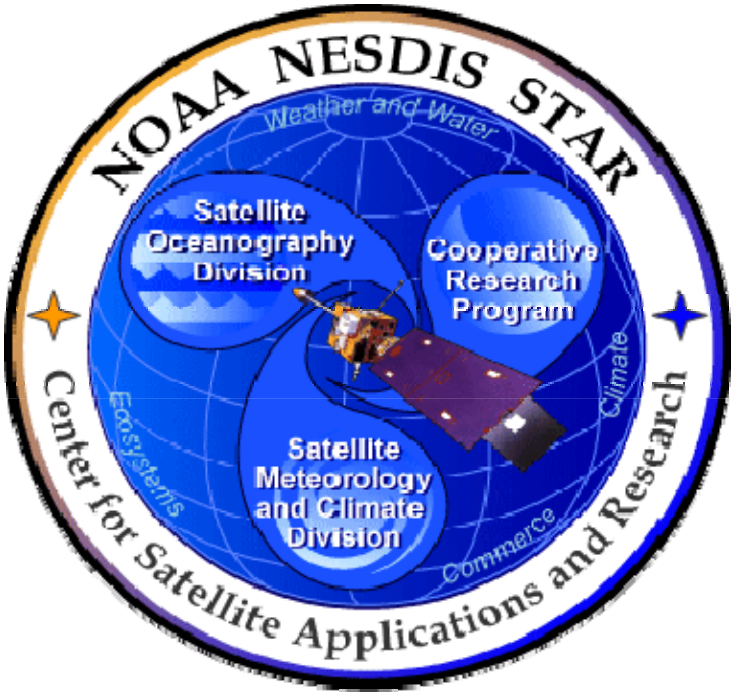

NOAA NESDIS
CENTER for SATELLITE APPLICATIONS and RESEARCH

TRAINING DOCUMENT

TD-11.1
FORTRAN PROGRAMMING STANDARDS and GUIDELINES

Version 3.0

TITLE: TD-11.1: FORTRAN PROGRAMMING STANDARDS AND GUIDELINES VERSION 3.0

AUTHORS:

Ken Jensen (Raytheon Information Solutions)

Alward Siyyid (Raytheon Information Solutions – version 1)

FORTRAN PROGRAMMING STANDARDS AND GUIDELINES
VERSION HISTORY SUMMARY

Version	Description	Revised Sections	Date
1.0	New Work Instruction (WI-12.1.1) adapted from Raytheon SOI 505 by Ken Jensen (Raytheon Information Solutions)	New Document	03/31/2006
1.1	Revision by Alward Siyyid (Raytheon Information Solutions). Added sections on error trapping (4.10) and interoperability (4.17).	4.10, 4.17	05/05/2006
1.2	Revision by Ken Jensen (Raytheon Information Solutions). Applied STAR standard style to entire document.	All	06/02/2006
2.0	Revision by Ken Jensen (Raytheon Information Solutions). Changed from WI-12.1.1 to Training Document TD-12.1.1 for version 2 of the STAR Enterprise Product Lifecycle (EPL). Numerous revisions in response to peer review comments. Quick reference by Walter Wolf (STAR) added to appendices.	All	09/28/2007
3.0	Renamed TD-11.1 and revised by Ken Jensen (RIS) for version 3.	1, 2	10/1/2009

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ACRONYMS

CICS	Cooperative Institute for Climate Studies
CIMSS	Cooperative Institute for Meteorological Satellite Studies
CIOSS	Cooperative Institute for Oceanographic Satellite Studies
CIRA	Cooperative Institute for Research in the Atmosphere
COTS	Commercial Off-The-Shelf
CREST	Cooperative Remote Sensing and Technology Center
EPL	Enterprise Project Lifecycle
FCD	Final Committee Draft
IEC	International Engineering Consortium
IMSL	International Mathematical and Statistical Library
INCITS	International Committee for Information Technology Standards
ISO	International Organization for Standardization
NESDIS	National Environmental Satellite, Data, and Information Service
NOAA	National Oceanic and Atmospheric Administration
PAR	Process Asset Repository
PG	Process Guideline
STAR	Center for Satellite Applications and Research
TD	Training Document

1. INTRODUCTION

The NOAA/NESDIS Center for Satellite Applications and Research (STAR) develops a diverse spectrum of complex, often-interrelated, environmental algorithms and software systems. These systems are developed through extensive research programs, and transitioned from research to operations when a sufficient level of maturity and end-user acceptance is achieved. Progress is often iterative, with subsequent deliveries providing additional robustness and functionality. Development and deployment is distributed, involving STAR, multiple cooperative institutes (CICS, CIMSS, CIOSS, CIRA, CREST) distributed throughout the US, multiple support contractors, and NESDIS operations.

NESDIS/STAR is implementing an increased level of process maturity to support the exchange of these software systems from one location or platform to another. The purpose of this coding standards guideline is to assist software developers reliably and repeatably develop, port, and deliver NOAA/NESDIS environmental software systems across platforms, locations, and organizations.

1.1. Objective

The objective of this Training Document (TD) is to provide STAR standards for Fortran code that is developed, tested, and reviewed during the STAR Enterprise Product Lifecycle (EPL)[1]. The intended users of this TD are programmers of Fortran code that will be used to implement an algorithm that creates an operational product from remote sensing satellite data. To achieve the objective, this TD shall:

- Establish Fortran programming standards for STAR, drawn from international standards
- Provide Fortran programming guidelines
- Provide examples of good Fortran programming practices
- Serve as a common reference for programming practices within the STAR Enterprise.

[1] For a description of the STAR EPL, refer to the STAR EPL Process Guidelines (PG-1 and PG-1.A).

1.2. Background

This TD defines programming standards and provides implementation guidelines for coding in the Fortran programming language for software programs within STAR. This TD has been adapted from Raytheon Software Operating Instruction 505 "FORTRAN [2] Programming Standards and Guidelines" and has benefited from review and comments by John Stroup, Sid Boukabara, Paul van Delst and Walter Wolf.

The development of code that will be used to implement an algorithm that creates an operational product from remote sensing satellite data is part of a unified STAR EPL. As such, it takes place in a series of defined steps:

Basic Research Code: In this step, a new or improved algorithm is being developed by a scientist. Usually, some coding is needed to implement the Basic Research algorithm so that the algorithm developer can do sufficient testing to determine whether the algorithm has operational potential. At the discretion of the Basic Research organization, Basic Research code can be prototype "throwaway" code that does not have to conform to standards, and there will be no code review at this step. If the programmer intends to reuse his Basic Research code in future steps, he should be aware that the reused code will be required to conform to standards.

Research Grade Code: In this step, the algorithm has been identified as having operational potential and additional development has been authorized to determine whether a STAR Research Project proposal should be submitted. Research grade code is a required artifact for the STAR review of a Project Proposal. STAR reviewers will expect that this code can be re-used in the development of pre-operational code. The conformance of the code to these standards may be a factor in STAR's decision to approve the project for development.

Pre-operational Code: In this step, the algorithm has been approved for development and has passed a Critical Design Review. The code is developed from research grade to pre-operational status. The conformance of the pre-operational code to the standards in this TD shall be a factor in a decision to approve its installation in an operations environment.

[2] Beginning with Fortran 77, it has been the convention to use Sentence Case instead of Upper Case for the name. Many Fortran documents retain the old convention.

Operational Code: In this step, the pre-operational code has been successfully integrated into the operational environment and is ready for approval for operations. There are no additional programming standards for operational code.

1.3. Fortran Versions

The Fortran programming language standards are maintained by the International Committee for Information Technology Standards (INCITS, http://www.incits.org/). The U.S. Fortran standards committee J3 (http://j3.incits.org/), an INCITS technical subcommittee, developed the Fortran 66, Fortran 77, Fortran 90, Fortran 95 and Fortran 2003 standards. Fortran 2003, published 18 November 2004, is an upwardly-compatible extension of Fortran 95, adding, among other things, support for exception handling, object-oriented programming, and improved interoperability with the C language. Working closely with ISO/IEC/JTC1/SC22/WG5 (http://www.nag.co.uk/sc22wg5/), the international Fortran standards committee, J3 is the primary development body for Fortran 2008. Fortran 2008 is planned to be a minor revision of Fortran 2003.

Fortran 90 represented a substantial revision from Fortran 77. Later revisions were significant, but not nearly as substantial.

This TD is tailored for STAR, based on the recognition that many Fortran programmers and potential reviewers in the STAR Enterprise are accustomed to the Fortran 77 version, and much of the legacy Fortran code is in the Fortran 77 style. While it is recommended that Fortran programming be in accordance with Fortran 90 or later standards when practical, it is not required. Fortran 77 code, when written according to its standards, is compatible with newer Fortran compilers. To assist those who would like to begin programming in later versions of Fortran, and those who will have to peer review Fortran 90 or later code, there is an Appendix to this TD (TD-11.1.A “Transition from Fortran 77 to Fortran 90”) available in the STAR EPL Process Asset Repository (PAR, c.f. Section 2).

