SCIENCE COMPUTATION FORMULATION (SCF)
FILE FORMAT DEFINITION
Version 1.0

Department of Space Science
Southwest Research Institute ® (SwRI ®)
6220 Culebra Road
San Antonio, TX 78238-5166

Prepared By:

Carrie Gonzalez
cgonzalez@swri.org

The development of this Software was supported by contracts NAG5-3148, NAG5-6855, NAS8-36840, NAG5-2323, and NAG5-7043 issued on behalf of the United States Government by its National Aeronautics and Space Administration.

AGREE TO HOLD SwRI HARMLESS FROM, AND YOU COVENANT NOT TO SUE SwRI FOR, ANY CLAIMS BASED ON USING THE SOFTWARE.

Local Laws: Export Control

You acknowledge and agree this Software is subject to the U.S. Export Administration Laws and Regulations. Diversion of such Software contrary to U.S. law is prohibited. You agree that none of the Software, nor any direct product therefrom, is being or will be acquired for, shipped, transferred, or reexported, directly or indirectly, to proscribed or embargoed countries or their nationals, nor be used for nuclear activities, chemical biological weapons, or missile projects unless authorized by U.S. Government. Proscribed countries are set forth in the U.S. Export Administration Regulations. Countries subject to U.S embargo are: Cuba, Iran, Iraq, Libya, North Korea, Syria, and the Sudan. This list is subject to change without further notice from SwRI, and you must comply with the list as it exists in fact. You certify that you are not on the U.S. Department of Commerce's Denied Persons List or affiliated lists or on the U.S. Department of Treasury's Specially Designated Nationals List. You agree to comply strictly with all U.S. export laws and assume sole responsibilities for obtaining licenses to export or reexport as may be required.

General

These Terms represent the entire understanding relating to the use of the Software and prevail over any prior or contemporaneous, conflicting or additional, communications. SwRI can revise these Terms at any time without notice by updating this posting.

Trademarks

The SwRI logo is a trademark of SwRI in the United States and other countries.

Revision Log

Revision	Release Date	Changes to Document
Version 1.0	07/10/06	• Original Release in Word Format • Updated supported tensor functions • data type for **s_input_opers** was changed from short to long • spacecraft potential IDFS data type added to input variable section

TABLE OF CONTENTS

Science Computation Formulation (SCF) Overview

The SCF file definition is separated into five basic sections. The first section consists of general information; the second section contains the description of the input variables to be used in the computations; the third section contains the description of any temporary variables that may be needed for the computations; the fourth section contains the description of the output variables generated by the computations; and the fifth section contains the description of the algorithm steps. Each line in the SCF file is defined to be a maximum of 80 characters and must be terminated by the line feed (newline) character. Throughout the SCF file, comment lines may be inserted at any location. Comment lines are designated to start with "/*" and conclude with either the end of a line or a "*/", whichever comes first. Thus, if multiple line comments are to be included, a "/*" must appear on each line before any other text. A comment will be assumed to end the line; no input on the line following a comment will be considered.

An overview of the SCF fields is shown in the tables below. The SCF file consists of a set of fields. These fields may be filled with an individual entry or multiple (block) entries. A block entry is a set of individual entries that are taken as a group to define a field within the SCF. An example of a block entry is the temporary variable field, which is defined by specifying three individual entries **s_temp_name**, **s_temp_dimension** and **s_temp_lengths**. Each individual field is characterized by a data type, an array size, and a repetition value. The block entry definitions for the input variables, temporary variables and output variables are defined in secondary tables below the main SCF file format table. Note that while a block entry does not have either a data type or array size associated with them, they do have a repetition value. All of the data types used for the SCF fields are signed quantities unless otherwise stated. The following table summarizes the byte sizes for each of the data types specified:

DATA TYPE	BYTE SIZE
char	1
short	2
long	4
float	4

SCF FILE FORMAT				
FIELD ID	**FIELD DESCRIPTION**	**DATA TYPE**	**ARRAY SIZE**	**FIELD REPETITION**
s_title	SCF title	char	79	1
s_num_contact	Number of lines for the contact	long		1
s_contact	Contact address	char	79	s_num_contact
s_num_comments	Number of comment lines	long		1
s_comments	General comments	char	79	s_num_comments
s_num_input	Number of input variables	long		1
INPUT VARIABLE	Put Input Variable Info. Here			s_num_input
s_num_temp	Number of temporary variables	long		1
TEMP VARIABLE	Put Temporary Variable Info. Here			s_num_temp
s_num_output	Number of output variables	long		1
OUTPUT VARIABLE	Put Output Variable Info. Here			s_num_output
s_equation	Algorithm step	char	79	>= 1

The INPUT VARIABLE field id represents a set of individual entries that define the input variables within the SCF file. There are **s_num_input** sets defined. The structure of the INPUT VARIABLE block entry is shown in the table below.

SCF INPUT VARIABLE FORMAT				
FIELD ID	**FIELD DESCRIPTION**	**DATA TYPE**	**ARRAY SIZE**	**FIELD REPETITION**
s_input_name	Variable Name	char	19	1
s_input_proj	Project Name	char	14	1
s_input_mission	Mission Name	char	14	1
s_input_exp	Experiment Name	char	14	1
s_input_inst	Instrument Name	char	14	1
s_input_vinst	Virtual Instrument Name	char	14	1
s_input_dtype	Data type	char	14	1
s_input_dnum	Item Number	short		1
s_input_cset	Calibration Set Number	short		1
s_input_num_tbls	Number of tables	short		1
s_input_tbls	Table Number	short		s_input_num_tbls
s_input_opers	Table Operator	long		s_input_num_tbls
s_input_lower_cut	Lower Cutoff Value	float		1
s_input_upper_cut	Upper Cutoff Value	float		1
s_input_qual_min	Data Quality Minimum Exclusion Value	unsigned short		1
s_input_qual_max	Data Quality Maximum Exclusion Value	unsigned short		1

The TEMPORARY VARIABLE field id represents a set of individual entries that define the temporary variables within the SCF file. There are **s_num_temp** sets defined. The structure of the TEMPORARY VARIABLE block entry is shown in the table below.

SCF TEMPORARY VARIABLE FORMAT				
FIELD ID	**FIELD DESCRIPTION**	**DATA TYPE**	**ARRAY SIZE**	**FIELD REPETITION**
s_temp_name	Variable Name	char	19	1
s_temp_dimension	Dimension of the Variable	long		1
s_temp_lengths	Length of each dimension	long		s_temp_dimension

The OUTPUT VARIABLE field id represents a set of individual entries that define the output variables within the SCF file. There are **s_num_output** sets defined. The structure of the OUTPUT VARIABLE block entry is shown in the table below.

SCF OUTPUT VARIABLE FORMAT				
FIELD ID	**FIELD DESCRIPTION**	**DATA TYPE**	**ARRAY SIZE**	**FIELD REPETITION**
s_output_name	Variable Name	char	19	1
s_output_dimension	Dimension of the Variable	long		1
s_output_lengths	Length of each dimension	long		s_output_dimension

The remaining portion of this document will describe in detail the individual fields within the SCF file. Each field must be entered as a single line of input in the SCF file unless otherwise stated. The SCF software utilizes some reserved keywords and reserved characters that cannot be used when defining input, temporary and output variable names. The reserved characters include the left parenthesis "(", right parenthesis ")", left bracket "[", right bracket "]", less than "<", greater than ">", equal "=", exclamation point "!" and comma "," characters. The list of reserved keywords is given in the table below:

RESERVED KEYWORDS
SYEAR
SDAY
SMILLI
SNANO
EYEAR
EDAY
EMILLI
ENANO
NUM_SAMPLES
VECTOR_LEN
OUTSIDE_MIN
OUTSIDE_MAX
VALID_MIN
VALID_MAX
FIRST_TIME
FOR
TO
BEGIN
END
BREAK
IF
ELSE
ENDIF

The keyword **NUM_SAMPLES** is an internal SCF variable that is used to specify the actual number of samples returned for the input variable that controls the time interval being processed. The keyword **VECTOR_LEN** specifies the maximum number of samples that can be returned for the input variable that controls the time interval being processed. In some cases, the two values will be identical; in other cases, the **NUM_SAMPLES** value will be less than the **VECTOR_LEN** value. The keywords **OUTSIDE_MIN**, **OUTSIDE_MAX**, **VALID_MIN** and **VALID_MAX** represent the predefined values that the SCF software utilizes for data values. These internal SCF variables can be used within the algorithm section as comparison values for the IF-ELSE-ENDIF or as initialization values. The keywords **SYEAR**, **SDAY**, **SMILLI**, **SNANO**, **EYEAR**, **EDAY**, **EMILLI** and **ENANO** are internal SCF variables that are used to specify the time duration covered at each iteration of the algorithm. These internal variables can

be accessed by the user within the algorithm definition. The keyword **FIRST_TIME** may be utilized within the algorithm definition section, serving as a conditional flag which indicates if the algorithm is being executed for the first time. A value of 1 means that the algorithm is being executed for the first time; a value of 0 indicates successive iterations of the algorithm.

