LA-UR-19-30186
Approved for public release; distribution is unlimited.

Title: Sesame IO Library User Manual

Author(s): Young, Ginger Ann
Abhold, Hilary

Intended for: To distribute with software release.
Web

Issued: 2019-10-10 (rev.1)

LOS ALAMOS NATIONAL LABORATORY

Sesame IO Library User Manual

Version 8.4

LA-UR-19-30186

Hilary Abhold, Ginger Young
10/2/2019

Contents

1.0 Introduction

This document is a user manual for SES_IO, a low-level library for reading and writing sesame files. The purpose of the SES_IO library is to provide a simple user interface for accessing and creating sesame files that does not change across sesame format type (such as binary, ascii, and xml).

2.0 The Sesame Format

A sesame file contains equation of state (EOS) data on a implicit structured grid for use by hydrodynamic modeling codes. A sesame file is essentially a collection of material libraries. Each material library contains equation of state information as a set of pre-defined data tables (see Appendix A). Each data table contains a number of arrays of data defined on the same two-dimensional implicit structured grid.

3.0 Sesame IO Library Paradigm

The SES_IO library uses the concept of a sesame file handle. A sesame file is associated with a sesame file handle as it is opened. It is then setup, accessed, and closed through that file handle. More than one handle (each associated with one file) may be open at the same time. More than one file handle may refer to the same file.

4.0 Examples of Usage

This section contains examples of usage of SES_IO. All examples are written in the "C" programming language; see sections 6 and 7 for Fortran 90 and Java examples. The appropriate error checks are included in each example.

4.1 Opening a File

A sesame file is opened with the ses_open function. A ses_file_handle is returned.

Example:

```
#include "ses_defines.h"

/*  open the file */

ses_file_handle  the_handle = ses_open("sesame", 'R');
if (ses_is_valid(the_handle) == SES_FALSE) {
        /*  file not opened correctly – error */
}
```

Note that the file "ses_defines.h" must be included at the top of every file that uses SES_IO.

4.2 Setting up a File (Setup)

Before it can be accessed, a sesame file handle is setup using the ses_setup function. Once setup, a ses_file_handle may be accessed. A file handle that is setup properly is associated with a sesame material and a sesame table.

Example:

```
/*  setup a valid file handle  for material id 2030 and standard table id 301*/

ses_error_flag  didit_setup = ses_setup(the_handle, 2030, 301);
if (didit_setup != SES_NO_ERROR) {
        /*  file not setup correctly – error */
}
```

4.3 Access

A file may be opened for access in one of ways: for Read ('R'), Write ('W'), Append ('A') or Change ('C').

4.3.1 Read

A file may be read in one of two ways: using the *iterator* interface, or the *array* interface. Upon setup, the sesame library is pointed at the table associated with the given material and table. The iterator interface allows the user to iterate over all arrays in the data table associated with that table id without explicitly knowing how many arrays exist in the table.

The *array* interface assumes that the user knows the structure of the table and allows the user to directly access the data in the table.

4.3.1.1 Read with Iterator example

```
/*  reading a 301 table for material 2030 with the iterator interface */

ses_file_handle the_handle = ses_open("sesame", 'R');
if (ses_is_valid(the_handle) == SES_FALSE) {
        /*  invalid file handle – error */
}

ses_error_flag didit_setup = ses_setup(the_handle, 2030, 301);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}

ses_number array_size = 0;
ses_word_reference data_buffer = (ses_word_reference)NULL;

while (ses_has_next(the_handle) == SES_TRUE) {
        array_size = ses_array_size_next(the_handle);
        if (array_size <= 0) {
                /*  array size incorrect – error */
        }
        data_buffer = ses_read_next(the_handle);
        if (data_buffer == (ses_word_reference)NULL) {
                /*  error – did not read correctly */
        }
        /*  here, the user may do what they like with the data */
        /*  user should de-allocate memory  */
        free(data_buffer);
        data_buffer = (ses_word_reference)NULL;
}
```