Appendix A of this TD contains programming examples in Fortran 77 (Example A-1) and Fortran 90 (Example A-2).

1.4. Benefits

Code developed in accordance with the standards in this TD assists the programmers and testers by increasing the efficiency of code testing and debugging.

Code developed in accordance with the standards in this TD assists code reviewers by ensuring that the code presented for review is well documented, readable, and traceable to design.

Code developed in accordance with the standards in this TD makes it easier to perform code maintenance during operations.

Most important to the programmer, it is a STAR requirement that Fortran code be developed in accordance with the standards in this TD. Failure to do so may result in disapproval and the need to rewrite the code for a delta review.

1.5. Overview

This TD contains the following sections:

Section 1.0 -	Introduction
Section 2.0 -	Reference Documents
Section 3.0 -	Definitions
Section 4.0 -	Programming Standards and Guidelines
Appendix A -	Programming Examples
Appendix B -	STAR Fortran Coding Standards Quick Reference
Appendix C -	Transition from Fortran 77 to Fortran 90

2. REFERENCE DOCUMENTS

ISO/IEC FCD 1539-1:2004 is the international standard for Fortran code. This is a very large document that can be used as a reference at the programmer's discretion. This document is available at:

http://www.iso.org/iso/catalogue_detail.htm?csnumber=39691

Boukabara, S.-A. and P. van Delst (2007), *Standards, Guidelines and Recommendations for Writing FORTRAN 95 Code* provides the SPSRB standards, guidelines and recommendations for Fortran 95. The STAR standards and guidelines are intended to be completely consistent with the SPSRB standards. This document is available on the SPSRB web site at:

http://projects.osd.noaa.gov/spsrb/standards_docs/Fortran95_standard_rev22Jun2009.pdf

The following references are STAR EPL process assets that are accessible in a STAR EPL Process Asset Repository (PAR) on the STAR web site:

http://www.star.nesdis.noaa.gov/star/EPL_index.php.

TD-11.1.A: *Transition from Fortran 77 to Fortran 90*, an Appendix to this TD, is a useful training document for programmers and code reviewers who are not familiar with Fortran 90 and later versions of Fortran. This document is available to approved STAR EPL stakeholders in the STAR EPL PAR.

PG-1: STAR EPL Process Guideline provides the definitive description of the standard set of processes of the STAR EPL.

PG-1.A: STAR EPL Process Guideline Appendix, an appendix to PG-1, is a Microsoft Excel file that contains the STAR EPL process matrix (Stakeholder/Process Step matrix), listings of the process assets and standard artifacts, descriptions of process gates and reviews, and descriptions of stakeholder roles and functions.

3. DEFINITIONS

Main Program. A Fortran main program begins with the reserved word PROGRAM and ends with a matching reserved word END, and consists of a sequence of executable statements and optional declarations. A program is always a separately compilable unit.

Program Unit. A program unit is any of the three structures in Fortran, namely PROGRAM, SUBROUTINE, and FUNCTION.

Subprogram. A Fortran subprogram begins with either the reserved word FUNCTION, SUBROUTINE, MODULE, or BLOCK DATA and ends with a matching reserved word END. It consists of a sequence of executable statements and is called by a main program or by another subprogram. A subprogram may or may not be a separately compilable unit, depending upon implementation.

This section contains the STAR Fortran programming standards and associated guidelines.

3.1. Language Features

Only language features and capabilities that are documented or defined in the **ISO/IEC FCD 1539-1:2004(E) (c.f. Section 2)** shall be used. Use the -iso compiler flag to ensure this.

Fixed source form (Fortran 77) and free source form (Fortran 90 [3]) shall not be mixed within a subprogram. Program units written in fixed form and free form may be mixed in the same program, but each unit must be only in one form and at the compilation both forms may not be in the same source file.

If an algorithm is re-using a substantial amount of legacy Fortran 77 code, it is advisable to complete the code in the Fortran 77 (fixed form) style. If there is no re-use, it is recommended that new code be re-written in free form style. If the amount of re-use is small, it is recommended that new code be written in free form style and the legacy code be re-written in free form style. An exception to this guideline is the case where the designated programmers are not familiar with free form style and the scope of the project does not

[3] In this document, references to Fortran 90 are intended to apply to all later versions on Fortran (e.g. Fortran 95).

warrant the training cost. In this case, new code written in fixed form style shall be acceptable.

It is expected that a waiver will be required for approval of fixed form code. It is recommended that this waiver be obtained before a significant coding effort is expended.

One of the requirements for code to pass STAR reviews is that the code can be compiled on a standard Fortran compiler.

During the development and testing phases, all compiler options that provide additional checks of the code should be turned on.

3.2. Readability

Pagination:

Begin each program unit at the top a new page.

Continuation Statements:

Fortran 77: When a single statement extends to more than one (1) line, begin each continuation line with an ampersand (&) in column six (6), and indent two (2) levels of indentation relative to the first statement line.

Fortran 90: When a single statement extends to more than one (1) line, insert one blank character and an ampersand (" &") at the end of the line and begin the continuation line at an indentation of at least two spaces. Refer to lines 160-161, 187-188, 190-191, 206-207, 223-224, and 284-285 of Example A-2. Though a statement may have as many as 39 lines of continuation, it is recommended that long statements be partitioned into a number of smaller statements wherever possible. Try to limit the number of continuation lines to 10 or less.

Characters per line of code:

Fortran 77: A “line” of code cannot extend more than 80 columns.

Fortran 90: Use a maximum of 90 characters per line (maximum allowed under ISO is 132)

Alphabetic case shall be used consistently to enhance readability throughout a program.

GOOD:
```
**********
  C = A + B * X
**********
```

GOOD:
```
**********
  c = a + b * x
**********
```

BAD:
```
**********
  C = a + B * x
**********
```

Blocking with blank lines shall be used consistently to enhance readability throughout a program. A comment line shall be separated from a preceding executable line of code by a single blank line. All comment lines that are followed by an executable line of code should be separated from the executable line of code either by a single blank line or by no blank line. This is an optional matter of style that should be used consistently throughout a program.

GOOD:
```
****************************************
  !     Compute the sides of a right triangle

        a = x + 6
        b = y / 4.5

  !     Compute the square of the hypotenuse

        c_squared = a * a + b * b
****************************************
```

GOOD:
```
****************************************
  !     Compute the sides of a right triangle
        a = x + 6
        b = y / 4.5

  !     Compute the square of the hypotenuse
        c_squared = a * a + b * b
 ****************************************
```

BAD:
```
****************************************
  !     Compute the sides of a right triangle
        a = x + 6
        b = y / 4.5
  !     Compute the square of the hypotenuse
        c_squared = a * a + b * b
 ****************************************
```

BAD:
```
****************************************
  !     Compute the sides of a right triangle

        a = x + 6
        b = y / 4.5

  !     Compute the square of the hypotenuse
        c_squared = a * a + b * b
 ****************************************
```

Compound Expressions: Place spaces before and after relational operators, Fortran reserved words, identifiers, and arithmetic operators to enhance readability of compound expressions. Refer to example A-1, line 236.

Input/Output: In addition to identifying input and output variables in the Preamble, it is helpful for readability and clarity to separate Input, Output, and Processing functions in a

program so that all Input functions precede all Processing functions, followed by all Output functions. Exceptions to this rule occur when memory constraints require dynamic allocation of memory within the processing function. When dynamic allocation is used, input and output functions within processing functions should be clearly identified by comments that identify the input/output variables with references to the Preamble and/or design documents.

Indentation shall be used consistently to enhance readability throughout a program. Each indentation should use at least two spaces. A comment line should be indented in the same way as the following executable line of code. Statements in nested loops should be indented so that all statements in the same nesting are indented by the same amount. Statements in inner nested loops should be indented by a greater amount than statements in outer nested loops.

GOOD:
```
!       Loop over values of x and y
        DO i=1,5
                x = x_value(i)
                DO j=1,4
                        y = y_value(j)

                !       Compute the sides of a right triangle
                        a = x + 6
                        b = y / 4.5

                !       Compute the square of the hypotenuse
                        c_squared(i,j) = a * a + b * b

        !       Close the loops
                END DO
        END DO
```

BAD:
```
!       Loop over values of x and y

                DO i=1,5
                x = x_value(i)
```

```
                    DO j=1,4
                    y = y_value(j)

!                       Compute the sides of a right triangle
                            a = x + 6
                        b = y / 4.5

!           Compute the square of the hypotenuse
                        c_squared(i,j) = a * a + b * b

!                       Close the loops
                    END DO
                        END DO
*****************************************************
```

3.3. Naming Conventions

There is no standard for Fortran naming conventions, as Fortran code will work with all names composed of recognized characters. That is why they are called naming “conventions”, not naming “standards”. The only required standard is for what names can NOT be: Names can not be identical to Fortran reserved words or implementation supplied function names.

In addition, naming conventions within a programming community are under continual development, as programmers communicate with each other and agree to adopt particular conventions.