Where appropriate, fields which are linked together by a common basis will be grouped together under a single heading. The SCF has a fixed format and the fields MUST be in the order outlined in the tables above. Sample SCF entries are included for some of the fields and these are shown exactly as they would appear in the SCF file.

SCF GENERAL INFORMATION

The first five field id's that are defined within the SCF file pertain to general information concerning the SCF.

1. s_title

This field is a single line of text, less than 80 characters in length, which can be used as annotation for a plot. If this field is not utilized, a blank line **must** be inserted in the SCF file.

2. s_num_contact

This field is the number of contact lines entered in the SCF. This value must be greater than or equal to zero. If this value is zero, no other contact information should be entered in the SCF.

3. s_contact

This field is a set of lines, each line less than 80 characters in length, which contains the name and address of someone who can act as a focus for any questions which might arise concerning the design of the SCF. The number of lines is defined in the **s_num_contact** entry.

4. s_num_comments

This field is the number of comment lines entered in the SCF. This value must be greater than or equal to zero. If this value is zero, no other comments information should be entered in the SCF.

5. s_comments

This field is a set of free form text, each line less than 80 characters in length. The number of lines of text is defined in the **s_num_comments** entry. Comment lines may be used for any general documentation.

INPUT VARIABLES

Each SCF file contains information describing the input variables that are utilized in the computations. Input variables are only derived from IDFS data products. Input variables are either scalar quantities, IDFS vector quantities or multi-dimensional (tensor) quantities, based upon the derivation of the IDFS data source. A scalar quantity is a single data value that is dependent only upon time and position. An IDFS vector quantity is a one-dimensional data item that has a functional dependence on a single variable, which in IDFS terminology is called the scanning variable. The length of the vector is dictated by the IDFS data source. A multi-dimensional quantity is an N-dimensional data item, with N limited to a maximum value of 10. The original IDFS paradigm provided for only scalar and IDFS vector quantities. The IDFS paradigm has since been expanded to include the multi-dimensional, or tensor, quantities.

The fields that are pertinent to the definition of input variables in the SCF are described below. It is important to note that not all input fields are relevant for a given data source. The interactions of the fields will be described where appropriate.

6. s_num_input

This field is the number of input variables defined in the SCF. This number must be greater than zero.

7. s_input_name

This field is the name of the input variable, less than 20 characters in length, with no blank spaces between any two characters. The first character of the input variable name must be a letter. The variable name cannot be identical to any of the reserved keywords nor can it contain any of the reserved characters used within the SCF.

Input Variable Data Source

All input variables must be defined as coming from an IDFS data source. This is accomplished by providing the five lineage fields of the virtual instrument, all on a single line of input in the SCF, with each field separated by one or more blanks. The five lineage fields are described below.

8. s_input_proj

This field is the acronym, less than 15 characters in length, which identifies the particular project which the virtual instrument is associated with.

9. s_input_mission

This field is the acronym, less than 15 characters in length, which identifies the particular mission within a project which the virtual instrument is associated with.

10. s_input_exp

This field is the acronym, less than 15 characters in length, which identifies the particular experiment within a mission which the virtual instrument is associated with.

11. s_input_inst

This field is the acronym, less than 15 characters in length, which identifies the particular instrument within an experiment which the virtual instrument is associated with.

12. s_input_vinst

This field is the acronym, less than 15 characters in length, which identifies the particular virtual instrument to be used as the data source for the input variable.

Data Type

There are several data sources that may be obtained from an IDFS data set. These data sources include the following: (1) sensor data, (2) scan data, (3) calibration data, (4) instrument status (mode) values, (5) data quality flags, (6) pitch angle, (7) start azimuthal angle, (8) stop azimuthal angle, (9) spacecraft potential, (10) spin rate, (11) data accumulation time in milliseconds, (12) data accumulation time residual in nanoseconds, (13) data latency value in milliseconds and (14) data latency time residual in nanoseconds. An input variable must be identified as coming from one of these fourteen sources, all of which reside in the IDFS paradigm. Some of these data types, as noted in the table below, are not currently defined for multi-dimensional IDFS data sets. The fields that are used to describe the data type for the input variable in the SCF must all appear on a single line, in the proper order. A description of these fields and associated parameters are found below.

13. s_input_dtype

This field is a string, less than 15 characters in length, which identifies the input variable source. There are fourteen possible sources and each is described in the table below:

S_INPUT_DTYPE	MEANING	MULTI-DIMENSIONAL IDFS DATA SET
SENSOR	sensor data	Defined
SCAN	scan data	Not Defined
CAL_DATA	calibration data	Not Defined
MODE	instrument status value	Defined
D_QUAL	data quality flag	Defined
PITCH_ANGLE	pitch angles, one value per sample	Not Defined
START_AZ	start azimuthal angles, one value per sample	Not Defined
STOP_AZ	stop azimuthal angles, one value per sample	Not Defined
SC_POTENTIAL	spacecraft potential data, one value per sample	Not Defined
SPIN_RATE	instrument spin rate	Defined

S_INPUT_DTYPE	MEANING	MULTI-DIMENSIONAL IDFS DATA SET
DATA_ACCUM_MS	amount of time for a single data acquisition, in milliseconds	Defined
DATA_ACCUM_NS	remainder of DATA_ACCUM_MS, in nanoseconds	Defined
DATA_LAT_MS	amount of dead time between successive data acquisitions, in milliseconds	Defined
DATA_LAT_NS	remainder of DATA_LAT_MS, in nanoseconds	Defined

The data that is selected is either a scalar value, an IDFS vector (1-D) value, or a multi-dimensional value, depending upon the data type and IDFS data set selected. The SPIN_RATE, DATA_ACCUM_MS, DATA_ACCUM_NS, DATA_LAT_MS and DATA_LAT_NS data sources are scalar quantities that do not utilize the **s_input_dnum** field since the data is instrument specific, not sensor specific

14. s_input_dnum

The interpretation of this field depends upon the data type selected for the input variable. For all data types, the value must be greater than or equal to zero, with numbering starting at zero. For the **MODE** data type, this field is the instrument status value of interest. For all other data types, this field is the sensor number. The table below gives an explanation of the data that is returned when the data type specification stops with this parameter.

