

# StrongForth 3.1

## Contents

1. Introduction
2. Download
3. Files
4. Starting and Using StrongForth
5. Forth 2012 Compliance
6. Known issues
7. StrongForth for Forth 2012
8. Contact

## 1. Introduction

StrongForth is a programming language based on Forth 2012 that demonstrates a number of innovations:

**Strong static type-checking.** Forth itself is often called *typeless* or *untyped*, which means that neither the interpreter nor the compiler do any type-checking when applying a word to one or more operands. It is completely up to the programmer to choose the correct words. In StrongForth, the interpreter and the compiler check if a word matches the number and types of the operands on the stack. This allows finding type mismatches and unbalanced loops and conditional branches already at compile time. The majority of Forth users might believe that strong static type-checking is not Forth-like. Nevertheless, the implementation of StrongForth proves that Forth can incorporate static type-checking without increasing the complexity of the language.

**Overloading.** As a consequence of strong static type-checking, StrongForth allows overloading words by using the same name for different functions, as long as each word expects an unambiguous set of parameters on the stack. A typical example is the word `+`, which in StrongForth is overloaded for double-cell and floating-point numbers. Instead of writing `D+`, `F+` or `M+`, you can always write `+` in StrongForth to add two numbers, no matter which data types they belong to.

**Object-orientation.** StrongForth incorporates object orientation into the Forth language. Words and wordsets, input and output streams, control-flow structures, exception frames, memory spaces and (of course) data types are objects of dedicated classes. Furthermore, it is possible to define application-specific classes or class libraries like container classes. Access to data structures in an object-oriented environment is much more straight-forward and saver than in a typical Forth 2012 environment.

**Native-code compiler.** StrongForth's compiler produces optimized x86 native machine code. Since it uses only one stack, it can take full advantage of the x86 processor architecture. The data stack is actually emulated by the processor's six general-purpose registers. Each word expects its input parameters in registers and returns its output parameters in registers as well. Many primitives allow their operands to be delivered in any register. The necessity for register juggling is further reduced by dynamically assigning registers to data stack positions. Words like `dup`, `drop`, `swap`, `over` and `rot`, for example, can usually be compiled without producing any machine code at all.

**Complex number support.** StrongForth supports complex numbers. You can calculate with complex numbers in the same way as you calculate with scalar floating-point numbers.

## 2. Download

Before downloading and running StrongForth, you have to agree to the following terms and conditions.

*You will be granted a non-exclusive, non-transferable, unlimited, free of charge license to use StrongForth 3.1 for private, academic and testing purposes. You may use StrongForth 3.1 on an unlimited number of devices and by an unlimited number of users. You may change the included StrongForth source code to your individual needs. However, you may not change the StrongForth executable or any of the PDF files that comprise the documentation.*

*The functionality of StrongForth 3.1 and the absence of errors are not guaranteed. Any liability and claims for damages are excluded. Support services are not provided. However, I reserve the right to further develop StrongForth 3.1, fix errors, and make changes upon request.*

*For commercial distribution of StrongForth 3.1 or parts of it, including transfer of the license to third parties, a fee is applicable, to be negotiated between the parties on a case-by-case basis.*

*This agreement is governed by the laws of the Federal Republic of Germany. The place of jurisdiction is Ulm, Baden-Württemberg, Federal Republic of Germany.*

[Download StrongForth 3.1](#)

[Download StrongForth 3.1 with complex numbers](#)

## 3. Files

StrongForth does not need to be installed. Simply unpack the downloaded ZIP file to copy a number of files into a new directory:

- `StrongForth.exe` is the executable.
- `StrongForth.blk` is the default block file with 4096 empty blocks.
- `StrongForth.msg` is another block file with 32 blocks. Each line contains one of 512 messages padded with space characters.
- `glossary.txt` is a pure text file that contains a copy of the StrongForth forth glossary. `help` uses this file to display glossary entries for all overloaded words with a given name.
- `*.sf`, with `*` being the usual wildcard, are StrongForth source files.
- `*.sfx`, with `*` being the usual wildcard, are StrongForth template files.