When writing code, the names of files, subroutines, functions and variables created by a programmer are always up to the programmer. A programmer can choose to make the names long or short, descriptive or useless, clear or confusing. A lot depends on the mindset of the programmer. Will this code be reused? Is this "quick and dirty" code? Is this code so clear that it is self-explanatory? The names of files, subroutines, functions and variables can be extremely useful in making code more readable. Choosing names may seem not very important, but insisting on meaningful names helps a programmer to organize thoughts and produce code that is readable and reviewable.

Avoid names that look alike by differing only in characters that resemble each other, as do 2 and z, 0 and O, 5 and S, or I and 1.

Name programs, subroutines, and functions to indicate purpose. Familiarize yourself with the STAR Common Library of Fortran routines. This serves two main purposes: 1) You may find a library routine that you can use to implement your desired function, 2) You should avoid using names that are similar to library routines.

Name symbolic variables to indicate what they are, not what values they may contain.

Names should be as mnemonically descriptive as possible, subject to constraints imposed by Fortran standards.

Names shall not be identical to Fortran reserved words or implementation supplied function names. Names should not resemble Fortran reserved words or implementation supplied function names.

3.4. Compound Expressions

The evaluation of logical and arithmetic expressions shall be clarified through the use of parentheses and spaces.

GOOD:
```
****************************************
 pk = pk - 1.0 + ( 0.5 * REAL(ning) )
****************************************
```

BAD:
```
****************************************
 pk = pk - 1.0 + 0.5 * REAL(ning)
****************************************
```

The nesting of parentheses in logical and arithmetic expressions shall be limited to four (4) levels. If an expression requires a greater level of nesting, it shall be separated into more than one expression.

GOOD:
```
******************************************************
 mu_s = COS(pi*sza/180.)
 mu_v = COS(pi*sva/180.)
 tan_s = TAN(pi*sza/180.)
 tan_v = TAN(pi*sva/180.)

 d = SQRT(tan_s * tan_s + tan_v * tan_v - &
 2.0 * tan_s * tan_v * COS(pi*relaz/180.))

 fac = tan_s * tan_v * SIN(pi*relaz/180.)
```

```
            cost = SQRT(d * d + fac * fac)/ ((1./mu_s) + (1./mu_v))

           ******************************************************
```

BAD:

```
           ******************************************************
            mu_s = COS(pi*sza/180.)
            mu_v = COS(pi*sva/180.)
            tan_s = TAN(pi*sza/180.)
            tan_v = TAN(pi*sva/180.)

            cost = SQRT((tan_s * tan_s + tan_v * tan_v - &
            2.0 * tan_s * tan_v * COS(pi*relaz/180.) + &
            (tan_s * tan_v * SIN(pi*relaz/180.) *  tan_s * &
            tan_v * SIN(pi*relaz/180.)))) / ((1./mu_s) + (1./mu_v))
           ******************************************************
```

3.5. Preamble

Every new Fortran program unit shall contain a preamble. Designate information required in the preamble with the following keywords:

a. NAME: The name of the program unit.

b. FUNCTION: A brief description of the program unit function (e.g., 1-2 sentences).

c. DESCRIPTION: A description of the program unit processing (e.g., diagrams, PDL).

d. REFERENCE: The reference(s) to program unit design materials (e.g., requirements document, design document, standards, algorithm decisions).

e. CALLING SEQUENCE: The source statements necessary to invoke the program unit.

f. INPUTS: A description of the program unit inputs (e.g., parameters, files).

g. OUTPUTS: A description of the program unit outputs (e.g., parameters, files).

h. DEPENDENCIES: A description of the program unit dependencies (e.g., HW/SW dependencies, INCLUDE files, operating systems, initialization).

i. SIDE EFFECTS: A description of the program unit side effects.

j. HISTORY: The revision history of the program unit.

Refer to examples A-1 (lines 002 – 117) and A-2 (lines 001 – 112).

The requirement for a preamble can be waived for re-used legacy Fortran programs, but it is recommended that a preamble be added to these programs.

3.6. Organization

Elements of the program units shall include the following and shall be organized as shown:

a. program unit identifier,
b. preamble,
c. INCLUDE files,
d. specification statements,
e. DATA statements,
f. statement function statements,
g. executable statements,
h. EXIT statement, and
i. END statement.

3.7. Size

It is recommended that each program unit is kept as small and simple as possible to perform a specific task. Use multiple, smaller routines with well-defined functions rather than a larger routine that does a lot of things. Program units containing more than 200 lines of code should be examined to see if they can be segmented.

3.8. Declarations

Fortran 77: Align each declaration type name in column seven (7). Refer to example A-1, lines 127, 131, 134 and 136.

Avoid continuation lines in a declaration statement by using multiple statements. Refer to example A-2, lines 118-121.

List several variables of a single type on a line alphabetically. Refer to example A-2, lines 118-121.

Use named common blocks for all global externally referenced common data.

Explicitly dimension all arrays. Use parameters as much as possible to specify array dimensions/sizes. Use of dynamic memory allocation is encouraged.

3.9. Error Trapping

The Fortran programmer is encouraged to read applicable compiler and operating system documentations and consult http://www.nag.co.uk/sc22wg5/ for the latest concerning Fortran standards (see section 1.3) relating to error trapping. General guidelines regarding this topic are as follows:

- Check for error return values, even from functions that "can't" fail. It is recommended that the following convention be used for error return values:
  - A value of zero indicates the function completed successfully
  - A negative value indicates the function failed
  - A positive value indicates the function completed successfully but encountered something unexpected.
- Include the system error text for every system error message.
- Take special care with I/O statements since these are usually affected by events beyond the control of the programmer. Include an item of the form ERR=label which causes control to be transferred to the statement attached to that label in the event of an error. This must, of course, be an executable statement and in the same program unit. For example:

```
READ(UNIT=IN, FMT=*, ERR=999) VOLTS, AMPS
WATTS = VOLTS * AMPS
rest of program in here . . . . . and finally
STOP
```

```
        WRITE(UNIT=*,FMT=*)'Error reading VOLTS or AMPS'
    END
```

- Similarly, handle the end-of-file condition when reading beyond the end of a sequential or internal file. If an item of the form: END=label (as opposed to ERR=999 in the above example) then control is transferred to the labeled statement when the end-of-file condition is detected.
- The END= keyword may only be used in READ statements, but it can be used in the presence of both ERR= and IOSTAT= keywords. End-of-file detection is very useful when reading a file of unknown length.

3.10. Statement Numbers (Fortran 77)

Avoid referencing statement numbers in comments (Fortran 77).

Consistently justify statement numbers in columns two (2) through five (5) in ascending order throughout a program.

3.11. Subroutine Control

To retain the value of a variable in a program unit after control is returned from that unit, use a SAVE attribute or SAVE statement. Do not rely on the compiler to retain the variable's value for you. This will help make it clear to anyone reading the code that the value of the variable is being retained between calls to the routine.

The use of an alternate return specifier as an argument in a calling sequence in the event of an error (e.g., "CALL foo (a, b, *999)") is highly discouraged.

3.12. Statements

Each statement shall begin on a separate line.

Nesting of statements shall be limited to five (5) levels.

All variables shall be declared using a type-statement, INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, and CHARACTER. The KIND statement may be used to specify the type. Variables of derived types (Fortran 90) may be declared as structures.

Computed GO TO (Fortran 77) or ELSE IF statements shall be acceptable in lieu of a CASE statement. However, out of range conditions shall be included in the statement.

The use of COMMON blocks and INCLUDE files is discouraged. Use global variables or modules instead of COMMON blocks. If COMMON blocks are used, the following practices should be followed:

- All variables in a common block should be named the same in every program unit that uses the common data.
- Common blocks should be declared in a separate file and copied into the source file using the INCLUDE statement.
- For efficiency, variables in the COMMON statement should be aligned on their proper word and byte boundaries. This generally means listing the variables in descending order according to their data type size: quad-word variables first, then double-word variables, word variables, half-word variables, and finally single byte variables.

All loops shall terminate with a unique CONTINUE statement (Fortran 77)

Each GO TO statement shall target a unique CONTINUE statement (Fortran 77).

The DO statement shall not contain any statements that change the value of the loop-controlling variable.

GO TO statements (Fortran 77) shall be used only where required to meet specific execution time, space constraints, or to reduce unnecessary complexity. Every GO TO statement shall be accompanied by comments placed near the GO TO statement to document the applicable constraints and comments placed near the statement receiving control to document the origin of the transfer of control. GO TO statements shall not be used to transfer control outside loops. The use of GO TO statements in new Fortran 77 code is discouraged. GO TO statements should not be used in Fortran 90 code.

Assigned GO TO statements (Fortran 77) shall not be used.

3.12.1. IMPLICIT NONE Statement

All program units should include the "IMPLICIT NONE" statement and should be compiled with the option so the compiler flags any variables that are not explicitly declared.

3.12.2. DO Statement

Indent the statements following the Fortran DO statement one (1) level of indentation.

