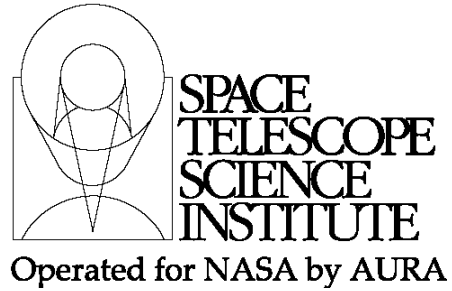




TECHNICAL REPORT



Title: STScI CECIL Scripting Standards	Doc #: JWST-STScI-001146, SM-12
	Date: 15 May 2007
	Rev: -
Authors: Alan Welty Phone: 410-338-4948	Release Date: 17 May 2007

1.0 Abstract

This document summarizes the guidelines for writing CECIL scripts at STScI. Two categories of standards are included: 1) those governing style, the emphasis being on readability of comments and code, and 2) those governing internal documentation.

2.0 Introduction

When multiple people work on code development there are as many coding styles as coders are likely to be present in the final product unless standards are enforced. Indeed, most individual coders do not consistently apply even their own standards. This document is intended to provide a set of standards which all CECIL coders and reviewers will be expected to know and follow. A rationale is provided with each standard. The intent is that STScI CECIL scripts, written for diverse purposes, will be easily readable and understandable, and have uniformity of appearance.

3.0 Stylistic Guidelines

- 1) Line length should not exceed 80 characters, including the continuation character. When continuing a long statement, breaks should be made after commas or before operators. However, a break must not occur before the * or ** operators lest the continuation line be treated as a comment line.

Rationale: Applications that people may use to view or edit scripts, may wrap long lines or require scrolling to view unwrapped long lines. If a print application does not wrap long lines, the ends of such lines may not show up at all. In these cases, readability is impaired.

Operated by the Association of Universities for Research in Astronomy, Inc., for the National Aeronautics and Space Administration under Contract NAS5-03127

Check with the JWST SOCCER Database at: <http://soccer.stsci.edu/DmsProdAgile/PLMServlet>
To verify that this is the current version.

- 2) A continuation line must be indented at least 3 spaces more than the first line of the statement. Spacing beyond three spaces is at the discretion of the coder and reviewers.

Rationale: Such spacing enhances readability.

- 3) For `DO` loops and `IF` blocks, the termination statement must start at the same column as the `DO` or `IF` statement, and the contents of those structures are to be indented 3 more spaces.

Rationale: This standard sets code to be executed repeatedly or conditionally apart from code that is not in the same control or logical structure, thus enhancing readability. The number of spaces to indent is intended as a balance between setting off the subordinate code and keeping line length to a minimum, especially in nested situations.

- 4) Statement labels (most commonly used in `DO/CONTINUE` constructs) should be short strings that identify the purpose of the statements, as opposed to other commonly used labels like numbers.

Rationale: This guideline facilitates self-documenting code, and can be useful when reading code with nested control and logical structures.

- 5) Avoid negative logic in control statement expressions. For example, try to write code like

```
IF (<expression>)
```

rather than

```
IF (!<expression>)
```

Rationale: Most programmers think and program in terms of positive logic. Thus, programming with positive logic will benefit the JWST software collaborators.

- 6) Do not use “backward” logic. For example, use

```
IF (<LOGICAL VARIABLE> == TRUE) or IF (<VARIABLE> == <value>)
```

rather than

```
IF (TRUE == <LOGICAL VARIABLE>) or IF (<value> == <VARIABLE>)
```

Rationale: `IF/ELSE` statements should be written so that they can be read sensibly from left to right.

- 7) Declare all local variables, one per line, at the start of the script code. List them alphabetically after a comment banner “; Local Variables”.

Check with the JWST SOCCER Database at: <http://soccer.stsci.edu/DmsProdAgile/PLMServlet>
To verify that this is the current version.

Rationale: This is for standardization, and should make it easier for reviewers to review code.

- 8) Do not abbreviate Eclipse statements or keywords.

Rationale: Although Eclipse allows “minimum match” specifications, e.g., `ST` for `START`, full spellings are desired for clarity.

- 9) The TAB character shall not be used. The space character is to be used for all horizontal whitespace.

Rationale: Different platforms and applications may display TABs differently, sometimes destroying the visual alignment.

- 10) Spacing around operators, commas, and parentheses should be maintained as follows:

- Binary operators: one space before and one space after
- Unary operators and “)”: one space before
- Commas and “(“: one space after

Rationale: This horizontal whitespace enhances readability.

- 11) Avoid complex nesting of code within statements.

Rationale: Use of function calls or expressions as arguments of operators or parameters of function calls is frequently a convenient shorthand. However, multiple instances of this capability in a statement can render the statement very difficult to read and understand.

- 12) Before any use of the division operator, it must be verified that the denominator is not equal to 0. If it is 0, an error path must be taken.

Rationale: Division by zero causes a runtime error.

- 13) Letters, digits, and the underscore are the only characters permitted in variable names. The first character must be a letter.

Rationale: This set of characters is adequate for creating meaningful variable names. CECIL does not allow other characters in variable names.

- 14) CECIL is case insensitive. Nevertheless, the following case rules shall be followed:

All letters in global constant and variable names shall be upper case.

Check with the JWST SOCCER Database at: <http://soccer.stsci.edu/DmsProdAgile/PLMServlet>
To verify that this is the current version.

All letters in local variable names shall be lower case, unless the name is a concatenation of words or acronyms, in which case all but the first word or acronym may be capitalized.

All other letters, e.g., in operators and keywords, shall be lower case.