Here's an alphabetical list of the source files and template files that are available out of the box. You might decide to call it a library. Chapter numbers refer to the StrongForth Reference Manual:

Filename	Description
<code>2dup.sf</code>	Words dealing with pairs of single-cell items: <code>2dup</code> and <code>2drop</code> (see chapter 1), <code>2@</code> and <code>2!</code> (see chapter 2) and <code>2constant</code> (see chapter 10).
<code>accept.sf</code>	<code>accept</code> with advanced line-editing functions (see chapter 32).
<code>alias.sf</code>	<code>alias</code> is a replacement for <code>SYNONYM</code> (see chapter 22).
<code>ancestor.sf</code>	Obtain the ancestor of a given data type (see chapter 7).
<code>ascii.sf</code>	<code>upcase</code> , <code>locase</code> and ASCII control characters (see chapter 29).
<code>asm.sf</code>	Assembler and disassembler (see chapter 28).
<code>baddress.sf</code>	Words dealing with bit addresses and bit fields (see chapter 30).
<code>base.sf</code>	Temporarily changing and restoring the number conversion radix.
<code>bcd.sf</code>	Conversion words for double-cell binary coded decimal numbers.
<code>bench.sf</code>	<code>Sieve</code> benchmark from <i>Byte</i> magazine (see chapter 33).

Filename	Description
bits.sf	Assembler demo: Calculate the number of the highest 1 bit in a single cell (see chapter 28).
block.sf	Implementation of the <i>Block</i> word set (see chapter 25).
bounds.sf	Convert <i>address-and-size</i> to <i>limit-and-index</i> representation (see chapter 29).
catch.sf	execute-only version of catch (see chapter 32).
cflag.sf	Convert a Forth flag ( <code>false</code> or <code>true</code> ) to a C flag (0 or 1).
complex.sf	Support for complex floating-point numbers (see appendix B).
constant.sfx	Template for creating constant definitions (see chapter 32).
countbit.sf	Count the number of 1 bits in a single-cell or double-cell value.
dot.sf	Formatted display of time, date, weekday and roman numbers.
dpl.sf	Vocabulary for numbers entered in fixed-point notation.
editor.sf	Port of the <i>FIG</i> Forth block line editor (see chapter 25).
escape.sf	Source code of <code>\</code> for escaped string literals (see chapter 6).
float.sf	Support for floating-point numbers (see chapter 24).
fxam.sf	Query the state of the top-most floating-point number on the stack.
help.sf	Display glossary entries using <code>glossary.txt</code> (see chapter 32).
long.sf	Conditionals and loops with long branches (see chapter 32).
macro.sf	Defining word for macros that evaluate given character strings.
model.sf	Equivalent definitions for StrongForth words that are not compiled from source code. This file is for reference only and should not be included.
mrg-splt.sf	Versions of <code>merge</code> and <code>split</code> that parse the destination data types.
msvcrt.sf	Constants for MSVCRT data types (see chapter 21).
on_off.sf	Assign <code>true</code> or <code>false</code> flags to single-cell and character-size variables.
order.sf	Search-Order word set (see chapter 23).
permutat.sf	Calculate permutations of an array (see chapter 32).
propagat.sf	Propagate a compound data type from the compiler data type heap to the interpreter data type heap.
qdup.sf	Replacements for <code>?DUP</code> : <code>?if</code> , <code>?while</code> and <code>?until</code> (see chapter 32).
qq.sf	<code>?? &lt;name&gt;</code> is a shortcut for <code>if &lt;name&gt; then</code> (see chapter 32).
qsort.sf	Quicksort algorithm for single-cell items (see chapter 32).
qsort.sfx	Template of quicksort algorithm (see chapter 32).
sdump.sf	Reverse and destroying stack dump in data type specific formats.
see.sf	Assign de-compilation and disassembly words to the virtual method <code>see</code> of class definition and its child classes (see chapter 22).
self.sf	<code>self</code> is used in the stack diagram of class methods to specify a data type reference to the last input parameter.
sgn.sf	Signum function for single-cell and double-cell signed numbers.
smartptr.sf	The smart pointer <code>@@</code> compiles <code>@</code> repeatedly while a suitable overloaded version is available.
sqrt.sf	Calculate the square root of single-cell and double-cell unsigned numbers.
stack.sf	Container classes for <i>stack</i> and <i>queue</i> (see chapter 13).
strext.sf	String Extension word set (see chapter 26).
StrongForth.sf	Definitions of StrongForth words that are to be compiled from source code. This file is automatically included on startup.
struct.sf	Words supporting structures (see chapter 27).
test.sf	Words used by the StrongForth test suite (see chapter 32).
token.sf	Interpreting and compiling qualified token literals (see chapter 32).
xparsing.sf	execute-parsing and execute-parsing-file (see chapter 32).

A subdirectory called `Documentation` contains a number of PDF files with the complete StrongForth 3.1 documentation.