Fortran 77: Format the DO statement as follows:

```
            DO  <stmt#>  <expression>
                statements
<stmt#>     CONTINUE
```

Refer to example A-1, lines 161-173.

Fortran 90: Format the DO statement as follows:

```
            DO  <expression>
                statements
            END DO
```

Refer to example A-2, lines 216-218.

3.12.3. FORMAT Statement

Fortran 77: Accompany each READ and WRITE statement with the corresponding FORMAT statement. Example:

```
      READ(5,10) SIZE
10    FORMAT(I5)
```

Fortran 90: Specify format with the FMT specifier in the argument list of a READ or WRITE statement. Example:

```
   READ (UNIT=15,FMT='I5',IOSTAT=io) size
```

3.12.4. IF Statement

Enclose the condition(s) following the Fortran reserved word IF in parentheses. Refer to example A-1, lines 315-318 and example A-2, line 193.

Indent the statements following the IF, ELSE IF, and ELSE statement, one (1) level of indentation. Refer to example A-1, lines 266-272 and example A-2, lines 316-326.

Fortran 77: Format the Arithmetic IF statement as follows:

```
        IF (  <expression>  )  s#1, s#2, s#3
C
 s#1    CONTINUE
        <statements>
        GO TO s#4
C
 s#2    CONTINUE
        <statements>
        GO TO s#4
C
 s#3    CONTINUE
        <statements>
C
 s#4    CONTINUE
```

The ELSE IF statement can be used in lieu of an Arithmetic IF statement if so desired.

Format the IF statement as follows:

```
        IF (  <expression>  )  THEN
          statements
        END IF
```

Refer to example A-1, lines 290-307 and example A-2, lines 178-185.

Format the ELSE statement as follows:

```
        IF (  <expression>  )  THEN
          statements
        ELSE
```

```
      statements
     END IF
```

Refer to example A-1, lines 266-272 and example A-2, lines 316-326.

Format the ELSE IF statement as follows:

```
     IF (  <expression>  )  THEN
      statements
     ELSE IF (  <expression>  )  THEN
      statements
     ENDIF
```

Refer to example A-1, lines 309-319 and example A-2, lines 234-253..

Fortran 77: Format the computed GO TO statement as follows:

```
     GO TO ( s#1, s#2, s#3 ) i
C
 s#1    CONTINUE
        <statements>  )
        GO TO s#4
C
 s#2    CONTINUE
        <statements>  )
        GO TO s#4
C
 s#3    CONTINUE
        <statements>  )
C
 s#4    CONTINUE
```

The ELSE IF statement can be used in lieu of the Computed GO TO statement if so desired.

3.12.5. SAVE Statement

To retain the value of a variable in a program unit after control is returned from that unit, use a SAVE attribute or SAVE statement. Do not rely on the compiler to retain the variable's value for you. By explicitly putting the variable in a SAVE statement, it will help make it clear to anyone reading the code that the value of the variable is being retained between calls to the routine.

3.12.6. EXIT/CYCLE Statement

Fortran 90: Use an EXIT/CYCLE statement to exit/cycle loops

3.12.7. EQUIVALENCE Statement

The use of EQUIVALENCE statements is discouraged. The TRANSFER intrinsic function should be used instead of EQUIVALENCE statements.

3.12.8. END Statement

An END statement shall be used as the last statement for all functions, subroutines, modules and programs.

The last statement in a DO loop should be an END DO statement. If there are a large number of statements in a DO loop, include a comment that relates the END statement to the DO statement for that loop.

The last statement in an IF loop should be an END IF statement. If there are a large number of statements in an IF loop, include a comment that relates the END statement to the IF statement for that loop.

3.13. Common Libraries

Use the IMSL Fortran Numerical Library. This library is usually available from suppliers of Fortran compilers. The IMSL Fortran Library, a complete collection of mathematical and statistical algorithms for high performance computing applications, integrates the IMSL F90 Library with the IMSL Fortran 77 library into a single, cohesive package. The IMSL Fortran Library includes all of the algorithms from the IMSL Family of Fortran libraries for more than

three decades, including the IMSL F90 Library, the IMSL Fortran 77 Library, and the IMSL parallel processing features.

3.14. Use of Standard Constants

Use standard mathematical and geophysical constants (e.g. PI).

3.15. Efficient Use of Memory

Use dynamic allocation of memory wherever possible. Avoid COMMON blocks. Do not allocate memory for local variables until they are used in a subprogram, and deallocate the memory for a local variable as soon as its use in the program is finished. This is especially important when handling large, multi-dimensional arrays.

Example:

```
REAL(KIND=4),ALLOCATABLE :: variable5(:,:,:), variable6(:,:,:)

<statements preceding use of variable “variable5”>
ALLOCATE( variable5(n_dimension_1,n_dimension_2,n_dimension_3) )
<statements using variable5>
DEALLOCATE( variable5 )

<statements preceding use of variable “variable6”>
ALLOCATE( variable6(n_dimension_1,n_dimension_2,n_dimension_3) )
<statements using variable6>
DEALLOCATE( variable6 )
```

3.16. Fortran/C Interoperability

Fortran provides a standard mechanism (ISO_C_BINDING) for interoperating with C. Fortran and C entities should be declared equivalently. Reader is encouraged to read applicable compiler and operating system documentations and consult http://www.nag.co.uk/sc22wg5/ for the latest concerning Fortran standards (see section 1.3) and interoperability issues. General considerations are summarized in Table 1.

Table 1. Fortran/C Interoperability

Language Entity	Description
Data types	ISO_C_BINDING module defines named constants for the intrinsic data types. Some examples are : Type — Named constant — C type or types INTEGER — C_INT — int, signed int C_LONG — long int, signed long int REAL — C_FLOAT — float C_DOUBLE — double CHARACTER — C_CHAR — char
Pointers	For interoperating with C pointers, the module contains a derived type C_PTR that is interoperable with any C pointer type and a named constant C_NULL_PTR with the value NULL of C.
Derived types	Derived types are given the BIND attribute explicitly: TYPE, BIND(C) :: MYTYPE : END TYPE MYTYPE E.g. typedef struct { int m, n; float r; } ctype is interoperable with USE ISO_C_BINDING TYPE, BIND(C) :: FTYPE INTEGER(C_INT) :: I, J REAL(C_FLOAT) :: S

	END TYPE FTYPE
Arrays	Array variables must be of explicit shape and size. However, indices are reversed. Example: FORTRAN: INTEGER :: A(30, 3:7, *) C: int b[][5][30]
Procedures And Subroutines	A Fortran procedure is interoperable if it is declared with the BIND attribute: FUNCTION FUNC(I, J, K, L, M), BIND(C, NAME='C_Func') Such a procedure corresponds to a C function prototype with the same binding label. Fortran function result must be interoperable. Fortran subroutine must have a void result.
Common Block	An interoperable module variable or a common block with interoperable members may be given the BIND attribute: USE ISO_C_BINDING INTEGER(C_INT), BIND(C) :: C_EXTERN INTEGER(C_LONG) :: C2 BIND(C, NAME='myVariable') :: C2 COMMON /COM/ R, S REAL(C_FLOAT) :: R, S BIND(C) :: /COM/ It has a binding label defined by the same rules as for procedures and interoperate with a C variable of a corresponding struct type.

3.17. Documentation

Comments should be used on a regular basis throughout a program. As a general rule, there should be no more than 10 – 15 executable lines of code without a comment. The comment should be descriptive enough to explain the function of the following block of code. Comments preceding a block of code that initiates a major program function should be referenced to the description in the preamble and/or the design documents. In particular, a block of code that initiates a data flow identified in the algorithm's software architecture should be preceded by a comment that refers to the data flow identifier.

Fortran 77: Delimit comments consistently, with a 'C' in column one (1) and a blank line before and after the comment line. See example A-1 in the appendix.

Fortran 90: Delimit comments consistently, using a '!' character preceding the comment. Comments customarily occupy a distinct line ('comment line') separated from executable lines of code by blank lines. With Fortran 90 and later versions, it is possible to include executable statements and comments on the same line. This practice can be helpful in identifying the comment with the relevant executable statement, in cases where the comment applies to a single executable statement. When this is done, the line should be preceded and followed by blank lines, to improve readability. See example A-2 in the appendix.

Precede each major section of code within a program unit with a block comment briefly describing the processing involved.

Precede each conditional statement (e.g., DO, IF, Computed GO TO) with a block comment describing the condition being tested and the branching alternatives. Refer to example A-1, lines 228-230.

Accompany each variable declaration with a comment. Refer to example A-1, lines 57-107 and example A-2, lines 54-110.

Accompany each program unit END statement with a comment indicating the program unit name. Refer to example A-1, lines 385-389 and example A-2, lines 346-349. Alternatively, include the name of the program unit in the END statement (see example A-2, line 349).

3.18. Grandfathering

This section explains what can be excluded from these Programming Standards and Guidelines.