DATA TYPE FIELD INTERACTION		
DATA TYPE	S_INPUT_DNUM	MEANING
SENSOR	X	Sensor data for sensor X is returned. (scalar, 1-D vector or tensor value based upon the IDFS data set being used)
SCAN	X	Scan data for sensor X is returned. (1-D vector)
CAL_DATA	X	Calibration data for sensor X is returned. (scalar or 1-D vector value based upon the IDFS data set being used)
MODE	X	Instrument status value for mode X is returned. (scalar value)
D_QUAL	X	Data quality flag for sensor X is returned. (scalar value)
PITCH_ANGLE	X	Pitch angle data for sensor X is returned. (scalar or 1-D vector value based upon the IDFS data set being used)
START_AZ	X	Start azimuthal angles for sensor X is returned. (scalar or 1-D vector value based upon the IDFS data set being used)
STOP_AZ	X	Stop azimuthal angles for sensor X is returned. (scalar or 1-D vector value based upon the IDFS data set being used)
SC_POTENTIAL	X	Spacecraft potential data for sensor X is returned (scalar or 1-D vector value based upon the IDFS data set being used)

15. s_input_cset

This field is applicable to the **CAL_DATA** data type only. This field is utilized to specify the calibration set number. The value must be greater than or equal to zero, with numbering starting at zero. The following are four examples of possible data type definitions that may appear in SCF files:

```
SPIN_RATE          /* spin rate for the instrument */
D_QUAL 3           /* data quality for sensor three */
```

```
CAL_DATA 0 1                /* data from calibration set 1 for sensor 0 */
SENSOR 1                    /* sensor data for sensor one */
```

Data Unit

The definition of the input variable must specify the physical unit that is to be utilized for the calculations. The fields that are used to describe the physical unit for the input variable in the SCF are described below.

16. s_input_num_tbls

This field is the number of tables that are to be applied to convert the data from raw IDFS format into a set of physical units. This value must be greater than or equal to zero and less than 128. If this value is zero, no other table information should be entered in the SCF file. For multi-dimensional IDFS data sets, there is currently no mechanism available for unit conversion of the raw IDFS data; therefore, this field must be set to zero or an error condition will be reported when the SCF file is processed.

17. s_input_tbls

This field holds the table number(s) that are to be applied to the raw IDFS data to convert the data into the physical unit of interest. These table numbers pertain to the tables defined in the VIDF file, with table numbers starting at zero. There must be **s_input_num_tbls** table numbers specified, all on a single line of input in the SCF file with each value separated by one or more blanks. If the table numbers do not fit on a single line, the line may be continued by use of the line continuation character '\' placed at the end of the line before any in-line comment. The tables are applied in the order in which they appear in the list. In almost all cases, table numbers should be greater than or equal to zero and less than 128. The only exception would be the usage of -1 as a table number, which can be used in conjunction with table operations in the range of 2001 – 2005 and table operations greater than 10000. These table operations utilize data buffers as opposed to VIDF tables; therefore, the table number is ignored.

18. s_input_opers

This field holds the operations that are to be applied to the specified tables in order to convert the data into the physical unit of interest. There must be **s_input_num_tbls** operations specified, all on a single line of input in the SCF file with each value separated by one or more blanks. If the table operations do not fit on a single line, the line may be continued by use of the line continuation character '\' placed at the end of the line before any in-line comment.

Data Cutoff Values

The input data can be filtered such that only data that falls within a defined range will be included in the computation. The definition of this range is achieved through the two fields described below. Both fields must be specified on the same line of input in the SCF file.

19. s_input_lower_cut

This field holds the smallest data value that is to be included when collecting the input data. This value should be entered in terms of the data unit being returned for this input variable. The **s_input_lower_cut** value must be less than the **s_input_upper_cut** value.

20. s_input_upper_cut

This field holds the largest data value that is to be included when collecting the input data. This value should be entered in terms of the data unit being returned for this input variable. The **s_input_upper_cut** value must be greater than the **s_input_lower_cut** value.

For either of these two fields, the predefined values VALID_MIN and VALID_MAX may be used. These two values represent the smallest and the largest data value that is recognized by the IDFS data access routines, respectively. Upon execution of the algorithm, if all of the data for the acquisition period falls outside of the defined data cutoff range, the value for that input variable will be set to the predefined value OUTSIDE_MIN.

Data Quality Exclusion

The input data can be filtered to exclude data based upon the value of the data quality flag. Data quality exclusion is defined by the two fields described below. Both fields must be specified on the same line of input in the SCF file.

21. s_input_qual_min

This field holds the minimum data quality exclusion flag value. This value must be greater than or equal to zero and less than or equal to 256. The **s_input_qual_min** value must be less than or equal to the **s_input_qual_max** value.

22. s_input_qual_max

This field holds the maximum data quality exclusion flag value. This value must be greater than or equal to zero and less than or equal to 256. The **s_input_qual_max** value must be greater than or equal to the **s_input_qual_min** value.

If a single data quality flag is to be excluded, both fields should be set to the same value. If all data is to be included, that is, no data quality flags are to be excluded, both fields should be set to 256; otherwise, any data that has a data quality flag contained between the **s_input_qual_min** and **s_input_qual_max** values will not be utilized in the calculation. Upon execution of the algorithm, if all of the data for the acquisition period is excluded based upon the data quality flags, the value for that input variable will be set to the predefined value OUTSIDE_MIN.

TEMPORARY VARIABLES

An SCF file may contain information describing temporary variables that are utilized in the computations. Temporary variables are symbolic names given to hold constant values (such as π), format conversion strings, or intermediate results during the evaluation of the algorithm. Temporary variables are not returned by the SCF data access routines; only the defined output variables are returned. The values for the temporary variables are not cleared after each iteration of the SCF algorithm; therefore, temporary variables can be treated as "static" variables. The fields that are pertinent to the definition of temporary variables in the SCF are described below.

23. s_num_temp

This field is the number of temporary variables defined in the SCF. This number must be greater than or equal to zero. If this value is zero, no other information pertinent to the definition of temporary variables should be entered in the SCF.

24. s_temp_name

This field is the name of the temporary variable, less than 20 characters in length, with no blank spaces between any two characters. The first character of the temporary variable name must be a letter. The variable name cannot be identical to any of the reserved keywords nor can it contain any of the reserved characters used within the SCF.

Dimension Of Temporary Variable

All temporary variables must be given a dimension. The size or length of each dimension must be specified on the same input line of the SCF file. If the dimension and lengths do not fit on a single line, the line may be continued by use of the line continuation character '\' placed at the end of the line before any in-line comment. The two fields described below define the dimension of the temporary variable.

25. s_temp_dimension

This field is the dimensionality of the temporary variable, e.g. 1-D, 2-D, etc. The SCF software supports a storage class up to ten dimensions. To specify a scalar value, a value of zero should be specified for this field. Data products of higher dimensions can be created in the SCF equation section using data from defined input variables and/or other temporary variables.

26. s_temp_lengths

This field holds the length or size of each dimension defined for the temporary variable. If the temporary variable is defined as a scalar value (**s_temp_dimension** equals zero), this field is not applicable and must be left blank. Otherwise, there must be a length defined for each dimension specified, with each length value being greater than zero. The keyword **SWP_LEN** may be used as a value for the **s_temp_lengths** field. If this is done, the SCF software will set

the size of the dimension equal to the size of the input variable with the largest sample size (max_samp_size).

The following are some examples of possible dimension definitions that may appear in SCF files:

```
0                /* scalar value */
1 SWP_LEN        /* 1-D variable of size max_samp_size */
2 3 3            /* 3 x 3 matrix */
```

OUTPUT VARIABLES

Each SCF file contains information describing the output variables that are generated by the computations. Output variables are symbolic names given to the derived data products that are generated by the computations. There must be at least one defined output variable; otherwise, the point of the computation is meaningless. Unlike input variables where there is a selection of physical units, each output variable returns data in only one physical unit that is determined by the algorithm used to create it in conjunction with the units of the input variables used in the computation. The fields that are pertinent to the definition of output variables in the SCF are described below.

27. s_num_output

This field is the number of output variables defined in the SCF. This number must be greater than zero.

28. s_output_name

This field is the name of the output variable, less than 20 characters in length, with no blank spaces between any two characters. The first character of the output variable name must be a letter. The variable name cannot be identical to any of the reserved keywords nor can it contain any of the reserved characters used within the SCF.

Dimension Of Output Variable

All output variables must be given a dimension. The size or length of each dimension must be specified on the same input line of the SCF file. If the dimension and lengths do not fit on a single line, the line may be continued by use of the line continuation character '\' placed at the end of the line before any in-line comment. The two fields described below define the dimension of the output variable.