Filename	Description
<code>Cross-reference.pdf</code>	A Forth 2012 vs. StrongForth 3.1 cross reference.
<code>Glossary assembler.pdf</code>	Glossary of the assembler vocabulary.
<code>Glossary forth.pdf</code>	Glossary of the forth vocabulary.
<code>Glossary msvcrt.pdf</code>	Glossary of the msvcrt vocabulary.
<code>Glossary protected.pdf</code>	Glossary of the protected vocabularies of all predefined classes.
<code>Introduction.pdf</code>	An introduction to StrongForth 3.1.
<code>Readme.pdf</code>	A copy of this HTML page in PDF format.
<code>Reference Manual.pdf</code>	The complete StrongForth reference manual.

Within the glossaries, marginal notes mark all words that need to be included from their source files. In order to get familiar with StrongForth, it is recommended to begin reading the Introduction to StrongForth 3.1.

## 4. Starting and Using StrongForth

StrongForth is a 32-bit Windows console application that uses the MSVCRT C runtime library as its interface to the operating system.

To run StrongForth 3.1, make the directory containing the above mentioned files the working directory and execute `StrongForth.exe`. `StrongForth.exe` starts with including the source file `StrongForth.sf`. Many of StrongForth's predefined words are not part of the executable. Instead, they have to be compiled from their source files. At the end of `StrongForth.sf`, you can add commands to include additional sources, for example `float.sf` if you need floating-point support, or `block.sf`, if you want to work with blocks. You can also include your own source files, so your application compiles and runs automatically.

Standard words are recognized in lower case only, because StrongForth is case sensitive. Integer literals have data type `unsigned` by default, and data type `signed` if preceded by a sign character (`-` or `+`). Literals with a trailing decimal point are double-cell numbers. Floating-point numbers are always decimal and have an exponent indicated by an `e`. To exit StrongForth, type `bye`.

## 5. Forth 2012 Compliance

StrongForth is not compliant to Forth 2012. Forth 2012 code will usually cause errors when interpreted or compiled by StrongForth. In StrongForth, each word with at least one input or one output parameter requires a stack diagram. StrongForth stack diagrams look similar to Forth 2012 stack comments. However, they have to comply well-defined rules, and the names of the data types are different (e. g., `single` instead of `x`, `unsigned` instead of `u`). The fact that StrongForth overloads words leads to different names in certain cases (e. g. `+` instead of `F+`, `<` instead of `U<`, `open` instead of `open-file`). Look at the Forth 2012 vs. StrongForth 3.1 Cross-Reference to see the differences. StrongForth's object oriented techniques handle execution tokens, word sets and other objects differently than Forth 2012. Furthermore, StrongForth's built-in address arithmetic makes `CELL+` and `CELLS` mostly superfluous. Sometimes, you might stumble over strange looking restrictions like the one that StrongForth does not provide a word that can add two

addresses. However, the look and feel is still Forth-like, and the definitions of most words will look almost identical in both languages.

StrongForth 3.1 provides all word sets specified by Forth 2012, except for the Extended-Character word set. Some word sets have to be explicitly included from source files. Examples are the Floating-point word set and the Block word set.

## 6. Known issues

None.

## 7. StrongForth for Forth 2012

*StrongForth for Forth 2012* is an implementation of StrongForth in Forth 2012. I. e., it is supposed to run as an application on Forth 2012 systems, taking over properties like the cell size from its host. Although *StrongForth for Forth 2012* has its own interpreter loop and its own dictionary, it takes full advantage of all optimizations that are incorporated in the host system's compiler. At present, *StrongForth for Forth 2012* has been successfully validated for SwiftForth™, VFX Forth and GForth 0.7.0.

### [Download StrongForth for Forth 2012](#)

Unpack the downloaded ZIP file into a new directory. The ZIP file contains modified versions of the files the stand-alone version of StrongForth contains, plus the additional file `strong.f`, which contains pure Forth 2012 source code that builds the StrongForth dictionary and the interpreter within the environment of the host system.