3.18.1. COTS

Commercial Off The Shelf (COTS) software currently in use is grandfathered and does not have to comply with the standards and guidelines documented in this TD. If adding additional functionality to COTS software, consider implementing the standards and guidelines documented in this TD wherever possible.

3.18.2. Reuse

Software reuse from a common Product Line baseline or any other STAR baseline is grandfathered and does not have to comply with the standards and guidelines documented in this TD.

STAR-unique Software Components that are developed for use with the reuse software shall follow the standards and guidelines in this TD.

STAR-unique Software Units that are developed to integrate with reuse software shall follow the standards and guidelines in this TD, if possible, given the reuse software architecture and reuse software standards involved.

Newly developed STAR software that is deemed to be generic in nature and suitable for addition to the reuse software baseline will follow the standards and guidelines established for the reuse software.

APPENDIX A. FORTRAN PROGRAMMING EXAMPLES

This section contains examples that clarify and are referenced by the standards and guidelines in section 4.0.

Example A-1. Subroutine MLTGEN (Fortran 77)

```
001       SUBROUTINE MLTGEN
002 C
003 C$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
004 C
005 C NAME: Template Generation
006 C
007 C FUNCTION:
008 C    This process transform the list of sets of 3D points defined
009 C    in  the target  reference coordinate system into the 2D image
010 C    plan.  The 3D data is contained in the object point data base,
011 C    the 2D transformed data is contained into the screen coordinate
012 C    data base.
013 C
014 C DESCRIPTION:
015 C  - Construct the direction COSINE matrix and the altitude
016 C  - Initialize screen coordinate data base
017 C  - Determine the bin selector index
018 C  - Program the image screen programion points for each zone onto
019 C    the ground-seeker coordinates, with y and z coordinates scaled
020 C    by the focal length.
021 C  - Prepare the transformation matrix
022 C  - Transform the verticies
023 C  - Determine the number of polygons to be used in zone
024 C  - Program verticies into the image plane.
025 C  - Transform the aimpoint
026 C  - Program the aimpoint into the image plane
027 C  - Calculate the minimum ploygon size and area
028 C  - Calculate the minimum background/target ration
029 C
030 C CALLING SEQUENCE: CALL MLTGEN
031 C
032 C INPUTS: NONE
033 C
034 C OUTPUTS: NONE
035 C
036 C REFERENCES: Spec # 2410820, CSCI for MGFV Program
037 C
038 C DEPENDENCIES:
039 C     SUBPROGRAMS: H:MKMTML.HDR, H:MLGDMA.HDR,
040 C             H:MLGPIX.HDR,H:ERLG.HDR
041 C     COMMON BLOCKS: H:CLOALP.FCU, H:CLOPDB.FCY
042 C
043 C SIDE EFFECTS: NONE
```

```
044 C
045 C TARGET PROCESSOR: VHSIC 1750A
046 C-------------------------------------------------------------
047 C HISTORY:
048 C
049 C     D: 09/04/86       C: Modified compiler directives for
050 C        use with ACT FORTRAN compiler.
051 C
052 C     D: 10/28/86       C: Removed reference to CFTRIG
053 C        and replaced FTR180 with local constant PI.
054 C
055 C VARIABLES : DESCRIPTION
056 C
057 C        ALT: real, scalar, altitude
058 C        DCM: real, array(1..3,1..3), direction cosine matrix
059 C      TRANS: real, array(1..3,1..3), transformation matrix
060 C        LXB: real, scalar, effective x-axis focal length
061 C        LYB: real, scalar, effective y-axis focal length
062 C          G: real, scalar, G
063 C         DN: real, scalar, denominator used in calculating G
064 C         DX: real, array(1..6), X distance from sensor to target
065 C         DY: real, array(1..6), Y distance from sensor to target
066 C         DZ: real, array(1..6), Z distance from sensor to target
067 C         S7: real, scalar, sine of aspect 1 angle
068 C         C7: real, scalar, cosine of aspect 1 angle
069 C      ASPA1: real, scalar, aspect 1 angle
070 C          X: real, scalar, X vrtx coord prior to xfrm.
071 C          Y: real, scalar, Y vrtx coord prior to xfrm.
072 C          Z: real, scalar, Z vrtx coord prior to xfrm.
073 C         XJ: real, array(1..18,1..6), X vrtx coord. after xfrm.
074 C         YJ: real, array(1..18,1..6), Y vrtx coord. after xfrm.
075 C         ZJ: real, array(1..18,1..6), Z vrtx coord. after xfrm.
076 C       XJSA: real, array(1..18), sngl adj check after xfrm.
077 C       YJSA: real, array(1..18), sngl adj check after xfrm.
078 C       ZJSA: real, array(1..18), sngl adj check after xfrm.
079 C      DENOM: real, scalar, denominator used in misc calc
080 C          I: integer, scalar, loop index
081 C          J: integer, scalar, loop index
082 C         J1: integer, scalar, loop index
083 C         J2: integer, scalar, loop index
084 C        INX: integer, scalar, local parameter index
085 C      EROVF: logical, scalar, error overflow flag
086 C      ERDYN: logical, scalar, error dynamic range
087 C       XMAX: integer, scalar, max proj x value, temp value
088 C       YMAX: integer, scalar, max proj y value, temp value
089 C       XMIN: integer, scalar, min proj x value, temp value
090 C       XMIN: integer, scalar, min proj y value, temp value
091 C         IX: integer, scalar, proj x coordinate, temp value
092 C         IY: integer, scalar, proj y coordinate, temp value
093 C      XMAXZ: integer, array(1..6), max proj x for each zone
094 C      XMINZ: integer, array(1..6), min proj x for each zone
095 C      YMAXZ: integer, array(1..6), max proj y for each zone
096 C      YMAXZ: integer, array(1..6), min proj y for each zone
097 C         LX: integer, scalar, proj polygon x-axis length
```

```
098 C         LY: integer, scalar, proj polygon y axis length
099 C         FM: real, array(1..3,1..3), array used in calc TRANS
100 C        HLX: real, scalar, horizon location, x axis
101 C      COUNT: integer, scalar, count # of polyhedrons selected
102 C        AMG: integer, scalar, amg segment selected counter
103 C      XLATE: integer, array(1..6), translation bet image screen
104 C             programion point and zone target reference point
105 C
106 C         PI: Real, scalar, 3.141593 radians.
107 C
108 C SUBPROGRAMS:
109 C============================ subpgm ==========================
110 C
111 C           INCLUDE 'H:MKMTML.HDR'
112 C           INCLUDE 'H:MLGDMA.HDR'
113 C           INCLUDE 'H:MLGPIX.HDR'
114 C           INCLUDE 'H:ERLG.HDR'
115 C
116 C$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
117 C
118       IMPLICIT NONE
119 C
120       INCLUDE   'H:CLOALP.FCY'
121       INCLUDE   'H:CLOPDB.FCY'
122 C
123 C*****************************************************************
124 C                   LOCAL VARIABLE DECLARATIONS                  *
125 C*****************************************************************
126 C
127       REAL ALT,DCM(3,3),LXB,LYB,G,DN,DX(6),DY(6),DZ(6),
128      &     TRANS(3.3),S7,C7,ASPA1,X,Y,Z,XJ(18,8),YJ(18,8),ZJ(18,8),
129      &     XJSA(18),YJSA(18),ZJSA(18),DENOM,
130      &     FM(3,3),HLX,PI
131       INTEGER I,J,J1,J2,INX,XMAX,XMIN,YMAX,YMIN,IX,IY,COUNT,AMG,
132      &        XMAXZ(6),XMINZ(6),YMAXZ(6),YMINX(6),LX,LY,
133      &        XLATE(6)
134       LOGICAL EROVF,ERDYN
135 C
136        DATA PI /3.141593/
137 C
138 C/EJECT
139 C
140 C*****************************************************************