4.3.1.2 Read with Array Interface Example:

```
/*  reading a 301 table for material 2030 with the array interface */

ses_file_file the_handle = ses_open("sesame", 'R');
if (ses_is_valid(the_handle) == SES_FALSE) {
        /*  invalid file handle – error */
}

ses_error_flag didit_setup = ses_setup(the_handle, 2030, 301);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}

/*  data in the 301 table is ordered as follows:
        nr
        nt
        density(1..nr)
        temperature (1..nt)
        pressure(1..nr*nt)
        energy(1..nr* nt)
        free energy(1..nr *nt)  */

ses_error_flag didit_read = SES_NO_ERROR;
ses_number nr, nt;

didit_read = ses_read_number(the_handle, &nr);
didit_read = ses_read_number(the_handle, &nt);

ses_word_reference array1 = malloc(sizeof(ses_word)*nr);

read_error = ses_read_1D(myHandle, array1, nr);
print_data("density", array1, nr);
free(array1);
array1=(ses_word_reference)NULL;

ses_word_reference array2 = malloc(sizeof(ses_word)*nt);

read_error = ses_read_1D(myHandle, array2, nt);
print_data("temperature", array2, nt);
free(array2);
array2=(ses_word_reference)NULL;

ses_word_reference array3 = malloc(sizeof(ses_word)*nr*nt);
read_error = ses_read_2D(myHandle, array3, nr, nt);
print_data("pressure", array3, nr*nt);

read_error = ses_read_2D(myHandle, array3, nr, nt);
```

```
print_data("energy", array3, nr*nt);

read_error = ses_read_2D(myHandle, array3, nr, nt);
print_data("free energy", array3, nr*nt);

free(array3);
array3=(ses_word_reference)NULL;
```

4.3.2 Write

A file may be written in one of two ways: using the *iterator* interface, or the *array* interface. The file may not exist previous to being written.

4.3.2.1 Write with Iterator Example:

```
/*  writing a 301 table for material 2030 with the iterator interface */

ses_file_handle the_handle = ses_open("sesame_write", 'W');
if (ses_is_valid(the_handle) == SES_FALSE) {
        /*  invalid file handle – error */
}

ses_number nr = 33;
ses_number nt = 3;
ses_number ntab = -1;
ses_error_flag didit_setup = ses_write_setup(the_handle, 2030, 301, nr, nt, ntab);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}

ses_number array_size = 0;
ses_error_flag didit_write = SES_NO_ERROR;
ses_word_reference data_buffer;

while (ses_has_next(the_handle)) {

        /*  user creates memory and sets array sizes here */
        /*  the user puts data into the data_buffer here */
        data_buffer[1] = 3.285;    /*  etc */
        array_size = 3;  //  what ever the size will be written
        didit_write = ses_write_next(the_handle, data_buffer, array_size);
        if (didit_write  != SES_NO_ERROR) {
                /*  error – did not write correctly */
        }
}
```