29. s_output_dimension

This field is the dimensionality of the output variable, e.g. 1-D, 2-D, etc. The SCF software supports a storage class up to ten dimensions. To specify a scalar value, a value of zero should be specified for this field. Data products of higher dimensions can be created in the SCF equation section using data from defined input variables and/or other defined variables.

30. s_output_lengths

This field holds the length or size of each dimension defined for the output variable. If the output variable is defined as a scalar value (**s_output_dimension** equals zero), this field is not applicable and must be left blank. Otherwise, there must be a length defined for each dimension specified, with each length value being greater than zero. The keyword **SWP_LEN** may be used as a value for the **s_output_lengths** field. If this is done, the SCF software will set the size of the dimension equal to the size of the input variable with the largest sample size (max_samp_size).

The following are some examples of possible dimension definitions that may appear in SCF files:

```
0                /* scalar value */
1 SWP_LEN        /* 1-D variable of size max_samp_size */
2 3 3            /* 3 x 3 matrix */
```

ALGORITHM

The SCF is a user defined algorithm, specifying a set of mathematical operations to be performed on the data from one or more instruments to produce a set of modified data values. There is only **one** defined operation per algorithm step.

31. s_equation

This field is a set of lines, each line less than 80 characters in length, where each line defines a single algorithm step. At least one line must be defined in the algorithm section of the SCF. If blank lines are desired for readability, a comment line must be used. Comment lines are designated to start with "/*" and conclude with either the end of a line or a "*/", whichever comes first. **There must be at least one blank character between each term specified in the algorithm step**. There are five formats which an algorithm can take. In each of these formats, a non-scalar variable can be indexed to retrieve a scalar quantity, e.g. FMT[0] references the first element in the 1-D variable FMT. The index value can be a number or a variable, as in the case, FMT[T1]. If the index value is a variable, the index variable must be a scalar quantity and the index variable cannot be indexed itself, that is, indexing of the form FMT[T1[0]] is not allowed. If a variable name is used as an index, the index value is determined at execution time. Since the SCF software is written in ANSI C, indexing of multi-dimensional variables start at **zero**, not one, and the index values run from zero to dimension size minus one.

FORMAT 1 : FOR LOOP CONSTRUCT

The first algorithm format defined is the FOR loop construct. The FOR loop construct is used to iterate over a series of algorithm steps and is of the form:

```
FOR LOOP_VAR = START TO END
BEGIN
      algorithm step(s) to iterate
END
```

There must be at least one algorithm step contained between the BEGIN-END block. The tokens **FOR**, **TO**, **BEGIN** and **END** are SCF reserved keywords. **LOOP_VAR** is referred to as the looping variable and must be one of the defined input, temporary or output variables. The tokens **START** and **END** can either be a number or a variable name. If the token is a number, only positive values can be specified. If the token is a variable name, the referenced value must be a scalar quantity. At execution time, the SCF software **assumes** that the **START** value is less than or equal to the **END** value. The SCF software supports a **BREAK** statement within a FOR loop construct. This statement forces an immediate exit from the innermost enclosing FOR loop and is specified

```
BREAK
```

The following example loops over a 2-D variable called **VALUES** and sets each element of the variable to the product of the looping variables:

```
START = 0
STOP = 3
FOR T1 = START TO STOP
BEGIN
  FOR T2 = 0 TO 2
  BEGIN
    VALUES[T1][T2] = T1 * T2
  END
END
```

In this example, **VALUES** is a 4 x 3 matrix. Notice that the start values for both FOR loops are set to zero. This is necessary since the SCF software is written in ANSI C where indexing of multi-dimensional variables start at **zero**, not one. Also note that the stop values are one less than the length of the dimensions (4 x 3). This is necessary since the loop is iterated until the looping variable is equal to the stop value.

FORMAT 2 : CONDITIONAL TEST CONSTRUCT

The second algorithm format defined is the IF-ELSE-ENDIF construct. This construct is used to execute a series of algorithm steps based upon the result of a numerical comparison. The IF-ELSE-ENDIF construct can take one of the two forms shown below:

```
IF (VAR_1 oper VAR_2)
   algorithm step(s)
ELSE
  algorithm step(s)
ENDIF
```

IF (VAR_1 **oper** VAR_2)
  algorithm step(s)
ENDIF

There must be at least one algorithm step contained between the IF-ELSE, ELSE-ENDIF and IF-ENDIF blocks. The tokens **IF**, **ELSE** and **ENDIF** are SCF reserved keywords. The left and right parenthesis must be specified in the IF statement. **VAR_1** and **VAR_2** are referred to as comparison values. A comparison value can either be a number or a variable name. If the comparison value is a variable name, the referenced value must be a scalar quantity. The token **oper** should be replaced one of the following symbols:

OPER	MEANING
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

In the case of the IF-ELSE-ENDIF construct, if the conditional test succeeds, the algorithm steps between the IF-ELSE block are executed. If the conditional test fails, the algorithm steps between the ELSE-ENDIF block are executed. In the case of the IF-ENDIF construct, if the conditional test succeeds, the algorithm steps between the IF-ENDIF block are executed. If the conditional test fails, no algorithm steps are executed.

FORMAT 3 : VARIABLE ASSIGNMENT

The simplest format for an algorithm step initializes the resultant variable and is of the form:

RESULT = value

where **RESULT** must be one of the defined input, temporary or output variables and **value** can be a variable name, a number that is specified in decimal, floating point or exponential format or a format conversion string ("%10.2e"). If the value is a variable name, the referenced value must be a scalar quantity. The SCF software currently does not support the unary operator. The same function can be accomplished by assigning a temporary variable the value -1 and then multiplying by this variable or by using the **NEGATIVE** SCF function call (see below). If a constant is to be utilized in a mathematical operation, it must be assigned to a temporary variable and referenced through the temporary variable. The SCF software provides a built-in print function which can serve as a mini-debugger so that the results of the algorithm can be displayed on the screen as the steps are being executed. The print function can also serve as a means of dumping the results of the computations. The print function takes two arguments, the format conversion string and the variable to be printed.

FORMAT 4 : STANDARD MATHEMATICAL OPERATORS

The fourth algorithm format defined should be used when the mathematical operations (+, -, *, /) are utilized and is of the form:

RESULT = VAR_1 **oper** VAR_2

where **oper** should be replaced by one of the 4 mathematical symbols listed above, **RESULT** must be one of the defined input, temporary or output variables and **VAR_1** and **VAR_2** must be either a user defined variable or an internal SCF variable. As an example, the algorithm step to multiply the variable BX by the variable BY and place the result in the variable TP is written

TP = BX * BY

Even though the SCF software supports a storage class up to ten dimensions (10-D), these mathematical operations work on scalar-scalar, vector-scalar and vector-vector quantities. To process variables of higher dimensions, the user should refer to the sections describing matrix and tensor operations supported by the SCF software.

An IDFS vector quantity is defined as a 1-D data item. If variables A, B and R are IDFS vectors then vector-vector operations are defined as

R[i] = A[i] **oper** B[i]

for all i where i runs from 0 to the vector length minus one. The resultant variable **R** is a vector of equal length to **A** or **B**. For vector-vector operations, the vector lengths of the operands must be the same. If the two vector operands are not the same length, each element of the resultant variable will be set to the predefined value OUTSIDE_MIN. If variables **A**, **B** and **R** are scalars then scalar-scalar operations are defined as

R = A **oper** B

Lastly, if the variables **A** and **R** are IDFS vectors and the variable **B** is a scalar then scalar-vector operations are defined as

R[i] = A[i] **oper** B

for all i where i runs from 0 to the vector length minus one. The resultant variable **R** is a vector of equal length to **A**.