141 C                           EXECUTABLE CODE                      *
142 C*****************************************************************
143 C
144 C Construct the direction Cosine matrix and the altitude
145 C
146       CALL MLGDMA(SLNUMB,DCM,ALT)
147 C
148 C  Initialize screen coordinate data base
149 C
150       LSSNUM = SLNUMB
151       LSFLDC = SLFCNT
```

```
152       LSSAST = LOSAST
153 C
154 C  Determine th Bin Selector Index
155 C
156       CALL MLGPIX(LSSNUM,INX)
157 C
158 C  Program image screen programion points for each zone onto
159 C  ground-seeker coordinates, with Y & Z coord scaled by focal length
160 C
161       DO 100 I = 1,LZONES
162         LXB = TQISPP(I) - SCCENX
163         LYB = LZTRPY(I) - SCCENY
164         DN =  DCM(1,3) + DCM(2,3) * LYB / SCCCON
165      &      + DCM(3,3) * LXB / SCACON
166         G = TQZALT(I) / DN
167         DX(I) = G
168         DY(I) = G * LYB
169         DZ(I) = G * LXB
170         LSDISX(I) = DX(I)
171         LSDISY(I) = DY(I)
173  100  CONTINUE
174 C
175 C Prepare the Transformation matrix
176 C
177 C   The following matrix will rotate target coordinate to align
178 C   with the navigation coordinates.
179 C
180       IF (LOAPAF .EQ. 0) THEN
181         LOCAPA(INX) = LPAPRA
182       ELSE
183         LOCAPA(INX) = SLHEAD - LPTAAN + PI
184       ENDIF
185 C
186       ASPA1 = LOCAPA(INX) + LOASAC(INX)
187       S7 = SIN(ASPA1)
188       C7 = COS(ASPA1)
189 C
190       TRANS(1,1) = -C7
191       TRANS(1,2) =  S7
192       TRANS(1,3) = KR0000
193       TRANS(2,1) = S7
194       TRANS(2,2) = C7
195       TRANS(2,3) = KR0000
196       TRANS(3,1) = KR0000
197       TRANS(3,2) = KR0000
198       TRANS(3,3) = -KR0001
199 C
200 C    The DCM transforms navigation coordinates to seeker
201 C    coordinates.  Scaling Z and Y coordinates by focal length
202 C    will simplify calc when points are programed into the
203 C    image plane (IMAGE COORDINATES).
204 C           |1 0  0 |
205 C     FM := |0 FY 0 |   *  DCM
206 C           |0 0  Fx|
```

```
C
C     FM(1,1) = KR0001
C     FM(1,2) = KR0000
C     FM(1,3) = KR0000
C     FM(2,1) = KR0000
C     FM(2,2) = SCCCON
C     FM(2,3) = KR0000
C     FM(3,1) = KR0000
C     FM(3,2) = KR0000
C     FM(3,3) = SCACON
C
C     CALL MKMTML(FM,FM,DCM)
C
      DO 175 I = 1,3
        DO 150 J = 1,3
         LSTRAN(I,J) = FM(I,J)
 150    CONTINUE
 175  CONTINUE
C
      CALL MKMTML(TRANS,FM,TRANS)
C
C Transform the verticies, the following process changes coord
C values for the polyhedron form target coordinates to seeker
C coordinates.
C
      COUNT = 0
      AMG = 0
C
      DO 250 I = 1,LONPOL
        IF (IOR (LOSMPF(I,INX), LOAMPF(I,INX)) .NE. 0) THEN
          COUNT = COUNT + 1
          LSSMPF(COUNT) = LOSMPF(I,INX)
          LSAMPF(COUNT) = LOAMPF(I,INX)
          AMG = AMG + LSAMPF(COUNT
          LSSEGT(COUNT) = LOSEGT(I)
          LSNVER(COUNT) = LONVER(I)
          DO 200 j = 1,LONVER(I)
            X = LOVERX(J,I)
            Y = LOVERY(J,I)
            Z = LOVERY(J,I)
            XJ(J,COUNT) = X * TRANS(1,1) + Y * TRANS(1,2)
     &                  + Z * TRANS(1,3)
            YJ(J,COUNT) = X * TRANS(2,1) + Y * TRANS(2,2)
     &                  + Z * TRANS(2,3)
            ZJ(J,COUNT) = X * TRANS(3,1) + Y * TRANS(3,2)
     &                  + Z * TRANS(3,3)
 200      CONTINUE
        ENDIF
 250  CONTINUE
      IF (AMG .NE. 0) THEN
        LOAMGS(INX) = .TRUE.
      ELSE
        LOAMGS(INX) = .FALSE.
      ENDIF
```

```
C
C  Determine the number of polygons to be used in zone
C
      DO  300 I = 1,LZONES
        HLX = -SCACON * (DCM(1,3) / DCM(3,3) + SCCENX
        IF ((LZSTRR(I) - HLX) .GT. LOMINX) THEN
          LSNPOL(I) = COUNT
          LSNVSA(I) = 0
        ELSE
          LSNPOL(I) = 0
          LSNVSA(I) = 0
        ENDIF
 300  CONTINUE
      EROVF = .FALSE.
      ERDYN = .FALSE.
C
C Program vert into image plane- for each zone & vertex of each
C polyhedron is translated around the target reference pt that
C was programed onto the ground, then the translated pt is programed
C into the image plane.  ONce ther vertex is programed, it is
C translated around the target reference point for the zone.
C
      DO 550 i = 1,LZONES
        XLATE(I) = LZTRPX(I) - TQISPP(I)
C
        DO 500 J1 = 1,LSNPOL(I)
          DO 400 J2 = 1,LSNVER(JI)
            DENOM = DX(I) + XJ(J2,J1)
            IX = (DZ(I) + ZJ(J2,J1)) / DENOM + SCCENX + LXLATE(I)
            IF (IX .GT. LOUPBX) THEN
              IX = LOUPBX
              EROVF = .TRUE.
            ENDIF
            IF (IX .LT. LOLWBX) THEN
              IX = LOLWBX
              EROVF = .TRUE.
            ENDIF
            LSVERX(J2,J1,I ) = IX
             IY = (DY(I) + YJ(J2,J1) / DENOM + SCCENY
             IF (IY .GT. LOUPBY) THEN
               IY = LOUPBY
               EROVF = .TRUE.
             ENDIF
             IF (IY .LT. LOLWBY) THEN
               IY = LOLWBY
               EROVF = .TRUE.
             ENDIF
             LSVERY(J2,J1,I) = IY
             IF (J2 .EQ. 1) THEN
               XMAX = IX
               YMAX = IY
               XMIN = IX
               YMIN = IY
             ELSE
```

```
315                IF (IX .GT. XMAX) XMAX = IX
316                IF (IX .LT. XMIN) XMIN = IX
317                IF (IY .GT. YMAX) YMAX = IY
318                IF (IY .LT. YMIN) YMIN = IY
319              ENDIF
320  400        CONTINUE
321 C
322            IF ((XMAX - XMIN) .GT. 2047) ERDYN = .TRUE.
323            IF ((YMAX - YMIN) .GT. 2047) ERDYN = .TRUE.
324            IF (J1 .EQ. 1) THEN
325              XMAXZ(I) = XMAX
326              YMAXZ(I) = YMAX
327              XMINZ(I) = XMIN
328              YMINZ(I) = YMIN
329            ELSE
330              IF (XMAX .GT. XMAXZ(I)) XMAXZ(I) = XMAX
331              IF (XMIN .GT. XMINZ(I)) XMINZ(I) = XMIN
332              IF (YMAX .GT. YMAXZ(I)) YMAXZ(I) = YMAX
333              IF (YMIN .GT. YMINZ(I)) YMINZ(I) = YMIN
334            ENDIF
335  500     CONTINUE
336 C
337          LSXMAX(I) = XMAXZ(I)
338          LSYMAX(I) = YMAXZ(I)
339          LSXMIN(I) = XMINZ(I)
340          LSYMIN(I) = YMINX(I)
341  550   CONTININUE
342 C/EJECT
343 C    log any errors that might have occurred
344 C
345        IF (EROVF)  CALL ERLG(200,LSSNUM)
346        IF (ERDYN)  CALL ERLG(201,LSSNUM)
347 C
348 C Transform the aimpoint
349 C
350        X =  FLOAT( LPAIMX(LPAIMI) )
351        Y =  FLOAT( LPAIMY(LPAIMI) )
352        Z =  FLOAT( LPAIMZ(LPAIMI) )
353        XJSA(1) = X * TRANS(1,1) + Y * TRANS(1,2) + Z * TRANS(1,3)
354        YJSA(1) = X * TRANS(2,1) + Y * TRANS(2,2) + Z * TRANS(2,3)
355        ZJSA(1) = X * TRANS(3,1) + Y * TRANS(3,2) + Z * TRANS(3,3)
356 C
357 C/EJECT
358 C Program the aimpoint into the image plane
359 C
360       DO 800 I = 1,LZONES
361          DENOM = DX(I) + XJSA(1)
362          IX = (DZ(I) + ZJSA(1)) / DENOM + SCCENX + XLATE(I)
363          LSAIMX(I) = IX
364          IY = (DY(I) + YJSA(I)) / DENOM + SCCENY
365          LSAIMY(I) = IY
366   800  CONTININUE
367 C
368 C/EJECT
```

```
C
C Calculate the minimum polygon size and area
C
      DO 1000 I = 1,LZONES
        LX = XMAXZ(I) = XMINZ(I)
        LY = YMAXZ(I) = YMINZ(I)
        IF (LX .LT. LY) THEN
          LOMSIZ(I,INX) = LX
        ELSE
          LOMSIZ(I,INX) = LY
        ENDIF
 1000 CONTINUE
C
C  Calculate the minimum background/target ratio
C
      LOMBTR(INX) = LOMBTZ(INX)
C
C  End MLTGEN
C
      RETURN
      END
```