4.3.2.2 Write with Array Interface Example:

```
/*  writing a 301 table for material 2030 with the array interface */

the_handle = ses_open_f90("sesame_write_array", 'W');
if (ses_is_valid(the_handle) == SES_FALSE) {
        /*  invalid file handle – error */
}

ses_number nr = 33;
ses_number nt = 3;
ses_number ntab = -1;

ses_error_flag didit_setup = ses_write_setup(the_handle, 2030, 301, nr, nt, ntab);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}

/*  data in the 301 table is ordered as follows:
        nr
        nt
        density(1..nr)
        temperature (1..nt)
        pressure(1..nr)(1..nt)
        energy(1..nr)(1..nt)
        free energy(1..nr)(1..nt)  */

ses_error_flag didit_write = SES_NO_ERROR;

didit_write = ses_write_number(the_handle, nr);
didit_write = ses_write_number(the_handle, nt);

ses_word  density[nr];
ses_word temperature[nt];

/*  fill in the data to be written here */

for (int i = 0;  i < nr; i++) {
        density[i] = .409;  /*  etc */
}
for (int j = 0; j < nt; j++) {
        temperature[j] = .2342;  /*etc*/
}
didit_write = ses_write_1D(the_handle, &density, nr);
didit_write = ses_write_1D(the_handle, &temperature, nt);

ses_word  pressure[nr*nt], energy [nr*nt], free_energy [nr*nt];
```

```
/*  put some more values in the data to be written here */

for (int j = 0; j < nr*nt; j++) {
        pressure[j] = 3.2;  / *etc */
        energy[j] = 3.2;  / *etc */
        free_energy[j] = 3.2;  / *etc */
}

didit_write = ses_write_2D(the_handle, &pressure, nr, nt);
didit_write = ses_write_2D(the_handle, &energy, nr, nt);
didit_write = ses_write_2D(the_handle, &free_energy, nr, nt);
```

4.3.3 Append

A new material library may be appended to a file. The given file must exist, and may not contain the material specified in the ses_setup function.

Example:

```
/*  appending  a 301 table for new material 2000 to an existing file*/

ses_file_handle the_handle = ses_open("sesame_append", 'A');
if (ses_is_valid(the_handle) == SES_FALSE) {
        /*  invalid file handle – error */
}

ses_number nt = 3;
ses_number nr = 33;
ses_number ntab = -1;
ses_error_flag didit_setup = ses_write_setup(the_handle, 2000, 301, nr, nt, ntab);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}

ses_number array_size = 0;
ses_error_flag didit_write = SES_NO_ERROR;

while (ses_has_next(the_handle)) {
        array_size = ses_array_size_next(the_handle);
        if (array_size <= 0) {
                /*  array size incorrect – error */
        }
        /*  the user puts data into the data_buffer here */
        ses_word data_buffer[array_size];
        for (int i = 0;  i < array_size; i++) {
                data_buffer[i] = 3.285;    /*  etc */
        }
        didit_write = ses_write_next(the_handle, data_buffer, array_size);
        if (didit_write  != SES_NO_ERROR) {
                /*  error – did not write correctly */
        }
}
```

4.3.4 Change

Data in an array in a sesame table may be changed. The given file must exist, and must contain the material/table pair specified in the setup function.

Example:

```
/*  changing data in the energy array in  a 301 table for material 2030*/

ses_file_handle the_handle = ses_open("sesame_change", 'C');
if (ses_is_valid(the_handle) == SES_FALSE) {
        /*  invalid file handle – error */
}

ses_error_flag didit_setup = ses_setup(the_handle, 2030, 301);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}

ses_number array_size = 0;
ses_error_flag didit_skip = SES_NO_ERROR;
ses_label the_label;
ses_error_flag didit_write = SES_NO_ERROR;

ses_word_reference data_buffer;

while (ses_has_next(the_handle)) {
        the_label = ses_get_label(the_handle);
        array_size = ses_array_size_next(the_handle);
        if (strcmp(the_label, "e – energy (MJ/kg)") == 0) {
                /*  found the energy array */
                /*  put data into the buffer here, then write it to the file */

                data_buffer = malloc(sizeof(ses_word)*array_size);

                for (int i = 0;  i <  array_size; i++) {
                        data_buffer[i] = 343.344;  /*etc*/
                }
                didit_write = ses_change_next(the_handle, data_buffer, array_size);

        }
        else {
                didit_skip = ses_skip(the_handle);
        }
}
```

4.3.5 Setup Options

4.3.5.1 Array Order

SES_IO uses as the default a multi-dimension array order of "column major", as in Fortran 90. It also reads and writes 2D and 3D arrays using a one-dimensional array reference, so the array order is important. The C language uses a row major order for multiple dimension arrays. The order of the elements, when reading from a sesame file in C, must be converted from column major order (on the file) to row major order (used in C). Likewise, when writing a sesame file in C the array must be converted from row major order (used in C) to column major order (on the file).

