FORTH CODING RULES

Created by Paul E. Bennett of Transport Control Technology Ltd.
Date of original: 23rd March 1995
Date last modified: 28th December 1996 (for the Forth Interest Group and comp.lang.forth)

As promised I am posting the Coding Rules, which have been modified from my own company standard and with some modifications as applied to a recent European Contract. The rules have been further edited to address a couple more specific issues. The stack picture information is configured in such a manner that automated tools can generate glossary entries. These stack picture formats will require some additional words to be defined in systems if they are used in source code. They are:

```
    (S adopts the function of ( unless used with the glossary tool
    (R adopts the function of ( unless used with the glossary tool
    (G adopts the function of ( unless used with the glossary tool
```

I shall, as soon as I am able, post some code, which will give an idea of how the Glossary generation Tool was intended to work. I am writing it at present (one of those jobs I intended to do for a long time now).

TABLE OF CONTENTS

---

1. INTRODUCTION

The goal of authoring software is to write all documentation as part of the source code in the form of comments. External documentation should be automatically derivable from the source code. To enable this requires that all sources conform to a standard layout and coding and commenting style.

The aim of this standard is to make the task of reviewing and certifying software easier. In achieving the aims of this standard, certain styles of software source layout, programming and commenting style are mandated.

---

2. SCOPE

This document provides guidance on the construction of software source code in order to meet the security, quality and integrity objectives of Europay International. Following this guidance should result in software source listings that are easy to read and understand by being presented in a standard format. This document is mandated for application providers and highly recommended for Kernel providers.

---

3. REFERENCE DOCUMENTS

1. ANS X3.215 "Information Systems - Programming Language - Forth" 1994
2. ISO/IEC 15145 "Information Technology - Programming Languages - FORTH" (=ISO/IEC DIS 15145:1995) 1995
3. "Thinking Forth" Brodie, Leo. 1984 (ISBN 0-13-917568-7)

---

4. GLOSSARY OF TERMS

The following terms are used within this document:

ANS	American National Standard
Application	Software that implements a system product. Applications are comprised of modules.
BS	British Standard
EN	Comite’ Europe’en de Normalisation
IEC	International Electrotechnical Commission
IOR	Result of an I/O operation (a flag value that conveys information about the type of failure, if any, from an operation with I/O device).
ISO	International Standards Organization
Lexicon	The words of a modular component made visible to other modules.
Library	A set of high-level software functions with a published interface.
Object	References within this document to objects relate to Method-Objects and not Data-objects. Method objects may relate to specific standard method for performing a task.
Word-set	A set of Forth definitions grouped together under a name indicating some shared aspect, typically their common functional area.

---

5. SOFTWARE STANDARDS APPLICABLE

All software shall conform to recognized standards for the language in which the coding is described. The International, European and National standards that are accepted as being applicable are ISO, IEC, EN, ANS and BS. Company

standards, where generated to provide specific standardized practices, shall take precedence. Specification documents shall state which standards are applicable.

---

6. SOFTWARE LISTING ORGANIZATION

Software listings shall utilize the format from the template files (TEMPLATE.FTH for text files or TEMPLATE.SCR for screen files) which provides the basic framework for the required document tracking information.

Listings should be structured hierarchically, like a well-written and organized book. Each listing should clearly relate to the specification document from which the requirements are drawn. The author of the software listing shall provide clear aids to the trace-ability of the software to its requirement's specification wherever possible. This shall include glossary entry text, further non-glossary descriptions and references to sections or paragraphs of the specification document.

6.1. DOCUMENT AUDIT INFORMATION

6.1.1 TITLE BLOCK

The title block contains the company identification and copyright notice sections of the listing template. The title block is an embedded form, which requires standardized information to be filled out. This information identifies part of the trace-ability to the requirement specification and gives basic details about the listing.

6.1.2 INTRODUCTORY OUTLINE

Each source file will contain a lexicographically complete set of functions for an application area. The Introductory outline will concisely describe the contents of the file. This shall be limited to a maximum of 100 words or as far as the first page break in a text file and one screen in a block file. The text for the introduction could be copied from the related specification document if such text concisely describes the listing contents.