Example A-2 Subroutine IC_TIE_POINT (Fortran 90)

```
001   ! SUBROUTINE NAME: IC_TIE_POINT
002   !
003   ! FUNCTION:
004   !   Calculates local ice tie-points and a global water tie point
005   !   for each band. Local tie points calculated by the use of a local search
006   !   window. Search window "size" is defined such that a window is
007   !   (2*size+1) pixels on a side and contains (2*size+1)*(2*size+1) pixels.
008   !
009   ! DESCRIPTION:
010   !  - Ice/water thresholds and water tie points are derived for band I1
011   !    surface reflectance, band I2 surface reflectance, and surface
012   !    temperature, using a global (granule) search window.
013   !  - Ice tie points for each pixel are derived for band I1 surface
014   !    reflectance, band I2 surface reflectance, and surface temperature,
015   !    using a local search window.
016   !  - The tie points and weights for each band are written to the Ice
017   !    Reflectance and Ice Temperature Intermediate Products.
018   !
019   ! CALLING SEQUENCE: CALL IC_TIE_POINT (Called from IC_MAIN)
020   !
021   ! INPUTS:   None
022   !
023   ! OUTPUTS:  None
024   !
025   ! REFERENCES:
026   !     Y3235 – Ice Concentration Detailed Design Document
027   !     Y2477 – Snow-Ice Module Software Architecture
028   !
029   ! DEPENDENCIES:   None
030   !
031   ! SIDE EFFECTS:   None
032   !
033   ! TARGET PROCESSOR: R10000
034   !
035   ! ------------------------------------------------------------------------
036   ! HISTORY:
037   ! D: 24 May 2002  C: initial version. Created by Mark Kowitt.
038   !
039   ! D: 14 Sept 2004 C: modified by Mark Kowitt: search_win_qual now defaults
040   !                 to 1_1 (bad) rather than 0_1 (good); pixels in search
041   !                 windows with insufficient good pixels now processed but
042   !                 flagged.
043   !
044   ! D: 15 Sept 2004 C: modified by Mark Kowitt: cleaned up use of nbin, nbig,
045   !                 nint, and ning; resized histogram arrays.
046   !
047   ! D: 20 Jan 2005  C: Modified by Mark Kowitt: fill_test global replaces
048   !                 local fill_real_test
```

```
049   !
050   ! ------------------------------------------------------------------------
051   !
052   ! GLOBAL VARIABLES:
053   !
054   !  HMAX: maximum range of histogram, by band
055   !
056   !  HMIN: minimum range of histogram, by band
057   !
058   !  IC_INDATA: Surface Reflectance (I1, I2) and Surface Temperature (I5)
059   !             data, by band and pixel, extracted from SDR and st_ip,
060   !             respectively
061   !
062   !  ICE_TIE_PT: local ice tie points by band and pixel
063   !
064   !  MIN_PIX_WIN: minimum good pixels in search window
065   !
066   !  MIN_WSIZE: minimum local search window size (pixels)
067   !
068   !  NBIN: no. of bins in histogram of modes (local window)
069   !
070   !  NBIG: no. of bins in histogram of modes (scene)
071   !
072   !  NING: no. of scene histogram bins to sum for sliding integral
073   !
074   !  NINT: no. of local histogram bins to sum for sliding integral
075   !
076   !  QBITS_I: quality bit (RDR, SDR, and IP quality), by band and pixel;
077   !           obtained from Surface Reflectance and Surface Temperature
078   !           IPs
079   !
080   !  SEARCH_WIN_QUAL: quality flag = 0 if search window contains at least
081   !    min_pix_win pixels, 1 otherwise
082   !
083   !  THRE: derived ice/water threshold, by band
084   !
085   !  THRE_MIN: min. ice/water threshold, by band
086   !
087   !  WATER_DEFAULT: default water tie points
088   !
089   !  WATER_MAX: default maximum water tie points
090   !
091   !  WATER_MIN: default minimum water tie points
092   !
093   !  WATER_TIE_PT: global water tie point, by band
094   !
095   !
096   ! LOCAL VARIABLES:
097   !
098   !  COL0: index of oldest column of search window
099   !
```

```
100   !  COL1: index of column to be added to search window
101   !
102   !  COUNT1: no. of good pixels in active window
103   !
104   !  HISTOGRAM: active histogram bin counts
105   !
106   !  ICE_BIN: ice histogram bin width
107   !
108   !  WATER_BIN: water histogram bin width
109   !
110   !  WIDTH: size of active search window
111   !
112   !*************************************************************************
113
114   SUBROUTINE IC_tie_point
115         USE IC_util
116         IMPLICIT NONE
117
118         INTEGER(KIND=4)::cmw, cpw, col0, col1, count1, decrement, dumpcount
119         INTEGER(KIND=4)::kmax, mws1, n1, n2, n3, n4, n5
120         INTEGER(KIND=4)::nbig1, nbin1, rmw, rpw,
121         INTEGER(KIND=4)::width, winmax, ymax, yn(nbig+1)
122
123         REAL(KIND=4) :: ice_bin, water_bin, pk, local_max, local_min
124
125   !-------------------------------------------------------------------------_
126   !                              EXECUTABLE CODE
127   !-------------------------------------------------------------------------
128
129          io = no_error
130
131          ! Change Sept 14, 2004 – search_win_qual default = BAD
132
133          search_win_qual = 1_1   ! Initialize search window quality
134
135          mws1 = min_wsize + 1
136          ice_tie_pt = fill_real
137          nbin1 = nbin+1-nint
138
139          ! Do for each band
140
141   loop3: DO band = 1,3
142
143             dumpcount=0
144
145             ! Bin size for water histogram (below the water/ice threshold)
146
147             water_bin = (thre(band) - hmin(band)) / REAL(nbig)
148
149             ! Do for each row
```

```
150
151   loop2:    DO row = mws1, (nrows_i - min_wsize)
152
153               count1 = 0  ! Initialize good pixel counter
154
155   loop1:       DO col = mws1, (ncols_i - min_wsize)
156
157
158                     ! Skip histogram if pixel fill value or deweighted
159
160                     IF(ic_indata(col,row,band) < fill_test .OR. &
161                                    w(col,row,band) <= 0.0) THEN
162                           CYCLE loop1
163                     END IF
164
165                     width = min_wsize ! Initialize search window
166
167                     cmw = col - width
168                     cpw = col + width
169                     rmw = row - width
170                     rpw = row + width
171
172                     ! A "good" pixel is one where qbits_i=0.
173
174                     count1 = COUNT(qbits_i(cmw:cpw,rmw:rpw,band)==0_1)
175
176                     ! If insufficient number of pixels, cannot make histogram
177
178                     IF( count1 < 2 ) THEN
179                           CYCLE loop1 ! Skip to next pixel
180                     END IF
181
182                     ! Enough pixels in the window -> good search window
183                     IF( count1 >= min_pix_win ) THEN
184                           search_win_qual(col,row,band) = 0_1
185                     END IF
186
187                     local_max = MAXVAL(ic_indata(cmw:cpw,rmw:rpw,band), &
188                                        qbits_i(cmw:cpw,rmw:rpw,band)==0_1)
189
190                     local_min = MINVAL(ic_indata(cmw:cpw,rmw:rpw,band), &
191                                        qbits_i(cmw:cpw,rmw:rpw,band)==0_1)
192
193                     IF( local_max > local_min) THEN
194                           ice_bin = (local_max - local_min) / REAL(nbin)
195                     ELSE
196                           CYCLE loop1 ! skip to next pixel
197                     END IF
198
199                     ! Populate the whole search window
```

```
200
201                     histogram = 0     ! Good pixel counter
202
203                     DO j = cmw,cpw
204                           DO k = rmw,rpw
205                                 IF( qbits_i(j,k,band)==0_1 ) THEN
206                                       i = 1+INT((ic_indata(j,k,band) - &
207                                                   local_min)/ice_bin)
208                                       histogram(i) = histogram(i) + 1
209                                 END IF
210                           END DO
211                     END DO
212
213                     ! Find the bin with the highest frequency value
214
215                     yn = 0
216                     DO j = 1,nint
217                           yn(1) = yn(1) + histogram(j)
218                     END DO
219
220                     ymax = yn(1)
221                     kmax = 1
222                     DO k = 2,nbin1
223                           yn(k) = yn(k-1) - histogram(k-1) + &
224                                       histogram(k+nint-1)
225                           IF(yn(k) > ymax) THEN
226                                 ymax = yn(k)
227                                 kmax = k
228                           END IF
229                     END DO
230
231                     ! Correction for extended maximum
232
233                     pk = REAL(kmax)
234                     IF(kmax==1) THEN
235                           ice_tie_pt(col,row,band) = fill_real
236                     ELSE IF(kmax==nbin1) THEN
237                           pk = pk - 1.0 + 0.5 * REAL(nint)
238                           ice_tie_pt(col,row,band) = thre(band)+ ice_bin * pk
239                     ELSE IF(kmax<nbin1) THEN
240
241                           ! Correction for extended maximum
242
243                           DO k=kmax+1, nbin1
244                                 IF(yn(k)==yn(kmax)) THEN
245                                       pk = pk + 0.5
246                                 ELSE IF(yn(k)<yn(kmax)) THEN
247                                       EXIT
248                                 END IF
249                           END DO
```

```
250                         pk = pk - 1.0 + 0.5 * REAL(nint)
251                         ice_tie_pt(col,row,band) = local_min + ice_bin * pk
252
253                   END IF
254
255                   IF(dumpcount==0) THEN
256
257                         ! First good pixel for ice tie point calculation
258
259                         dumpcount=1
260
261                   END IF
262
263               END DO loop1  ! cols
264
265             END DO loop2    ! rows
266
267             ! Bin size for water histogram (below the water/ice threshold)
268
269             water_bin = (thre(band) - hmin(band)) / REAL(nbig)
270
271             ! Build a water parameter histogram for the whole scene
272
273             histogram = 0
274             nbig1 = nbig+1-ning
275
276             DO j=1,ncols_i
277
278                DO k=1,nrows_i
279
280                     IF( IAND(qbits_i(j,k,band),7_1)==4_1) THEN
281
282                           ! Only include water pixels with "GREEN" quality
283
284                           i = 1 + INT((ic_indata(j,k,band)-hmin(band)) / &
285                                                              water_bin)
286                           histogram(i) = histogram(i) + 1
287
288                     END IF
289
290                DO     ! rows
291
292             END DO  ! cols
293
294             ! Find the peak and its index
295
296             yn = 0
297             DO j=1,ning
298                yn(1) = yn(1) + histogram(j)
299             END DO
```

```
300
301             ymax = yn(1)
302             kmax = 1
303             DO k = 2,nbig1
304                yn(k) = yn(k-1) - histogram(k-1) + histogram(k+ning-1)
305                IF(yn(k) > ymax) THEN
306                     ymax = yn(k)
307                     kmax = k
308                END IF
309             END DO
310
311             ! Correction for extended maximum
312
313             pk = REAL(kmax)
314             IF(kmax<nbig1) THEN
315                DO k = kmax+1, nbig1
316                     IF(yn(k)==ymax) THEN
317                           IF(k<nbig1) THEN
318                                 pk = pk + 0.5
319                           ELSE
320                                 water_tie_pt(band) = fill_real
321                           END IF
322                     ELSE IF(yn(k)<ymax) THEN
323                           pk = pk - 1.0 + ( 0.5 * REAL(ning) )
324                           water_tie_pt(band) = hmin(band) + water_bin * pk
325                           EXIT
326                     END IF
327                END DO
328             ELSE
329                water_tie_pt(band) = fill_real
330             END IF
331
332             ! Check water tie points against default values from LUT
333             IF (water_tie_pt(band) < WATER_MIN(band) .OR. &
334                 water_tie_pt(band) > WATER_MAX(band)) THEN
335
336                ! Default water tie point
337
338                water_tie_pt(band) = WATER_DEFAULT(band)
339
340             END IF
341
342          END DO loop3  ! band=1,3
343
344          io = no_error
345
346    !
347    ! End subroutine IC_TIE_POINT
348    !
349    END SUBROUTINE IC_TIE_POINT
```