To specify that the file will be read into arrays in row major order, the user can set the array order with the ses_set_array_order function.

Example:

```
/*  set up to read multi-dimensional arrays into row-major order arrays */

ses_error_flag didit_setup = ses_setup(the_handle, 2030, 301);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}


ses_error_flag didit_set_order= ses_set_array_order(the_handle, 'R');

if (didit_set_order != SES_NO_ERROR) {
        /*  did not set the array order properly – error */
}
```

To specify that the file will be written into arrays in column major order, the user sets the array order to column-major order.

Example:

```
/* set up to write row-major order multi-dimensional arrays into column-major order
    arrays on the file */

ses_error_flag didit_setup = ses_setup(the_handle, 2030, 301);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}


ses_error_flag didit_set_order= ses_set_array_order(the_handle, 'C');

if (didit_set_order != SES_NO_ERROR) {
        /*  did not set the array order properly – error */
}
```

4.3.5.2 Significant digits

SES_IO contains the option to truncate data read in from or written to a sesame file to a certain number of digits followed by 0's in the remaining decimal places. This can help the user community avoid floating point arithmetic precision errors, such as in the case when a computation produces a floating point number like 5.99999999e-12. Computations often provide the closest floating point number to a computed number because not all real numbers are representable in a fixed number of bits. Note that this is not a true significant digit read or write, where only the significant digits would be read or written.

To set the numbers read-in and written to a file to a constant number of significant digits, the ses_set_significant_digits function is used.

Example:

```
/*  set up to read or write 3 significant digits to a  file */

ses_error_flag didit_setup = ses_setup(the_handle, 2030, 301);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}


ses_error_flag didit_set_digits= ses_set_significant_digits(the_handle, 3);

if (didit_set_digits != SES_NO_ERROR) {
        /*  did not set significant digits properly – error */
}
```

4.3.5.3 Validation

SES_IO contains the option to inform the user when data that is not valid is read from or written to a file.

To set this option, the ses_set_validate function is used.

Example:

```
/*  set up to read or write with validation */

ses_error_flag didit_setup = ses_setup(the_handle, 2030, 301);
if (didit_setup != SES_NO_ERROR) {
        /*  did not set up properly – error */
}

ses_error_flag didit_set_valid= ses_set_validate(the_handle);
if (didit_set_valid != SES_NO_ERROR) {
        /*  did not set validation properly – error */
}
```

4.3.5.4 Material order

The user may specify that when a file is written, it will be written with the material ids in ascending order. To set this option, the ses_set_material_order function is used.

Example:

```
/*  set up to write materials in order */

ses_error_flag didit_set_materials = ses_set_material_order (the_handle);
if (didit_set_materials != SES_NO_ERROR) {
        /*  did not set validation properly – error */
}
```

4.4 Close

When finished, a sesame file handle is closed with the ses_close function.

Example:

```
/*  close the valid file handle */

ses_error_flag didit_close = ses_close(the_handle);
if (didit_close != SES_NO_ERROR) {
        /*  file not closed correctly – error */
}
```

4.5 Querying Sesame Files

The SES_IO library provides several functions useful for querying information about a sesame file.

4.5.1 Example for ses_date

The "ses_date" function returns a ses_string with the date information for the file.

Example:

```
/*  query the file date */

ses_string the_date = ses_date(the_handle);
if (the_date == (ses_string)NULL) {
        /*  function error – string not returned correctly */
}
```

4.5.2 Example for ses_version

The "ses_version" function returns a ses_string with the version information for the file.