To run *StrongForth for Forth 2012* as an application on your Forth 2012 system, make the StrongForth directory the working directory of the host system. If your host is SwiftForth™, you have to include its floating-point word set, e. g. with

```
INCLUDE C:\ForthInc-Evaluation\SwiftForth\lib\options\fpmath.f
```

If your host is VFX Forth or GForth 0.7.0, you have to define a constant:

```
TRUE CONSTANT VFX
```

or

```
TRUE CONSTANT GFORTH
```

Then, include the file `strong.f`. Next, enter `STRONGFORTH` to start the *StrongForth for Forth 2012* interpreter loop. After the `sf` command line prompt appears, you can optionally include one or more of the source code libraries, e. g.

```
include help.sf
include test.sf
```

To quit *StrongForth for Forth 2012* and return to the Forth 2012 host system, type `bye`. You can resume *StrongForth for Forth 2012* at any time by executing `STRONGFORTH`. Since the *StrongForth for Forth 2012* dictionary will remain as it was immediately before executing `bye`, you don't have to include your preferred libraries and application words again. However, note that *StrongForth for Forth 2012* itself is an application and not a library. You cannot mix StrongForth words and Forth 2012 words, because the execution of words that are not defined in the StrongForth dictionary will most likely corrupt StrongForth's data type system.

You can import additional words from the host system with `host'`, which returns the execution token, and `import`, which adds a new definition with a given execution token to the StrongForth dictionary. Make sure to provide a correct stack diagram. Here's an example:

```
host' DUMP import dump ( address unsigned -- )
```

If the word to be imported is a created definition, a constant, a variable, a value or a deferred definition, `host'` should be replaced with `import-created`, `import-constant`, `import-variable`, `import-value` or `import-deferred`, respectively. All these words including `import` have to be imported from `import.sf`.

Please pay attention to some issues related to the various host systems. Since GForth 0.7.0 is not a native-code system, `see` doesn't work at all. It is recommended not to include `see.sf`. VFX Forth seems not to support function keys and other special keys from the PC keyboard. Therefore, `K-F1` and other constants are not available. StrongForth's special version of `accept` cannot be included from `accept.sf`. This is regrettable, because VFX Forth's version of `accept` displays an irritating additional character at the beginning of each input line. Furthermore, `;` code does not work on a VFX Forth host.

*StrongForth for Forth 2012* provides almost the same functionality as StrongForth 3.1. However, there are some modifications, which are mostly caused by missing properties of Forth 2012:

- The data space is the only memory space. Class `memory-space` and all its members and methods are not provided.
- `#hold`, `#locals`, `base` and `state` are cell-size variables.
- Less efficient code, compared with StrongForth 3.1, because the host system does not emulate the data stack using registers.
- Instead of using class `control-flow` together with `(branch)` and `(0branch)`, *StrongForth for Forth 2012* takes advantage of the host system's control flow mechanism.
- `>r`, `r@` and `r>` directly use the return stack instead of the stack frame. Floating-point numbers cannot be stored on the return stack.
- `(>token)` does not expect a flag as an input parameter.
- class `compiler-workspace` does not exist.
- `(execute) ( definition -- )` is being provided in interpretation state, with similar semantics as `(compile) ( definition -- )` in compilation state.
- Using the host mechanism for locals limits the number of locals and restricts locals to single-cell values.
- The host mechanism is used for representing floating-point values.
- Static objects have to be created with `static` instead of with `new` and a given address or memory space.
- Constants, variables and values use the respective host mechanisms. Each has its own subclass of class `definition`.
- No mixed-mode operations for multiplication and division
- No direct access to the processor flags like *carry*, *overflow* and *parity*.
- Exceptions without handler are caught by the host system.
- `2dup`, `2drop`, `2swap`, `2over` and `2rot` are provided for single cells.
- Classes `virtual-definition`, `member-definition` and `bmember-definition` are not provided. Instead, members and methods are created definitions.
- Colon definitions are objects of the dedicated class `colon-definition`.
- Data type `tuple` is provided in order to support the control flow data types.
- `depth` is provided.
- No low-level access to the floating-point stack and to the floating-point status.
- `marker` uses the host mechanism, requiring a dedicated class `marker-definition`.

- Since there is no MSVCRT library, the `msvcrt` vocabulary and all its words are missing.
- More missing Words:
  - `inline`
  - `-! and -loop`
  - `bye ( integer -- )`
  - `?set and ?clear`
  - `lrotate and rrotate`
  - `pack and unpack`
  - `shrink`
  - `size ( address -- )`
  - `sp@ and sp!`
  - `smembers`
  - `split ( float -- signed 1st )`
  - `write-eol`

## 8. Contact

Questions, comments and new ideas are welcome. Send an email to [stephan.becher@t-online.de](mailto:stephan.becher@t-online.de).  
The latest versions and up-to-date information are available from this web site.

---

April 24th, 2024