6.1.3 EXPORTS

This shall provide a list of functions that are available for use by other modules. This is not necessarily the entire list of functions defined within the listing. Some functions within this listing may only be suitable for local usage.

6.1.4 ENVIRONMENTAL DEPENDENCIES

This shall list the specific non-standard words or word-sets required. It shall also define specific hardware requirements and their access arrangements for the functions specified within the listing.

6.1.5 REVISION HISTORY

The revision history shall be listed in reverse chronological order. Each item of revision shall be categorized for its effect (either as Major, Minor or Trivial) within the description of change(s) made. Rules for revision history cleansing is to be formulated through discussion.

6.2. SOURCE CODE DEFINITION

6.2.1. INCLUDES

Pre-load command list to include all other software sources required in building this module.

6.2.2. CONSTANTS AND VARIABLES

A list of Constants and Variables that are first used within the current source file or screen.

6.2.3. FUNCTIONS

In Forth, Functions are named sub-routines, also known as Words. The name of each function shall be as descriptive of what the function does as is practical. Functions may be colon definitions or assembler code definitions.

Each function definition is in the form of the following, see also colon definition in reference document 1.

```
: <name> (S input-stack --- output-stack) (R --- )
         (G descriptive glossary text)
  f1 f2  ( interim stack picture) \ programmers explanatory comment
  .. fn ;
```

Where:

:	begins a colon definition.
<name>	is the name of the function.
(S	is a data stack comment.
input stack	is a list of input stack items separated by a space character between each item.
output stack	is a list of output stack items returned by the function with a space character between each item.
(R	is a return stack comment.
(G	is the glossary text entry comment.
f1 f2 .. fn	are pre-defined functions.
interim stack picture	expected intermediate state of the functions stack elements. (S and (R may be used to distinguish data and return stacks.
programmers explanatory comment	A concise comment about the immediate line of source code which explains what is happening.
;	ends a colon definition.

---

7. PROGRAMMING STYLE

7.1. NAMING CONVENTIONS

Names for colon or code definitions shall relate to the functional aspects of the definition in a manner that reflects what function the definition performs rather than how the function is performed. Names shall be English-based and as short as possible, long enough for clarity but no longer than 31 characters.

7.1.1. PERMITTED CHARACTERS WITHIN NAMES

The following list of Graphic Characters, extracted from reference document 1, are the only characters permitted for use within Forth definitions.

```
  !"#$%&’()*+,-./ digits [0-9]:;<=>? @ ALPHA [A-Z] [\]^_‘ alpha [a-z] {|}~
```

These are represented by ASCII character codes 0x21 to 0x7E inclusive.

Use of Control Characters or any Graphic Characters outside of the above range within definition names is prohibited.

7.1.2. NAME CONSTRUCTION

Words for applications are of two types: those, which are contained within a module and those which are not. Use of modular constructs is highly recommended for application code. Similar objects shall be contained within the same module. The similarity shall be judged on the object type that is

to be handled. These are usually based on standards or written procedures.

7.2. DEFINITION LAYOUT

The body of a definition should be constructed with respect for the phrasing embodied within natural English. This uses words as if they were a collection of nouns or verbs with adjectives and adverbs as modifiers. White space in the definition is an important aspect of the readability of source text.

A definition should, ideally, be kept short. In no circumstances shall a definition extend for more than one page (approximately 50 lines of 12pt text, allowing for header and footer margins) and shall not occupy more than 80 characters in width.

7.2.1. INDENTING POLICY

The colon of a colon definition shall begin in the left-most position against the page's left margin. The main body of the definition shall be indented by one tab stop, which shall be set at 3 character spaces. Each control structure shall indent further from this by 3 characters for each subsequent indenting level, see section 7.7 for maximum depth of a control structure. Tab stops shall be implemented as hard space characters.

7.3. STRUCTURE

All programming shall follow the rules of Structured Programming. Each definition is highly recommended to have only one entry point and only one exit point except for where errors occur that require system state re-establishment. Exceptions and error conditions for which no IOR are foreseen shall be handled only by the standard exception handling mechanisms of CATCH and THROW and any logical extensions of those words.