Example:

```
/*  query the file version */

ses_string the_version = ses_version(the_handle);
if (the_version == (ses_string)NULL) {
        /*  function error – string not returned correctly */
}
```

4.5.3 Example for ses_get_materials

The “ses_get_materials” function returns a ses_material_id_reference, a reference to an array of ses_material_ids. This is a list of the materials on the file.

Example:

```
/*  return a list of the materials on the file */

long size = 0;
ses_material_id_reference the_materials = ses_get_materials(the_handle, &size);
if (the_materials == (ses_material_id_reference)NULL) {
        /*  function error – list not returned correctly */
}
if (size == 0) {
        /*  function error – size not returned correctly */
}

/*  print the list */
int i = 0;
for  (i=0; i < size; i++) {
        printf(“material [%d] is %d\n”, i, the_materials[i]);
}
```

4.5.4 Example for ses_get_table_ids

The "ses_get_table_ids" function returns a ses_table_id_reference, a reference to an array of ses_table_ids. This is a list of the materials on the file.

Example:

```
/*  after file is open and setup… */
/*  return a list of tables associates with a material on the file */

long size1;
ses_table_id_reference the_tables = ses_get_table_ids(the_handle, 2030, &size1);
if (the_tables == (ses_table_id_reference)NULL) {
        /*  function error – list not returned correctly */
}
if (size1 == 0) {
/*  function error – size not returned correctly */
}

/*  print the list */
int i = 0;
for  (i=0; i < size1; i++) {
        printf("table  [%d] for material 2030 is %d\n", i, the_tables[i]);
}
```

4.5.5 Example for ses_comments

The "ses_comments" function returns a ses_string* with any comments (from the sesame 101 table) on the file.

Example:

```
/*  after file is open and setup … */
/* retrieve comments */

ses_string the_comments = (ses_string)NULL;

ses_error_flag didit_get_comments = ses_comments(the_handle,  &the_comments);
if (didit_get_comments != SES_NO_ERROR) {
        /*  error – comments not retrieved */
}
```

4.6 Combining Sesame Files

The following function is used to combine two sesame files:

ses_error_flag ses_combine(ses_file_handle file1, ses_file_handle file2, ses_string new_filename);

4.7 User-Defined Tables

4.7.1 Defining a table with with ses_define_table

The ses_define_table function can be used with user written routines to define a new sesame table. Once a sesame table been defined, any file written with ses_io will write the file definition as a text string to the sesame file, so that future use of ses_io to read in that file will not need to define the table with ses_define_table.

Originally this functionality was only available when reading/writing binary files. Currently user-defined tables are also available for binary and sesame ASCII files. The functionality was shoe horned to add sesame ASCII files; there are some items a user needs to be aware.

The file definition written to a sesame file does contain binary characters. The ramifications of a string containing binary characters are that ASCII sesame files will contain unprintable characters. A user may choose to create binary files, if they find this offensive.

When creating a user-defined table for an ASCII sesame file, it must have a definition for nr and nt.

4.7.2 Defining a table with the “table_defs.sesio” file

Alternatively, putting the table definition into a “table_defs.sesio” file and letting the ses_io library find the definition can define a table. See appendix B for the parser format for user-defined tables.

There is a side effect of using table_defs.sesio. If this file is in a directory where you create or write other sesame files, the user table definition text string is added to those files. This may not be your desired outcome.

4.8 Error Handling

Most error handling in SES_IO can be done through the use of the *ses_error_flag* data type. The possible values of the ses_error_flag are defined as follows:

SES_NO_ERROR
SES_ARRAY_SIZE_ERROR
SES_MEMORY_ALLOCATION_ERROR
SES_OBJECT_CONSTRUCTION_ERROR
SES_NULL_OBJECT_ERROR
SES_OBJECT_COPY_ERROR
SES_OBJECT_DESTRUCTION_ERROR
SES_FUNCTION_FAIL_ERROR
SES_INVALID_FILE_HANDLE
SES_INVALID_MID
SES_OBJECT_READY_ERROR
SES_OPEN_ERROR
SES_INVALID_TID
SES_OBJECT_OUT_OF_RANGE
SES_SETUP_ERROR
SES_CLOSE_ERROR
SES_FILE_READY_ERROR
SES_FILE_WRITE_ERROR
SES_READ_ERROR
SES_WRITE_ERROR
SES_CHANGE_ERROR
SES_COMBINE_ERROR
SES_COMMENTS_ERROR
SES_DELETE_ERROR
SES_NOT_IMPLEMENTED
SES_INVALID_OPEN_TYPE
SES_DOUBLE_SIZE_ERROR
SES_TEST_ERROR
SES_APPEND_ERROR
SES_NO_DATA_ERROR
SES_INVALID_FILE_FORMAT_TYPE
SES_INVALID_ASCII_WORD_TYPE
SES_INVALID_NUM_PARAMETERS

Two functions are also available to do error handling: “ses_indicates_error”, which checks whether a given ses_error_flag is or is not an error, and “ses_print_error_condition”, which returns a string that contains text that describes the latest error.

Example:

```
/* Use ses_indicates_error to check for an error condition */

ses_error_flag  the_error_flag = ses_some_function(the_handle);
if (ses_indicates_error(the_error_flag)) {
        /*  you have an error – do your error handling here */
}
```

Example:

```
/*  use ses_print_error_condition to print a text string with the latest error */

ses_error_flag  the_error_flag = ses_some_function(the_handle);
if (ses_indicates_error(the_error_flag)) {
        /*  you have an error – do your error handling here */

        ses_string the_error = ses_print_error_condtion(the_handle);
        printf("The error is %s\n", the_error);
}
```

5.0 Using the Library

5.1 Linking to the Library

5.1.1 Linking to the C Library

SES_IO is released as a static library, libses.a. It will be found in $(exec_prefix)/lib.
To use the library, the user must include the file $(prefix)/include/ses_defines.h in files that use functions from the library.

5.1.2 Linking to the F90 Library

The user must link to the static C library, libses.a, as well as the F90 wrapper libraries, libsesf.a and libsesw.a. These will be found in $(exec_prefix)/lib.

5.1.3 Using the Java Library

The user must have the C library, libses.a, in the classpath, as well as the Java wrapper shared object libsesj.so. The user must have the Java Class SesIO (the *.class file) in the classpath as well.

5.2 Installation and Build

SES_IO is released (initially as a Unix library) and can be built as from a tar file, initially ses_VB0.0_08_25_2010.tar. Check the included README file for build and installation instructions.

6.0 Fortran 90 Examples

The F90 interface uses the F90 ses_io module. This file must be included with a "use ses_io" statement as the first line of any F90 routine that uses this library. The header file "ses_defines_f90.h" is used with every file containing code that accesses ses_io.

6.1 Opening a File

A sesame file is opened with the ses_open_f90 function. A ses_file_handle is returned.

Example:

```
! add in the ses_io module

use ses_io

! define f90 variables

SES_FILE_HANDLE :: the_handle
SES_STRING :: filename
SES_OPEN_TYPE :: open_flags

SES_LARGE_STRING, target :: actual_filename

! give the string memory and set the variables

actual_filename = "sesame"
open_flags = 'R'

filename=>actual_filename

! open the file

the_handle = ses_open_f90(filename, open_flags);
if (ses_is_valid_f90(the_handle) == SES_FALSE) then
        /* file not opened correctly – error */
endif
```

Note two things about this example: SES_STRING and SES_LABEL are implemented as F90 pointers, so they must be given memory and assigned values. In this example, memory is assigned by pointing the pointer at a target. SES_STRING's are pointed at a "SES_LARGE_STRING" target, and SES_LABEL's are pointed at a "SES_LABEL_STRING" target. Secondly, the name of the function called is not the same as the C version; all F90 functions contain in the SES_IO library have a "_f90" suffix.