FORMAT 5 : FUNCTION CALLS

The fifth and final algorithm format defined is used when a function is being called. The function can either be a pre-defined SCF function or a user defined function. A function can either return nothing (a void function) or return a single quantity. The algorithm step format for a non-void function is written as

RESULT = **FUNC** (VAR_1,VAR_2,...VAR_N)

where **RESULT** must be one of the defined input, temporary or output variables and **VAR_1**, **VAR_2**, ... **VAR_N** must be either a user defined variable or an internal SCF variable. For a void function, this algorithm step is written as

FUNC (VAR_1,VAR_2,...VAR_N)

where **VAR_1**, **VAR_2**, ... **VAR_N** must be either a user defined variable or an internal SCF variable. A function can have a multi-variable input. If the function call does not fit on a single line of input, the line may be continued by use of the line continuation character '\' placed at the end of the line before any in-line comment. This character must appear after the left parenthesis in the function call in order for the equation to be parsed correctly.

SCALAR AND VECTOR FUNCTION CALLS

The pre-defined SCF functions that work on scalar and IDFS vector data items are briefly described in the table below. Each of the defined ASCII function definitions are reserved words within the SCF software and cannot be used as variable names.

FUNCTION DEFINITIONS			
FUNCTION	**OPERATION**	**INPUTS**	**USAGE**
DOT	dot product	2	X = DOT (VAR_1, VAR_2)
CROSS	cross product	2	X = CROSS (VAR_1, VAR2)
MAG	magnitude	1	X = MAG (VAR_1)
SQRT	square root	1	X = SQRT (VAR_1)
SIN	sine, input in radians	1	X = SIN (VAR_1)
COS	cosine, input in radians	1	X = COS (VAR_1)
TAN	tangent, input in radians	1	X = TAN (VAR_1)
ASIN	arc sine $-\pi/2$ to $\pi/2$	1	X = ASIN (VAR_1)
ACOS	arc cosine 0 to π	1	X = ACOS (VAR_1)
ATAN	arc tangent $-\pi/2$ to $\pi/2$	1	X = ATAN (VAR_1)
ATAN2	arc tangent $-\pi$ to π	2	X = ATAN2 (VAR_1,VAR_2)
EXP	e^{VAR_1}	1	X = EXP (VAR_1)
POW	$VAR\_1^{VAR\_2}$	2	X = POW (VAR_1,VAR_2)
LOGE	$\log_e$	1	X = LOGE (VAR_1)
LOG10	$\log_{10}$	1	X = LOG10 (VAR_1)
POLY	polynomial expansion	ORDER + 3	X = POLY(ORDER,COEF_0,...COEF_N,VAR_1)
SUM_V	vector element summation	1	X = SUM_V (VAR_1)
DEG_TO_RAD	degrees to radians	1	X = DEG_TO_RAD (VAR_1)
RAD_TO_DEG	radians to degrees	1	X = RAD_TO_DEG (VAR_1)
PRINT	print variable's value	2	PRINT (VAR_1, VAR_2)
WALL_CLOCK	return time elements	2	X = WALL_CLOCK (VAR_1, VAR_2)
ROUND	integral rounding of data	2	X = ROUND (VAR_1, VAR_2)
MERGE	merge values of two arguments	2	X = MERGE (VAR_1, VAR_2)
VARIABLE_SIZE	the size of the argument	1	X = VARIABLE_SIZE (VAR_1)
BIN_DATA	bin the data	8	X = BIN_DATA (VAR_1,VAR_2,...,VAR_8)
INT	integral part of value	1	X = INT (VAR_1)
FRACT	fractional part of value	1	X = FRACT (VAR_1)

FUNCTION DEFINITIONS			
FUNCTION	**OPERATION**	**INPUTS**	**USAGE**
ABS	absolute value	1	X = ABS (VAR_1)
NEGATIVE	multiply value by -1	1	X = NEGATIVE (VAR_1)
MODULUS	modulus	2	X = MODULUS (VAR_1, VAR_2)

Only four of the SCF function calls check for illegal input. For the **LOG10** and **LOGE** functions, if the input value is less than or equal to zero, the result is set to the predefined value OUTSIDE_MIN. For the **MODULUS** function, if the value for the second argument is equal to zero, the result is set to the predefined value OUTSIDE_MIN. For the **SQRT** function, if the input value is less than zero, the result is set to the predefined value OUTSIDE_MIN. Most of the above functions are self-explanatory and are at times taken verbatim from the C computing language. Several, however require more discussion than the above table allows and these are given individually below.

POLY

The function **POLY** expands an N^{th} order polynomial about the scalar value VAR_1. The inputs to **POLY** are the order of the polynomial, the polynomial coefficients beginning with the constant term, followed by the first order coefficient, etc., and the variable which is to be expanded. The algorithm used by the function is

$$\text{POLY} = \sum_{i=0}^{i=ORDER} COEF[i] * (VAR\_1)^{i}$$

where it should be understood that COEF_0 is COEF[0], COEF_1 is COEF[1], etc.

ATAN2

The function **ATAN2** is the standard C **atan2** function. It is equivalent to

$$\text{ATAN2} = \text{atan (VAR\_1/VAR\_2)}$$

with the exception that the result has no ambiguity in the angular quadrant.

SUM_V

The function **SUM_V** returns the summation of the elements in a vector variable. The algorithm used is

$$\sum_{i=0}^{i=N-1} VAR[i]$$

where N is the vector length and VAR is the vector variable. If the variable is not a vector then the routine will echo back the input variable.

PRINT

The function **PRINT** prints the current value for VAR_2 using the format conversion string defined in VAR_1. The format conversion string is the standard C conversion specification string that is used to display

formatted output. Since all variables used by the SCF software are in floating point format, it is advised that the "%f", "%e", or "%g" conversion characters be used; otherwise, an invalid value may be displayed for the variable being printed.

The format conversion string must be assigned to a temporary variable that is of a 1-D storage class and is at least 4 elements in size. The format conversion string can be up to thirty-one (31) characters in length, leaving one character for the string terminator character. The quotation marks must be specified but are not included when determining the length of the string. Some examples of format conversion strings are given below:

```
T1 = "%10.2e"

T1 = "\nValue = %10.2e"
```

WALL_CLOCK

The function **WALL_CLOCK** takes two time parameters, VAR_1 (in milliseconds) and VAR_2 (in nanoseconds) and returns the hour, minute and seconds time elements. The resultant variable must be at least three elements in size since three values are returned. The seconds value contains a fraction component, which is expressed in nanoseconds. The algorithm steps listed below display the time of day in the format hour:minute:seconds

```
FMT1 = "%02.0f:"
FMT2 = "%012.9f\n"
TCLOCK = WALL_CLOCK (SMILLI, SNANO)
PRINT (FMT1, TCLOCK[0])
PRINT (FMT1, TCLOCK[1])
PRINT (FMT2, TCLOCK[2])
```

In the above example, **SMILLI** and **SNANO** are internal SCF variables that are used to specify the time duration covered at each iteration of the algorithm.

ROUND

The function **ROUND** will round the value for VAR_1 to an integral value in floating point format. The value will either be rounded up (ceiling) to the least integral value greater than or equal to VAR_1 or will be rounded down (floor) to the greatest integral value less than or equal to VAR_1. The rounding scheme used is defined by the value for VAR_2. If the ceiling scheme is desired, VAR_2 should be set to 1. If the floor scheme is desired, VAR_2 should be set to 2. The algorithm steps listed below display the results of both rounding schemes for the data value BX.

```
TFMT1 = "\nVALUE = %f"
FLAG = 1
CEIL = ROUND (BX, FLAG)
```

```
PRINT (TFMT1, CEIL)
FLAG = 2
FLOOR = ROUND (BX, FLAG)
PRINT (TFMT1, FLOOR)
```

MERGE The function **MERGE** will merge the two arguments VAR_1 and VAR_2, element by element, filling in missing data when possible. If the operands are not the same length, the resultant will be set to the predefined value OUTSIDE_MIN. If the length of the resultant is greater than the length of the operand, the function will be performed up to the size of the operand. The remainder of the elements in the resultant will be set to the predefined value OUTSIDE_MIN. The **MERGE** function works as follows: if both values are present, an average of the two values is returned. If only one value is present, the resultant is set equal to the value found for that element. If both values are missing, the resultant is set to the predefined value OUTSIDE_MIN.