Rationale: These are commonly used programming conventions and help make code readable.

- 15) Variable names should indicate the purpose of the variable. For example, the distance from point A to point B might be `distFromAtoB`, but should not be `x`.

Rationale: Meaningful variable names make it easier to understand code.

- 16) Use double quotes to delimit strings. Use single quotes only when necessary due to nesting.

Rationale: This requirement helps to standardize the code and facilitate code review.

- 17) Top-level scripts shall perform range checking on all user-supplied parameters prior to issuing any commands.

Rationale: User-supplied parameters are external to the code in the scripts. It is prudent to check the value of each parameter prior to its use. By putting all these checks in the top-level scripts, they are neither spread out over nor duplicated in lower level scripts.

- 18) Blank lines should be inserted liberally in the code to set off groups of related statements from other such groups, but do not insert consecutive blank lines.

Rationale: This practice enhances code readability.

- 19) Recursive algorithms shall not be employed.

Rationale: Recursive algorithms have the potential for infinite loops and excessive resource consumption.

4.0 Documentation Guidelines

- 1) There shall be only one procedure or function in a CECIL script file.

Rationale: This rule simplifies code maintenance.

- 2) The filename shall match that of the procedure or function in the file. The file extension shall be “.prc”.

Rationale: This rule simplifies code maintenance.

- 3) CECIL script naming conventions are as follows.

- Scripts for commanding NIRCam shall begin with “INRC”.
- Scripts for commanding NIRSpec shall begin with “INRS”.
- Scripts for commanding MIRI shall begin with “IMIR”.
- Scripts for commanding the attitude control system shall begin with “IACS”.
- Scripts for commanding the fine guidance sensor shall begin with “IFGS”.
- Scripts for commanding the tunable filter imager shall begin with “ITFI”.
- Scripts for commanding more than one of the above shall begin with “IGEN”.
- Test scripts shall begin with “ITST”.
- The 5th character in script names shall be an underscore.
- Script names shall have 22 or fewer characters (from a limited character set: capital letters, digits, and underscore).

Rationale: Such names will reflect the functionality and provide some information about the script calling hierarchy. The leading “I” is the indicator for STScI scripts.

- 4) All CECIL script comments will begin with the logical end-of-line character, the semi-colon, including lines that have no executable code. The asterisk will not be used (as first non-blank character) to identify comment lines.

Rationale: All comments in our scripts will be easily identifiable as such if they are written with the same comment indicator.

- 5) A code comment shall be indented the same amount as the code to which the comment applies, i.e., the semi-colon is to be in the same column as the start of the code. The semi-colon is to be followed by a single space and then the text of the comment.

Rationale: This practice makes it easier to read scripts.

Check with the JWST SOCCER Database at: <http://soccer.stsci.edu/DmsProdAgile/PLMServlet>
To verify that this is the current version.

- 6) A code comment shall not be separated from code to which it applies by unrelated code.

Rationale: Keeping code and related comments together helps avoid confusion.

- 7) A code comment shall be set off from preceding code by a blank line. It is at the discretion of the coder and reviewers whether a blank line is inserted between a code comment and the code to which it applies.

Rationale: This practice makes it easier to read scripts.

- 8) A variable declaration statement should generally have an end-of-line comment to describe the variable.

Rationale: Such comments help make code more understandable and facilitate code reviews.

- 9) All CECIL scripts will have a comment header, prior to the PROC or FUNC statement, as illustrated below. Note that blank lines, indentation, spacing, capitalization, & date format in the example are part of the intended standard! The opening and closing separator lines have 78 dashes after the semi-colon. Note that no characters follow the colon in the various headings, except when “None” is the only following text.

```

;-----
; NAME.prc
;
; PURPOSE:
;   Write text that briefly describes the purpose of this script. If more
;   than one line is needed, indent it the same as the first line.
;
; LIEN:
;   If there are liens against the script, list them here. Each lien gets
;   a new LIEN heading.
;
; INPUT PARAMETERS:   None
;
; OUTPUT PARAMETERS:
;   status             Script completion status: SUCCESS or FAILURE
;
; NOTES:
;   List items of information that may be helpful in understanding
;   the script or that should be noted when modifying it. Optional.
;
; *****
; * Optional banner containing important health and safety reminders *
; * and pertinent software interface constraints.                    *
; *****
;
; HISTORY:
;   02-Sep-2005  ADW  Initial version.
;   23-Feb-2006  ADW  Write brief text describing the changes made. If more
;                       than one line is needed, indent like this example.
;                       The date is that of the latest modification for the
;                       overall change being made. The initials are of the
;                       primary coder.
;-----
PROC NAME (status)

```

Check with the JWST SOCCER Database at: <http://soccer.stsci.edu/DmsProdAgile/PLMServlet>
To verify that this is the current version.

Rationale: Such a header provides basic information about the purpose, history, and interfaces of each script.

- 10) Scripts may also have a comment trailer block, set off by separator lines as above, after the `ENDPROC` statement. Such comment blocks will be used for supplemental material that is long or not appropriate for the header comment block or code comments. (For example, the telemetry monitor mnemonic vs. monitor ID number tables produced by the Telemetry Monitor Table Generation Tool is written to such a comment block in that tool's output CECIL scripts.)

Rationale: This allows useful or supplemental material to be presented in a way that does not interfere with readability of the header comment block or the code.

- 11) All code comments should be written as complete sentences. Exceptions can be made for banner-type comments, headings for variable declarations, and end-of-line comments for variable declarations.

Rationale: Writing a comment in complete sentences should make the meaning clear and reduce the chance of misunderstanding by later readers.

5.0 References

None