
Table of Contents

9. The optional Exception word set

See: A.9 The optional Exception word set

9.1 Introduction

9.2 Additional terms and notation

None.

9.3 Additional usage requirements

See: A.9.3 Additional usage requirements

9.3.1 THROW values

The THROW values {-255...-1} shall be used only as assigned by this Standard. The values {-4095...-256} shall be used only as assigned by a system.

If the File-Access or Memory-Allocation word sets are implemented, it is recommended that the non-zero values of ior lie within the range of system THROW values, as defined above. In an operating-system environment, this can sometimes be accomplished by **biasing** the range of operating-system exception-codes to fall within the THROW range.

Programs shall not define values for use with THROW in the range {-4095...-1}.

9.3.2 Exception frame

An exception frame is the implementation-dependent set of information recording the current execution state necessary for the proper functioning of CATCH and THROW. It often includes the depths of the data stack and return stack.

9.3.3 Exception stack

A stack used for the nesting of exception frames by CATCH and THROW. It may be, but need not be, implemented using the return stack.

9.3.4 Environmental queries

Append table 9.1 to table 3.5.

See: 3.2.6 Environmental queries

Table 9.1 - Environmental query strings

String	Value data type	Constant?	Meaning
EXCEPTION	flag	no	Exception word set present
EXCEPTION-EXT	flag	no	Exception extensions word set present

9.3.5 Possible actions on an ambiguous condition

A system choosing to execute THROW when detecting one of the ambiguous conditions listed in table 9.3.6 shall use the throw code listed there.

See: 3.4.4 Possible actions on an ambiguous condition

Table 9.2 - THROW code assignments

Code	Reserved for
-1	ABORT
-2	ABORT"
-3	stack overflow
-4	stack underflow
-5	return stack overflow
-6	return stack underflow
-7	do-loops nested too deeply during execution
-8	dictionary overflow
-9	invalid memory address
-10	division by zero
-11	result out of range
-12	argument type mismatch
-13	undefined word
-14	interpreting a compile-only word
-15	invalid FORGET
-16	attempt to use zero-length string as a name
-17	pictured numeric output string overflow
-18	parsed string overflow
-19	definition name too long
-20	write to a read-only location
-21	unsupported operation (e.g., AT-XY on a too-dumb terminal)
-22	control structure mismatch

```
-23     address alignment exception
-24     invalid numeric argument
-25     return stack imbalance
-26     loop parameters unavailable
-27     invalid recursion
-28     user interrupt
-29     compiler nesting
-30     obsolescent feature
-31     >BODY used on non-CREATEd definition
-32     invalid name argument (e.g., TO xxx)
-33     block read exception
-34     block write exception
-35     invalid block number
-36     invalid file position
-37     file I/O exception
-38     non-existent file
-39     unexpected end of file
-40     invalid BASE for floating point conversion
-41     loss of precision
-42     floating-point divide by zero
-43     floating-point result out of range
-44     floating-point stack overflow
-45     floating-point stack underflow
-46     floating-point invalid argument
-47     compilation word list deleted
-48     invalid POSTPONE
-49     search-order overflow
-50     search-order underflow
-51     compilation word list changed
-52     control-flow stack overflow
-53     exception stack overflow
-54     floating-point underflow
-55     floating-point unidentified fault
-56     QUIT
-57     exception in sending or receiving a character
-58     [IF], [ELSE], or [THEN] exception
```

9.3.6 Exception handling

There are several methods of coupling CATCH and THROW to other procedural nestings. The usual nestings are the execution of definitions, use of the return stack, use of loops, instantiation of locals and nesting of input sources (i.e., with LOAD, EVALUATE, or INCLUDE-FILE).

When a THROW returns control to a CATCH, the system shall un-nest not only definitions, but also, if present, locals and input source specifications, to return the system to its proper state for continued execution past the CATCH.