VARIABLE_SIZE The function VARIABLE_SIZE will return the size of the argument specified; that is, the number of data samples that are pertinent to the argument in question. Normally, for a scalar quantity, this value will be one. However, in the case of input variables, if the scalar quantity is intermixed with vector quantities, the value returned by this SCF function will be equal to the size of the input variable with the largest sample size. This is necessary since data for scalar input variables will be retrieved based upon the time of each element in the sweep for the control vector.

BIN_DATA The function **BIN_DATA** is pertinent to vector (1-D) quantities and will return a new data array that is created from the data array specified in argument VAR_6. The number of elements (bins) in the new data array is specified in argument VAR_3. This value must be greater than zero; otherwise, the resultant is set to the predefined value OUTSIDE_MIN. In addition, if this value is greater than the size of the resultant variable, the resultant is set to the predefined value OUTSIDE_MIN. The scan range the resultant array must cover is specified in arguments VAR_1 and VAR_2, with VAR_1 being the start value and VAR_2 being the stop value. VAR_1 must be less than or equal to VAR_2; otherwise, the resultant is set to the predefined value OUTSIDE_MIN. The bins can be either linearly (1) or logarithmically (2) spaced. The spacing is specified in argument VAR_4. If the start value (VAR_1) or the stop value (VAR_2) is less than or equal to zero and if logarithmic spacing was requested (VAR_4), the resultant is set to the predefined value OUTSIDE_MIN. The data samples can be stored in the newly created data array in one of two ways, point storage (1) or band storage (2). The storage scheme to use is specified in argument VAR_5. Data in a vector data set are taken as a function of a variable M (referred to as the scanning variable). These scan (SWEEP_STEP) values must be specified in argument VAR_7. The width of each scan value must be

specified in argument VAR_8. If the user selects the point storage scheme, the data is stored by the center variable M. If the center variable M is located between the upper and lower edge values of a given bin, the data value is placed only in this bin. If the user selects the band storage scheme, data is placed in all bins which are fully or partially contained within the range M - VAR_8/2 to M + VAR_8/2. The data is multiplied by the percentage of the bin covered by the range before the data is placed into the bin.

MODULUS The function **MODULUS** will return the remainder of VAR_1 with respect to VAR_2; that is, the remainder (r) is one of the numbers that differ from VAR_1 by an integral multiple of VAR_2. The magnitude of the remainder is less than that of VAR_2; its sign agrees with that of VAR_1. The algorithm steps listed below will result in the value 1 being displayed since 25 modulus 4 is equal to one.

```
TFMT1 = "\nVALUE = %.0f"
VAL = 25
MOD_VAL = 4
ANSWER = MODULUS (VAL, MOD_VAL)
PRINT (TFMT1, ANSWER)
```

MATRIX FUNCTION CALLS

The pre-defined SCF functions that work on matrix data items (2-D) are briefly described in the table below. Each of the defined ASCII function definitions are reserved words within the SCF software and cannot be used as variable names. In the calling sequences provided, variables **A**, **B** and **C** refer to matrix quantities and variable **X** refers to a scalar quantity. Most of the functions described in the table that follows are self-explanatory. However, following the table, some additional information will be provided which explains some assumptions/requirements for the function call in question.

FUNCTION DEFINITIONS			
FUNCTION	**OPERATION**	**INPUTS**	**USAGE**
ADD_MATRIX	matrix addition	2	C = ADD_MATRIX (A, B)
SUBTRACT_MATRIX	matrix subtraction	2	C = SUBTRACT_MATRIX (A, B)
MULTIPLY_MATRIX	matrix multiplication	2	C = MULTIPLY_MATRIX (A, B)
TRANSPOSE_MATRIX	transpose of a matrix	1	C = TRANSPOSE_MATRIX (A)
TRACE_MATRIX	trace of a square matrix	1	X = TRACE_MATRIX (A)
IDENTITY_MATRIX	identity matrix of order X	1	X = IDENTITY_MATRIX (A)
LOWER_TMATRIX	lower triangular matrix	1	C = LOWER_TMATRIX (A)
UPPER_TMATRIX	upper triangular matrix	1	C = UPPER_TMATRIX (A)
LU_TMATRIX	lower & upper triangular matrix	1	C = LU_TMATRIX (A)
DETERMINANT	determinant of a matrix	1	X = DETERMINANT (A)
INVERSE_MATRIX	inverse of a matrix	1	C = INVERSE_MATRIX (A)
AUGMENT_COL_MATRIX	matrix column augmentation	2	C = AUGMENT_COL_MATRIX (A, B)
AUGMENT_ROW_MATRIX	matrix row augmentation	2	C = AUGMENT_ROW_MATRIX (A, B)

ADD_MATRIX

The function **ADD_MATRIX** adds two matrices (**A** + **B**), element by element, and returns the result in matrix **C**. The two input matrices and the resultant matrix must all be the same size (n x m).

SUBTRACT_MATRIX

The function **SUBTRACT_MATRIX** subtracts two matrices (**A - B**), element by element, and returns the result in matrix **C**. The two input matrices and the resultant matrix must all be the same size (n x m).

MULTIPLY_MATRIX

The function **MULTIPLY_MATRIX** can take on one of two forms. In the first form

C = **MULTIPLY_MATRIX** (A, B)

two matrices (**A** * **B**) are multiplied and the result is returned in matrix **C**. If matrix **A** is of the size n x m, matrix **B** must be of the size m x p. The resultant matrix, termed the "inner-product", must be of the size n x p.

In the second form

C = **MULTIPLY_MATRIX** (A, X)

each element of the matrix **A** is multiplied by the scalar value **X** and the result is placed in the matrix **C**. In this case, the input matrix and the resultant matrix must be the same size (n x m).

TRANSPOSE_MATRIX

The function **TRANSPOSE_MATRIX** returns the transpose of the input matrix **A**. The transpose of a matrix is the matrix that results when the rows are written as columns (or when the columns are written as rows); therefore, if the input matrix is of size n x m, the resultant matrix must be of size m x n.

TRACE_MATRIX

The function **TRACE_MATRIX** returns the trace of the input matrix **A**. This scalar value is computed by summing the elements on the main diagonal of the matrix. The input matrix must be a square matrix; that is, the input matrix must be of the size n x n.

IDENTITY_MATRIX

The function **IDENTITY_MATRIX** converts the input matrix **A** into an identity matrix. An identity matrix is a matrix in which the elements on the main diagonal are set to one and all other elements are set to zero. The input matrix must be a square matrix; that is, the input matrix must be of the size n x n. The scalar value that is returned by this function is the order of the identity matrix, which depends upon the size of the input

matrix.

LOWER_TMATRIX

The function **LOWER_TMATRIX** returns the lower triangularization of the input matrix **A**. The lower triangular matrix is a matrix in which all elements above the diagonal are zero. The input matrix and the resultant matrix must be square matrices of the same size; that is, the matrices must be of the size n x n.

UPPER_TMATRIX

The function **UPPER_TMATRIX** returns the upper triangularization of the input matrix **A**. The upper triangular matrix is a matrix in which all elements below the diagonal are zero. The input matrix and the resultant matrix must be square matrices of the same size; that is, the matrices must be of the size n x n.

Note: Of the entire set of LU (lower/upper triangular) pairs whose product equals the input matrix, the SCF software has chosen the pair in which the upper triangular matrix has only ones on its diagonal.

LU_TMATRIX

The function **LU_TMATRIX** returns the lower and upper triangularization of the input matrix **A** in a single resultant matrix. The input matrix and the resultant matrix must be square matrices of the same size; that is, the matrices must be of the size n x n. For an input matrix of size 4 x 4, the combination resultant matrix is of the form:

$L_{11}\ U_{12}\ U_{13}\ U_{14}$
$L_{21}\ L_{22}\ U_{23}\ U_{24}$
$L_{31}\ L_{32}\ L_{33}\ U_{34}$
$L_{41}\ L_{42}\ L_{43}\ L_{44}$

DETERMINANT

The function **DETERMINANT** returns the determinant of the input matrix **A**. The input matrix must be a square matrix; that is, the input matrix must be of the size n x n. The determinant of a square matrix is a number (scalar value).