7.4. DIFFERING ACTIONS DEPENDING ON MODE

Where a word being defined is intended to perform different actions, dependent upon whether it's in compilation or execution or any other state, the stack effects for each state shall be given on separate lines and the state to which the stack effects relate denoted in a comment. Order of listing preferred is Compiling then Executing.

```
    : ARRAY
      CREATE (S x-size y-size ---)              \ compiling
      ...... .........
      DOES>  (S x-offset y-offset --- a-addr)    \ executing
      ...... ......... ;
```

7.5. STACK EFFECT ARGUMENTS

All data stack effect arguments shall be commented for all words defined, even if there is a null effect and no arguments are consumed or produced by the word. The stack effect arguments shall indicate the type parameter being passed, if any. The "Data types" nomenclature of the table in reference document 1 are recommended for denoting stack parameter data type. Other nomenclature may be used, but when used shall be as plainly indicative as is practical. A space character shall separate stack items. Hyphenation of data type nomenclature is permissible.

The effects on the Return Stack shall be indicated when words that affect the Return Stack such as >R, R> and R@ are used.

7.6. WORDS THAT RETURN DIFFERENT NUMBERS OF ITEMS

Words that return different numbers of items shall have both versions of their stack effects indicated on separate lines, e.g.:

```
    : ?RX (S --- char true)    \ character present
          (S --- false)        \ character not present
```

```
      ...... ......... ;
```

Additional comments may appear to right of stack effect comments to provide further explanation. Use of such words in an application shall be minimized.

7.7. CONTROL STRUCTURES

Control structures include all branching requirements for program control flow, including loops. The maximum depth of nesting control structures within a single definition is highly recommended to be limited to three. Requiring more than two nesting levels usually indicates the need to consider further factorization.

7.8. USE OF LITERAL VALUES AND "MAGIC NUMBERS"

Use of "magic numbers" or literals within the body of a function definition is not permitted. All required numbers shall be derived or contained within pre-defined constants. This eases the process of making changes to these values by changing just the value stored in the appropriate constant.

7.9. GLOSSARY TEXT

All words shall be accompanied by clear, concise explanatory text in the form of a glossary text entry, which may be extracted later by automated tools that aid in the generation of user documentation. Glossary text shall refer to the declared stack effects of a word, if any, and declare any exceptions that might be generated by the function for non-valid parameters.

7.10. VARIABLES, CONSTANTS AND VALUES

Variables and Constants are ways of providing named number placeholders. Good design of a program, within Forth, will minimize reliance on the use of CONSTANT or VARIABLE as much as practical. However, where they are needed then the following guidelines will assist in their use.

  a. Variables, Constants and Values should be defined local to where they are used the first time.
  b. Always initialize a VARIABLE to some value when it is initially declared.
  c. For the piece of code, in which a variable or value is used (which may deal with a specific function), create an initialization word for each functional section, doing this will reduce the size of the top-level initialization word and make code management easier.
  d. Always perform the top-level initialization with a GO or RUN word.

This way, if parts of the hierarchy are used within several applications then you will not miss any if you change something and create a new variable that will need initialization.

---

8. AIDS TO VERIFICATION AND VALIDATION

   * Test Scripts
   * Checklists
   * Pre-validated Test Data
   * Random Test Data

---

9. CERTIFICATION OF SOFTWARE

Software certification is a declaration that all software described within a source listing is complete, performs its functions without error, is resilient against out-of-bounds stimuli and has no unresolved references. The certificate also declares that the functions performed comply with the requirements as described in the specification document.

Certification is applied in a multi-staged process that examines the resultant code formally in Static and Dynamic testing regimes in accordance with a test script designed to prove compliance. All certification processes, consequently, require deriving the assurances they give about the software from a full and thorough audit trail tracing the design and development process from requirement's specification to final product. For this reason version and change control management systems, manual or automated shall be used to ensure full knowledge of the quality basis for all products.

All comments are welcome.

Paul E. Bennett <peb@transcontech.co.uk>
Transport Control Technology Ltd.
+44 (0)117-9499861
Going Forth Safely