned by FILE-POSITION) of the beginning of the line being interpreted. To restore the input source specification, seek to that position and re-read the line into the input buffer.

2) Allocate a separate line buffer for each active text input file, using that buffer as the input buffer. This method avoids

the **seek and reread** step, and allows the use of **pseudo-files** such as pipes and other sequential-access-only communication channels.

A.11.6.1.1718 INCLUDED

Typical use: `... S" filename" INCLUDED ...`

A.11.6.1.1970 OPEN-FILE

Typical use:

```
: X .. S" TEST.FTH" R/W OPEN-FILE  ABORT" OPEN-FILE FAILED" ... ;
```

A.11.6.1.2080 READ-FILE

A typical sequential file-processing algorithm might look like:

```
BEGIN                (  )
... READ-FILE THROW  ( length )
?DUP WHILE           ( length )
...                  (  )
REPEAT               (  )
```

In this example, THROW is used to handle (unexpected) exception conditions, which are reported as non-zero values of the ior return value from READ-FILE. End-of-file is reported as a zero value of the **length** return value.

A.11.6.1.2090 READ-LINE

Implementations are allowed to store the line terminator in the memory buffer in order to allow the use of line reading functions provided by host operating systems, some of which store the terminator. Without this provision, a temporary buffer might be needed. The two-character limitation is sufficient for the vast majority of existing operating systems. Implementations on host operating systems whose line terminator sequence is longer than two characters may have to take special action to prevent the storage of more than two terminator characters.

Standard Programs may not depend on the presence of any such terminator sequence in the buffer.

A typical line-oriented sequential file-processing algorithm might look like:

```
BEGIN                (  )
... READ-LINE THROW  ( length not-eof-flag )
WHILE                ( length )
...                  (  )
REPEAT DROP          (  )
```

In this example, THROW is used to handle (unexpected) I/O exception condition, which are reported as non-zero values of the **ior** return value from READ-LINE.

READ-LINE needs a separate end-of-file flag because empty (zero-length) lines are a routine occurrence, so a zero-length line cannot be used to signify end-of-file.

A.11.6.1.2165 S"

Typical use: `... S" ccc" ...`

The interpretation semantics for S" are intended to provide a simple mechanism for entering a string in the interpretation state. Since an implementation may choose to provide only one buffer for interpreted strings, an interpreted string is subject to being overwritten by the next execution of S" in interpretation state. It is intended that no standard words other than S" should in themselves cause the interpreted string to be overwritten. However, since words such as EVALUATE, LOAD, INCLUDE-FILE and INCLUDED can result in the interpretation of arbitrary text, possibly including instances of S", the interpreted string may be invalidated by some uses of these words.

When the possibility of overwriting a string can arise, it is prudent to copy the string to a **safe** buffer allocated by the application.

Programs wishing to parse in the fashion of S" are advised to use PARSE or WORD COUNT instead of S", preventing the overwriting of the interpreted string buffer.

 Table of Contents
 Next Section