INVERSE_MATRIX

The function **INVERSE_MATRIX** returns the inverse matrix of the input matrix **A**. The input matrix and the resultant matrix must be square matrices of the same size; that is, the matrices must be of the size n x n.

AUGMENT_COL_MATRIX

The function **AUGMENT_COL_MATRIX** merges the two input matrices **A** and **B**. The input matrices and the resultant matrix must all contain the same number of rows. The number of columns defined in the resultant matrix must equal the number of columns in input matrix **A** + the number of columns

in input matrix **B**.

The contents of the first input matrix **A** are copied into the resultant matrix. The contents of the second input matrix **B** are copied into the resultant matrix, to the right-hand side of input matrix **A**. In other words, if matrix **A** contained

```
3  -1   2
1   2   3
2  -2  -1
```

and matrix **B** contained

```
12
11
 2
```

C = **AUGMENT_COL_MATRIX** (A, B) would result in the following matrix

```
3  -1   2   12
1   2   3   11
2  -2  -1    2
```

AUGMENT_ROW_MATRIX

The function **AUGMENT_ROW_MATRIX** merges the two input matrices **A** and **B**. The input matrices and the resultant matrix must all contain the same number of columns. The number of rows defined in the resultant matrix must equal the number of rows in input matrix **A** + the number of rows in input matrix **B**.

The contents of the first input matrix **A** are copied into the resultant matrix. The contents of the second input matrix **B** are copied into the resultant matrix, to the bottom of input matrix **A**. In other words, if matrix **A** contained

```
3  -1   2
1   2   3
2  -2  -1
```

and matrix **B** contained

```
12  11   2
```

C = **AUGMENT_ROW_MATRIX** (A, B) would result in the following matrix

```
 3   -1   2
 1    2   3
 2   -2  -1
12   11   2
```

TENSOR FUNCTION CALLS

The pre-defined SCF functions that work on tensor data items (3-D up to 10-D) are briefly described in the table below. Each of the defined ASCII function definitions are reserved words within the SCF software and cannot be used as variable names. In the calling sequences provided, variable **TA** refers to a tensor quantity. Most of the functions described in the table that follows are self-explanatory. However, following the table, some additional information will be provided which explains some assumptions/requirements that are required for the function call in question.

In the SCF tensor function definitions, the specification of a requested dimension may be required. Dimension numbers are expected to start at 1, going from left-to-right indexing. That is, if a 3-D tensor of size 4 x 3 x 2 is defined, dimension = 1 corresponds to the dimension of length 4, dimension = 2 corresponds to the dimension of length 3, and dimension = 3 corresponds to the dimension of length 2.

FUNCTION DEFINITIONS			
FUNCTION	**OPERATION**	**INPUTS**	**USAGE**
TENSOR_SUM	straight summation	2	X = TENSOR_SUM (TA, VAR_2)
TENSOR_AVG	straight average	2	X = TENSOR_AVG (TA, VAR_2)
TENSOR_WSUM	weighted summation	3	X = TENSOR_WSUM (TA, VAR_2, VAR_3)
TENSOR_WAVG	weighted average	3	X = TENSOR_WAVG (TA, VAR_2, VAR_3)
TENSOR_MASK	applying a mask	2	X = TENSOR_MASK (TA, VAR_2)
TENSOR_EXTRACT	extract elements from tensor	4	X = TENSOR_EXTRACT (TA, VAR_2, VAR_3, VAR_4)
TENSOR_INSERT	insert elements into tensor	4	X = TENSOR_INSERT (TA, VAR_2, VAR_3, VAR_4)
TENSOR_INTEGRAL	tensor integration	6	X = TENSOR_INTEGRAL (TA, VAR_2, VAR_3, VAR_4, VAR_5, VAR_6)

TENSOR_SUM

The function **TENSOR_SUM** is used to collapse (reduce) the input tensor **TA** along a single specified dimension **VAR_2** using a straight summation of the elements. Only those data values that are within the valid IDFS data range are included in the summation. If no values are found, the element of the resultant being processed is set to the value defined as OUTSIDE_MIN. The argument **VAR_2** is a scalar quantity that specifies which **dimension**, starting with one, is to be reduced. The input tensor **TA** must be of a rank greater than or equal to two. The size of the resultant is based upon the size of the input tensor. The following table summarizes the resultant sizes:

INPUT TENSOR	RANK OF INPUT	RESULTANT	RANK OF RESULTANT
row vector	2 (1 x M)	scalar	0
column vector	2 (N x 1)	scalar	0
matrix	2 (N x M)	row or column vector	2
tensor	3 - 10	tensor	2 – 9

TENSOR_AVG

The function **TENSOR_AVG** is used to collapse (reduce) the input tensor **TA** along a single specified dimension **VAR_2** using a straight average of the elements. Only those data values that are within the valid IDFS data range are included in the summation; that is, the summed value ends up being divided by the number of data samples included, not by the size of the dimension being reduced. If no values are found, the element of the resultant being processed is set to the value defined as OUTSIDE_MIN. The argument **VAR_2** is a scalar quantity that specifies which **dimension**, starting with one, is to be reduced. The input tensor **TA** must be of a rank greater than or equal to two. The size of the resultant is based upon the size of the input tensor. The size requirements are explained in the write-up of the tensor function **TENSOR_SUM**.

TENSOR_WSUM

The function **TENSOR_WSUM** is used to collapse (reduce) the input tensor **TA** along a single specified dimension **VAR_3** using a weighted summation of the elements. The second argument **VAR_2** is a 1-D array that contains the weight factors to be applied (multiplied) to each element in the dimension being collapse. Only those data values/weight factors that are within the valid IDFS data range are included in the summation. If no data values are found or if all the weight factors are outside of the valid IDFS data range, the element of the resultant being processed is set to the value defined as OUTSIDE_MIN. The argument **VAR_3** is a scalar quantity that specifies which **dimension**, starting with one, is to be reduced. The input tensor **TA** must be of a rank greater than or equal to two. The size of the resultant is based upon the size of the input tensor. The size requirements are explained in the write-up of the tensor function **TENSOR_SUM**.

TENSOR_WAVG

The function **TENSOR_WAVG** is used to collapse (reduce) the input tensor **TA** along a single specified dimension **VAR_2** using a weighted average of the elements. The second argument **VAR_2** is a 1-D array that contains the weight factors to be applied (multiplied) to each element in the dimension being collapse. Only those data values/weight factors that are within the valid IDFS data range are included in the summation. If no data values are found or if all the weight factors are outside of the valid IDFS data range, the element of the resultant being processed is set to the value defined as OUTSIDE_MIN. The denominator used to compute the average

is the summation of the weight factors, not the number of data samples included in the summation. The argument **VAR_2** is a scalar quantity that specifies which **dimension**, starting with one, is to be reduced. The input tensor **TA** must be of a rank greater than or equal to two. The size of the resultant is based upon the size of the input tensor. The size requirements are explained in the write-up of the tensor function **TENSOR_SUM**.

TENSOR_MASK

The function **TENSOR_MASK** is used to apply the tensor mask **VAR_2** to the input tensor **TA** in order to produce the resultant. The input tensor **TA**, the mask **VAR_2**, and the resultant must be the same rank and the same size; that is, the lengths of each dimension must be the same for the inputs and the resultant. The tensor mask **VAR_2** is simply a series of 1's and 0's, indicating which elements of the input tensor **TA** are to be copied into the resultant. 1 means copy the element; 0 means do not copy the element (in its place will be the special value OUTSIDE_MIN).