See: A.9.3.6 Exception handling

9.4 Additional documentation requirements

9.4.1 System documentation

9.4.1.1 Implementation-defined options

- Values used in the system by 9.6.1.0875 CATCH and 9.6.1.2275 THROW (9.3.1 THROW values, 9.3.5 Possible actions on an ambiguous condition).

9.4.1.2 Ambiguous conditions

- no additional requirements.

9.4.1.3 Other system documentation

- no additional requirements.

9.4.2 Program documentation

- no additional requirements.

9.5 Compliance and labeling

9.5.1 ANS Forth systems

The phrase **Providing the Exception word set** shall be appended to the label of any Standard System that provides all of the Exception word set.

The phrase **Providing name(s) from the Exception Extensions word set** shall be appended to the label of any Standard System that provides portions of the Exception Extensions word set.

The phrase **Providing the Exception Extensions word set** shall be appended to the label of any Standard System that provides all of the Exception and Exception Extensions word sets.

9.5.2 ANS Forth programs

The phrase **Requiring the Exception word set** shall be appended to the label of Standard Programs that require the system to provide the Exception word set.

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

The phrase **Requiring the Exception Extensions word set** shall be appended to the label of Standard Programs that require the system to provide all of the Exception and Exception Extensions word sets.

9.6 Glossary

9.6.1 Exception words

9.6.1.0875 **CATCH**
EXCEPTION

```
( i*x xt -- j*x 0 | i*x n )
```

Push an exception frame on the exception stack and then execute the execution token xt (as with EXECUTE) in such a way that control can be transferred to a point just after CATCH if THROW is executed during the execution of xt.

If the execution of xt completes normally (i.e., the exception frame pushed by this CATCH is not popped by an execution of THROW) pop the exception frame and return zero on top of the data stack, above whatever stack items would have been returned by xt EXECUTE. Otherwise, the remainder of the execution semantics are given by THROW.

9.6.1.2275 **THROW**
EXCEPTION

```
( k*x n -- k*x | i*x n )
```

If any bits of n are non-zero, pop the topmost exception frame from the exception stack, along with everything on the return stack above that frame. Then restore the input source specification in use before the corresponding CATCH and adjust the depths of all stacks defined by this Standard so that they are the same as the depths saved in the exception frame (i is the same number as the i in the input arguments to the corresponding CATCH), put n on top of the data stack, and transfer control to a point just after the CATCH that pushed that exception frame.

If the top of the stack is non zero and there is no exception frame on the exception stack, the behavior is as follows:

If n is minus-one (-1), perform the function of 6.1.0670 ABORT (the version of ABORT in the Core word set), displaying no message.

If n is minus-two, perform the function of 6.1.0680 ABORT" (the version of ABORT" in the Core word set), displaying the characters ccc associated with the ABORT" that generated the THROW.

Otherwise, the system may display an implementation-dependent message giving information about the condition associated with the THROW code n. Subsequently, the system shall perform the function of 6.1.0670 ABORT (the version of ABORT in the Core word set).

See: A.9.6.1.2275 THROW

9.6.2 Exception extension words

9.6.2.0670 **ABORT**
EXCEPTION EXT

Extend the semantics of 6.1.0670 ABORT to be:

```
( i*x -- ) ( R: j*x -- )
```

Perform the function of -1 THROW .

9.6.2.0680 **ABORT"**
abort-quote EXCEPTION EXT

Extend the semantics of 6.1.0680 ABORT" to be:

```
Interpretation: Interpretation semantics for this word are undefined.

Compilation: ( "ccc<quote>" -- )
```

Parse ccc delimited by a " (double-quote). Append the run-time semantics given below to the current definition.

```
Run-time:       ( i*x x1 --  | i*x ) ( R: j*x --  | j*x )
```

Remove x1 from the stack. If any bit of x1 is not zero, perform the function of -2 THROW, displaying ccc if there is no exception frame on the exception stack.

See: 3.4.1 Parsing

Table of Contents
Next Section