Clarion Style Guide

This document defines programming standards to be used for the Clarion Language and the Clarion Template Language. All applications, data dictionaries, and source code produced by TopSpeed Corporation should comply with these standards.

This document defines the current programming standard. This standard will change from time to time, but there is always a single current standard that must be followed for all Clarion source and template modules produced by the company.

The Clarion Language

■ Columns and indentation

In the DATA section, labels begin in column 1, statements begin in column 18, and comments begin in column 50

In the CODE section, unindented statements begin in column 3 and comments begin in column 50.

Statement following compound statements such as MAP, MODULE, GROUP, QUEUE, FILE, RECORD, REPORT, HEADER, FOOTER, DETAIL, BEGIN, CASE, OF, OROF, ELSE, EXECUTE, IF, LOOP, etc. are indented 2 columns.

ROUTINE statements should be separated from their label with a single blank character.

■ Comments

Every data declaration and every non-blank statement following the CODE statement should be commented. SCREEN, APPLICATION, WINDOW, and REPORT structures are not commented. Only the data fields in FILE and VIEW structures are commented. Comments for END statements indicate the statement they terminate (e.g. !End IF). Comments may be indented for clarity, but the comment initiator (exclamation point) stays in column 50.

An OMIT block can be placed after a PROGRAM, PROCEDURE, FUNCTION, or ROUTINE statement for a block of comments. The block may be surrounded by single line graphic characters. If so, the OMIT terminator should be a lower right corner character.

■ Capitalization

Labels are word capitalized (e.g. PrintLine). Keywords are all capitalized (e.g. GET, FILE). The first character of a comment is capitalized.

■ IF Statements

IF statements with a single THEN statement group can be coded on a single line. Single line IFs contain a THEN separator and do not contain ELSIF or ELSE statement groups. Multi-line IF statements do not contain a THEN separator. Multi-line conditional expressions should be continued so that AND or OR connectors line up under the IF statement.

■ Labels

— Data Names

Data names stand for things and, therefore, should be nouns or modified nouns (e.g. Amount, VenderCode). They should be clear and concise and, if possible, elegant. Creating good data names is an art, but it is art that should be controlled by science. To illustrate, if a program needs a variable to count the number of items displayed, there are many options:

- "NumberOfItems" is verbose.
- "NoOfItems" is inelegant (unreadable and unpronounceable).
- "NoItems" is ambiguous. (It's a good label for a "no more items" flag).
- "ItemCount" is acceptable but not the best choice.
- "ItemCnt" is acceptable but not the best choice.
- "Items" is the best choice because it is clear, concise and elegant. Try some example syntax if you remain unconvinced.

— Procedure Names

In program syntax, procedure call statements request action . Accordingly, procedure names should be predicate forms (e.g. "Compute", "PrintTotals", "GetStatus").

It is not unusual to encounter a noun, such as "Initialization", used as a procedure name. This usage labels procedures by type. It is always easier to name what a procedure does than to name its type or category. Think about what happens if the "initialization" procedure needs to be split up. What do we name its parts? "InitializationOfData" and "InitializationOfStatus"? If the procedure were named "Initialize" its parts would be "InitializeData" and "InitializeStatus".

— Function Names

In program syntax, function call statements deliver values like data names. Accordingly, function names should be nouns or modified nouns. Often, functions are named for the action they produce (e.g. "GetSmallest", "ReadCharacter"). This usage usually isn't as clear:

Message = CLIP(Message) & ReadCharacter()   *doesn't read as well as*
Message = CLIP(Message) & NextCharacter()

■ Mnemonics

When you use a mnemonic as a substitute for a word, you assume that anyone reading the program knows what the mnemonic stands for. Use only well known abbreviations (e.g. Amt, Cnt, Col, Msg, Ndx, Prt, Rec, Tmp, etc.)   Don't use mnemonics that abbreviate a single character (e.g. Itm, Lne, Fil).

A silent "e" changes the pronunciation of the prior vowel. Therefore, a mneumonic formed by removing a silent "e" will ALWAYS be mispronounced. Since pronunciation is the primary tool used to reconsititute a mneumonic, it is a bad idea to remove a silent "e".

■ Parameter Lists

Parameter lists should be compressed with no extra spaces. Parameters should be separated by commas. Although long parameter lists must be continued on multiple lines, parameters should not be placed on separate lines. If parameters of a PROCEDURE or FUNCTION statement must be documented, an OMIT block should be used.

■ Punctuation

Each statement should be placed on a single line. Statement separators (semicolons) are not used in standard Clarion. Continuation symbols (pipe characters) should be placed prior to any comment in column 50. Although not required, a comment character (exclamation point) should introduce comments on continued lines.

The underscore character is not used in Clarion labels. Colons may be used to designate special prefixes (e.g. SAV:Record). Except for a single line IF statement, statement groups should be terminated with an END statement. Single line IF statements should be terminated by a period.

■ Spacing

Single blank lines can be placed strategically for statement grouping.

■ Language Relics

There are certain Clarion statements and functions that are no longer useful and are supported only to be compatible with prior versions. Such statements and functions must not appear in new Clarion code.

— The **EOF()** and **BOF()** functions should not be used. Importantly, the construct **LOOP UNTIL EOF(file);NEXT(file)** must never be used. There are two reasons for this. First, this construct contains a timing window between the **LOOP** and **NEXT** statements during which the last record of a file can be deleted by another workstation on a network. So

ERRORCODE() must be checked after **NEXT** anyway. Second, most database engines cannot supply an end of file or beginning of file status without reading the file. This effectively doubles the time required to read a record.

— The **POINTER()** function should not be used and the **SET** and **GET** statements should not be used with file pointer parameters. Many database drivers do not support this feature. The **POSITION()** function and **RESET** statement should be used to reposition during sequential processing.

— **REPEAT** and **MENU** structures should no longer be used. **PAUSE** and **POINT** fields should no longer be used.

(TrialPak Note: this is a TopSpeed internal document which we thought you might find useful as an insight into the language. It's not actually published in the Language Reference)