Using ANS Forth:
A report from the trenches

We will take a look at the practicalities of using ANS Forth systems and see what present systems are providing.

- What ANS Forth's are available
- The basics: loading the system
- File I/O
- What is **really** different
- Summary: Can it be used for large projects ?

What ANS Forths are available ?

The simple answer is: **none**

There are *no* Forth systems around that satisfy the ANS standard document as it is strictly interpreted.

The primary reason for this is the expense of the offical standard document, and it is not proper to claim standards compliance to the draft standard.

Fortunately, this is of no real practical impact. There are several systems available that are modeled on the draft standard; they are not officially standard, but for all practical purposes they are standard systems.

All of these systems are in the public domain or shareware, they include

- `F68KANS`, for the 68xxx family by Joerg Plewe (`jps@Forth-eV.de`)
- `PFE`, for MS-DOS, Linux, Unix by Dirk Zoller (`duz@roxi.rz.fth-mannheim.de`)
- `gForth` by Bernd Paysan (`paysan@informatik.tu-muenchen.de`)
- `iForth` by Marcel Hendrix (`mhx@bbs.forth-ev.de`)

- `ThisForth`, for practically anything with a C compiler, by Wil Baden
- `Win32For` by Tom Zimmer and Andrew McKewan

The last two in the above list are not strictly compliant, but are close enough to put in this list.

There are *no* commercial systems that claim compliance.

There is are two pieces of software that you can use to test compliance to the standard:

- To test the compliance of the *compiler*, there is John Hayes' test suite.
- To test the compliance of the *application*, there is Jet Thomas' `StanForth` program.

Loading the system (MS-DOS and Unix)

These systems are "toolbox" Forth systems —- Forth is only one tool in the programmers toolbox. Source files are edited from outside of Forth, external debugging tools are also required as no special debuggers are provided from *within* these Forths. (An excellant ANS debugger is the one written by **Joerg Plewe**, it was published in **Forth Dimensions, XVII, Number 1**).

To load these Forths, give the executable name on the command line listing any files to initially load as command line arguments. Various command line switches can be used to set up the initial behavior of the system (for example `PFE` can be told to be case sensitive, instead of the default non-sensitive).

There is no default configuration file that is read at startup.

`ThisForth` can be told to *interpret* the command line arguments and not just treat the arguments as file names to load.

Under Unix/Linux, both `ThisForth` and `PFE` can be run as standalone scripts (in place of `perl` or `Bourne` shell scripts) by placing the following line as the first line of the Forth source file that you want to run:

```
#!  /usr/local/bin/forth
```

To run the program, just type the name of the file.

```
\#! /usr/local/bin/forth
\  example executable Forth script
: hello
   ." hello world !" cr
;

hello
bye
```

Once the system is running, external files can be loaded and interpreted through standard ANS words, e.g.:

```
s" filename" included
```

this is equivalent to the `FPC` sequence,

```
fload filename
```

A file can also be loaded and interpreted under *program* control by opening a file and including it,

```
  s" filename" r/w open-file
abort" file open failed" include-file
```

File I/O

We have seen a little bit of how file I/O is handled.

Personal note: ANS Forth provides for Blocks or Files. When running Forth on top of an operating system that provides its own file system, I find Blocks to be impratical. They are not compatible with all the other tools that I use external to Forth itself for managing my source code.

A file is opened with the `open-file` word which returns a file handle and a success/failure flag. This handle is then used for subsequent access to the file.

The `File Access` wordset is fairly minimal, it provides the ability to get/set a position within a file and to read/write bytes or lines. There are no words to directly read/write a number in/to an ASCII file. For reading one needs to read in the characters and interpret them as a number separately. For writing, an integer has to be formatted as characters and then written out.

What is REALLY different

No messing with the dictionary headers

If one is writing a standard application, it cannot know about the threading mechanism that the system implements. There is no mucking about with the `LFA`, `NFA`, or `CFA` allowed.

INVERT, 0= and NOT

There are three inter-related words: `INVERT`, to change all the one bits to zeros and vice versa, `0=` for logical tests and `NOT` which is allowed to be an alias for *either* `INVERT` or `0=`. The compiler documentation is required to tell you which. The net result is that it is very dangerous to use `NOT` in a standard application.

POSTPONE

ANS does not specify that certain words must be immediate, this is left to the designer of the implementation. As a consequence, it is not always obvious whether the word `COMPILE` or `[COMPILE]` is needed within a definition. The word `POSTPONE` is designed to safely replace *both* words without regard to the immediacy of the word that is being compiled in.

In spite of this simplification, there are situations that can be confusing. The confusing cases typically would have used the sequence `COMPILE [COMPILE] some_word`, which is a confusing sequence to begin with.

UNLOOP

There is a word `UNLOOP` which is intended to allow a word to cleanly `EXIT` from within a `DO LOOP`,

: xx ... DO (stuff)
        IF UNLOOP (possible extra stuff) EXIT THEN
        (more loop stuff)
    LOOP ;

There is a **technical flaw in the specification** of `UNLOOP` in that it does not state that a following `EXIT` is *required* or that an ambiguous condition for `LOOP` will result if it is missing.

Division

The division and modulus words are allowed to *either* round towards zero (symmetrical division), or to round towards negative infinity (floored division). If a particular algorithm *requires* a certain form, then it should use the explicit operators (`sm/rem` for symmetrical, and `fm/mod` for floored).

Wordlists

The unification of Forth vocabulary mechanisms was as badly needed as unifying Forth File I/O. The ANS solution here was to provide a set of words that are at a much more fundamental or primitive level. These words can either be used directly, or to implement a more traditional appearing vocabulary scheme. Unfortunately, it appears that the practical result of this is that the user must learn yet another vocubulary management scheme, in spite of the technical merits of the wordlist mechanism.

Can you really use it?

Yes ! There are flaws in the standard, but the first time around that is bound to happen. It is certainly better than the first Ada standard ! ANS Forth is a practical usable language.

By using ANS Forth you gain,

- portability between compilers on a given platform
- portability (to a degree) between platforms and operating systems
- a practical language which can be used to write general purpose libraries.