TENSOR_EXTRACT

The function **TENSOR_EXTRACT** is used to extract elements from the input tensor **TA** and to store the extracted elements into the resultant. The variable **VAR_2** must be a scalar quantity and is used to indicate the number of dimensions being specified in the **VAR_3** and **VAR_4** input parameters. The value for **VAR_2** must be equal to the rank of the input tensor **TA**. The variable **VAR_3** must be defined as a 1-D array of values, as well as **VAR_4**. The start index **VAR_3**, along with the stop index **VAR_4**, taken together, define the subset of data to be extracted along each dimension of the input tensor **TA**. If the start and stop index values are the same for a particular dimension, all data along that specific index are extracted from that particular dimension; otherwise, the value for the start index must be less than the value for the stop index and the index values represent a subset (range) of data values to be extracted from that particular dimension. The rank of the resultant is based upon the start/stop index values provided. For each pair of start/stop index values that are the same, the rank of the resultant should be decremented by one. For example, the start/stop index values defined as:

start_ind[4] = {0, 0, 0, 2} stop_ind[4] = {3, 0, 2, 2}

should be inferred to result in a 2-D tensor that is 4 x 3 in size, where 4 represents the range 0 - 3 from the first dimension and 3 represents the range 0 - 2 from the third dimension of the original tensor. The second dimension is held constant at index value 0 and the fourth dimension is held constant at index value 2 so that all data values with index values of [0-3][0][0-2][2] are extracted.

TENSOR_INSERT

The function **TENSOR_INSERT** is used to insert elements into the tensor specified as the resultant using the data elements provided in the input tensor **TA**. The variable **VAR_2** must be a scalar quantity and is used to indicate the number of dimensions being specified in the **VAR_3** and **VAR_4** input parameters. The value for **VAR_2** must be equal to the rank of the resultant. The variable **VAR_3** must be defined as a 1-D array of values, as well as **VAR_4**. The start index **VAR_3**, along with the stop index **VAR_4**, taken together, define the subset of data to be inserted along each dimension for the resultant tensor. The rank of the insertion tensor **TA** is of no importance; what is required is that the total number of elements to be inserted is equal to the number of elements defined by the range (subset) represented by the start/stop index values. For example, the start/stop index values defined as:

start_ind[4] = {0, 0, 0, 2} stop_ind[4] = {3, 0, 2, 2}

should be inferred to mean that 4 x 1 x 3 x 1 = 12 elements are to be inserted into the resultant tensor; that is, sequential elements from **TA** are placed into the resultant tensor where the second dimension is held constant at index value 0 and the fourth dimension is held constant at index value 2 so that all data values with index values of [0-3][0][0-2][2] are set in the resultant tensor. The first value held in **TA** will be placed into element [0][0][0][2], the next value held in **TA** will be placed into element [0][0][1][2], etc. If the start and stop index values are not the same for a particular dimension, the value for the start index must be less than the value for the stop index.

TENSOR_INTEGRAL

The function **TENSOR_INTEGRAL** is used to integrate over the specified dimension **VAR_2** for the input tensor **TA**. For tensor data, only one dimension may be integrated over at a single time; therefore, the size of the resultant must match the dimensions left in tact. The integral is of the form

$$\int_{x=start}^{x=stop} f dx \Rightarrow \sum_{i} fi\Delta xi$$

The argument **VAR_2** is a scalar quantity that specifies the dimension number. Dimension numbers are expected to start at 1, going from left-to-right indexing. That is, if a tensor of size 4 x 3 x 2 were to be collapsed, dimension = 1 corresponds to the dimension of length 4, dimension = 2 corresponds to the dimension of length 3, and dimension = 3 corresponds to the dimension of length 2. The maximum value for **VAR_2** is equivalent to the rank of the input tensor **TA**.

The argument **VAR_3** is a 1-D variable which holds the center values that are to be used for the integration. There are three basic rules that the center values must abide by: (1) no two successive elements can be redundant (the same value), (2) the center values must be constantly increasing or constantly decreasing, not random, and (3) the number of elements in the 1-D variable must be equal to the length of the dimension being collapsed.

From the center values provided in **VAR_3**, bin widths are created. These bins are logarithmically or linearly spaced, based upon the value assigned to the scalar argument **VAR_4.** A value of 1 specifies linear spacing and a value of 2 specifies logarithmic spacing of the bins.

The start and stop limits of integration are specified in the two scalar arguments **VAR_5** and **VAR_6**, respectively. The start value does not have to be less than the stop value; however, this function does not handle wrap-around scenarios, nor does it perform split integrals. If the user is looking to perform a 2 piece integral (e.g. 315° - 360°, 0° - 15°), the user must make two separate calls for the two distinct regions – **DO NOT** specify the start value as 315 and the stop value as 15!

For the integral, only those data values within the valid IDFS data range are included. If no values are found, the element of the resultant being processed is set to the value defined as OUTSIDE_MIN.

The rank of the resultant depends upon the rank in the input tensor **TA.** If the input tensor is a tensor with a rank >= 3, then the rank of the resultant tensor is one less (>= 2). If the input tensor is a matrix, then the resultant tensor is either a row or column vector, based upon which dimension is being integrated over. Finally, if the input tensor is a 1-D variable, a row vector, or a column vector, then the resultant is a scalar quantity. Note that if a row or column vector is being integrated over, the dimension specified in **VAR_2** must NOT be the dimension which has a length of 1 (1 x M or N x 1)!

EXAMPLE SCF File

The following is an example of an SCF file which computes the magnitude of B, Phi and Theta from the specified magnetometer data.

```
/******************** Example SCF          ************************/
/*
Compute Magnitude of B, Phi and Theta          /* title
/*
/******************** Contact section ************************************
5                                              /* number of contact lines
Dr. J. David Winningham                        /* contact
Southwest Research Institute
6220 Culebra Road
San Antonio, TX 78228-0510
INTERNET: david@cluster.space.swri.edu
/******************** Comment section ************************************
1                                             /* number of comment lines
Routine returns variables in the order B, Phi, Theta
/******************** Input Variables ************************************
3                                             /* number of input variables
/*------------------- Input 00 ------------------------------------------
BX                                            /* input variable name
TSS TSS-1 TEMAG TEMAG TMMO                    /* IDFS source
SENSOR 0                                      /* data type
1                                             /* number of tables for unit
0                                             /* tables to apply
=                                             /* table operators
VALID_MIN VALID_MAX                           /* lower and upper cutoff values
256 256                                       /* d_qual exclusion range
/*------------------- Input 01 ------------------------------------------
BY                                            /* input variable name
TSS TSS-1 TEMAG TEMAG TMMO                    /* IDFS source
SENSOR 1                                      /* data type
1                                             /* number of tables for unit
0                                             /* tables to apply
=                                             /* table operators
VALID_MIN VALID_MAX                           /* lower and upper cutoff values
256 256                                       /* d_qual exclusion range
/*------------------- Input 02 ------------------------------------------
BZ                                            /* input variable name
TSS TSS-1 TEMAG TEMAG TMMO                    /* IDFS source
SENSOR 2                                      /* data type
1                                             /* number of tables for unit
0                                             /* tables to apply
=                                             /* table operators
VALID_MIN VALID_MAX                           /* lower and upper cutoff values
256 256                                       /* d_qual exclusion range
/******************** Temp Variables ************************************
1                                             /* number of temporary variables
T1                                            /* temporary variable name
0                                             /* rank and length of dimension
/******************** Output Variables ***********************************
3                                             /* number of output variables
```

```
/*------------------- Output 00 ---------------------------------------------
B                                           /* output variable name
0                                           /* rank and length of dimension
/*------------------- Output 01 ---------------------------------------------
PHI                                         /* output variable name
0                                           /* rank and length of dimension
/*------------------- Output 02 ---------------------------------------------
THETA                                       /* output variable name
0                                           /* rank and length of dimension
/*
/******************** Equation Definition **********************************
PHI = ATAN2 (BY,BX)                         /* compute phi
PHI = RAD_TO_DEG (PHI)                      /* convert phi to degrees
BX = BX * BX
BY = BY * BY
T1 = BY + BX                                /* BX**2 + BY**2
BX = SQRT (T1)
THETA = ATAN2 (BX,BZ)                       /* compute theta
THETA = RAD_TO_DEG (THETA)                  /* convert theta to degrees
BZ = BZ * BZ
T1 = T1 + BZ                                /* BX**2 + BY**2 + BZ**2
B = SQRT (T1)                               /* compute B
```