———

APPENDIX B. FORTRAN CODING STANDARDS - QUICK REFERENCE

The following quick reference was provided by Walter Wolf (STAR).

Program unit elements and order:

1. program unit identifier
2. preamble
3. INCLUDE files
4. specification statements
5. DATA statements
6. statement function statements
7. executable statements
8. EXIT statement
9. END statement.

Preamble:

1. NAME
2. FUNCTION
3. DESCRIPTION
4. REFERENCE
5. CALLING SEQUENCE
6. INPUTS
7. OUTPUTS
8. DEPENDENCIES
9. RESTRICTIONS
10. HISTORY

IMPLICIT NONE must be used in all program units.

Statement Order: Separate functions so that all Input functions precede all Processing functions, followed by all Output functions.

Each statement shall begin on a separate line.

All variables shall be declared using a type-statement, INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, and CHARACTER.

Variables shall be explicitly set or initialized before use, including complex data types such as arrays and structures. The KIND statement may be used to specify the type. Variables of derived types (Fortran 90) may be declared as structures.

Constants should be given an uppercase name. If it is only used in one file, it should be declared at the head of that file; if used in multiple files, it should be declared in an include file.

Names can not be identical to Fortran reserved words or implementation supplied function names.

Alphabetic case shall be used consistently to enhance readability throughout a program.

Common Block Variables shall be named the same in every program unit that uses the common data. Common blocks shall be declared in a separate file and copied into the source file.

Indentation shall be used consistently to enhance readability throughout a program.

- A comment line should be indented in the same way as the following executable line of code.
- All statements in the same block should be indented by the same amount.
- Statements in inner blocks should be indented by a greater amount than statements in outer blocks.
- No tabs.

Line Continuation

- F77: Begin each continuation line with an ampersand (&) in column six (6), and indent two (2) levels of indentation relative to the first statement line.
- F90: Insert one blank character and an ampersand (" &") at the end of the line and begin the continuation line at an indentation so that the last character of the continuation line lines up with the last character of the preceding line.

Blocking with blank lines shall be used consistently to enhance readability throughout a program.

Compound Expressions shall have spaces before and after relational operators, Fortran reserved words, identifiers, and arithmetic operators to enhance readability of compound expressions.

Parentheses and spaces shall be used to help clarify evaluation of logical and arithmetic expressions.

DO Statements shall not contain any statements that change the value of the loop-controlling variable.

Return all called subroutines to the next executable statement of the calling routine, except for error handling provisions. The use of an alternate return specifier as an argument in a calling sequence in the event of an error (e.g., "CALL foo (a, b, *999)") is highly discouraged.

Format Statements

- F77: Accompany each READ and WRITE statement with the corresponding FORMAT statement.
- F90: Specify format with the FMT specifier in the argument list of a READ or WRITE statement.

F77 Only

- Avoid referencing statement numbers in comments (Fortran 77). Consistently justify statement numbers in columns two (2) through five (5) in ascending order throughout a program.
- All loops shall terminate with a unique CONTINUE statement (Fortran 77)
- Each GO TO statement shall target a unique CONTINUE statement (Fortran 77).

Standard Mathematical and Geophysical Constants shall be used (e.g. PI).

Dynamic Allocation of Memory shall be used wherever possible. Avoid COMMON blocks. Do not allocate memory for local variables until they are used in a subprogram, and deallocate the memory as soon as its use is finished.

Guidelines:

- Program units containing more than 200 lines of code should be examined to see if they can be segmented.
- Nesting of statements shall be limited to five (5) levels.
- Computed GO TO (Fortran 77) or ELSE IF statements shall be acceptable in lieu of a CASE statement. However, out of range conditions shall be included in the statement.
- Indent the statements following the Fortran DO statement one (1) level of indentation.
- Enclose the condition(s) following the Fortran reserved word IF in parentheses.
- Use the IMSL Fortran Numerical Library. This library is usually available from suppliers of Fortran compilers.
- Recursive routines should be avoided on efficiency grounds.

Declarations:

- Fortran 77: Align each declaration type name in column seven (7).
- Avoid continuation lines in a declaration statement by using multiple statements.
- List several variables of a single type on a line alphabetically.
- Use named common blocks for all global externally referenced common data.

Naming:

- Avoid names that look alike by differing only in characters that resemble each other, as do 2 and z, 0 and O, 5 and S, or I and 1.
- Name programs, subroutines, and functions to indicate purpose.
- Familiarize yourself with the STAR Common Library of Fortran routines. This serves two main purposes: 1) You may find a library routine that you can use to implement your desired function, 2) You should avoid using names that are similar to library routines.
- Name symbolic variables to indicate what they are, not what values they may contain.
- Names should be as mnemonically descriptive as possible, subject to constraints imposed by Fortran standards.

Error Trapping:

- Check for error return values, even from functions that "can't" fail.
- Include the system error text for every system error message.
- In I/O statements include an item of the form ERR=label.
- In READ statements include an item of the form END=label.

Comments:

- No more than 10 – 15 executable lines of code without a comment.
- The comment should be descriptive enough to explain the function of the following block of code.

- F77: Delimit comments with a 'C' in column one (1) and a blank line before and after the comment line.
- F90: Delimit comments with a ‘!’ character and a blank line before and after the comment line.
- Precede each major section of code with a block comment briefly describing the processing involved.
- Precede each conditional statement (e.g., DO, IF) with a block comment describing the condition being tested and the branching alternatives.
- Accompany each variable declaration with a comment.
- Accompany each program unit END statement with a comment indicating the program unit name, unless the name is already included in the END statement.
- A comment should precede all lines of code or blocks of code that represent a significant revision of previously base-lined code. It is helpful to include a reference to the reason for the code change.

APPENDIX C. TRANSITION FROM FORTRAN 77 TO FORTRAN 90

Information to assist Fortran 77 (fixed form) programmers in the writing of free form Fortran code (90 and later) is provided in an Appendix to this TD, “STAR_TD-11.1.A._v3r0.doc”. This file will be available to authorized users in the STAR EPL PAR.

END OF DOCUMENT