6.2 Setup

Before it can be accessed, a sesame file handle is setup using the ses_setup_f90 function. Once setup, a ses_file_handle may be accessed. A file handle that is setup properly is associated with a sesame material and a sesame table.

Example:

```
 !  add in the ses_io module

use  ses_io

!  define f90 variables

 SES_ERROR_FLAG :: didit_setup
 SES_MATERIAL_ID :: the_mid
 SES_TABLE_ID :: the_tid
 SES_FILE_HANDLE :: the_handle

!  open the file   and get a handle

…..

!  set up the variables

the_mid = 2030
the_tid = 301

!  setup the valid file handle  for material id 2030 and standard table id 301

didit_setup = ses_setup_f90(the_handle,  the_mid,  the_tid);
if (didit_setup /= SES_NO_ERROR)  then
        /*  file not setup correctly – error */
endif
```

6.3 Access

A file may be opened for access in one of ways: for Reading (‘R’), Writing (‘W’), Append (‘A’) or Change (‘C’).

6.3.1 Read

A file may be read in one of two ways: using the *iterator* interface, or the *array* interface. Upon setup, the sesame library is pointed at the table associated with the given material and table. The iterator interface allows the user to iterate over all arrays in the data table associated with that table id.

The *array* interface assumes that the user knows the structure of the table and allows the user to directly access the data in the table.

6.3.1 1 Read with Iterator Example:

```
! reading a 301 table for material 2030 with the iterator interface

! add in the ses_io module

use ses_io

! define f90 variables

SES_FILE_HANDLE :: the_handle
SES_STRING :: filename
SES_OPEN_TYPE :: open_flags

SES_ERROR_FLAG :: didit_setup, didit_read
SES_MATERIAL_ID :: the_mid
SES_TABLE_ID :: the_tid
SES_NUMBER :: array_size
SES_WORD_REFERENCE :: pt_buffer

SES_LARGE_STRING, target :: actual_filename

! give the string memory and set the variables

actual_filename = “sesame”
open_flags = ‘R’

filename=>actual_filename

! open the file

the_handle = ses_open_f90(filename, open_flags)
if (ses_is_valid_f90(the_handle) == SES_FALSE)  then
        !  invalid file handle – error
endif

!  setup the variables

the_mid = 2030
the_tid = 301

didit_setup = ses_setup_f90(the_handle, the_mid, the_tid)
if (didit_setup != SES_NO_ERROR)  then
        !  did not set up properly – error
endif

array_size = 0
didit_read = SES_NO_ERROR
```

```
do while (ses_has_next_f90(the_handle) == SES_TRUE )
        array_size = ses_array_size_next_f90(the_handle)
        if (array_size <= 0)  then
                !  array size incorrect – error
        endif
        allocate(pt_buffer(array_size))
        didit_read = ses_read_next_f90(the_handle, pt_buffer, array_size)
        if (didit_read  /= SES_NO_ERROR) then
                !  error – did not read correctly
        endif

        !  here, the user may do what they like with the data
end do
```

6.3.1.2 Read with Array Interface Example:

```
use ses_io

!  define F90 variables

SES_FILE_HANDLE :: the_handle
SES_STRING :: filename
SES_ERROR_FLAG :: didit_setup,  didit_read

SES_NUMBER_REFERENCE :: pt_nr, pt_nt
SES_NUMBER, target :: nr, nt

SES_WORD_REFERENCE :: density
SES_WORD_REFERENCE :: temperature
SES_WORD_REFERENCE :: pressure
SES_WORD_REFERENCE :: energy
SES_WORD_REFERENCE :: free_energy

SES_LARGE_STRING, target :: actual_filename

SES_MATERIAL_ID :: the_mid
SES_TABLE_ID :: the_tid

!  create memory for the strings and numbers

actual_filename = “sesame”
filename => actual_filename

pt_nr => nr
```

```
pt_nt => nt

!  reading a 301 table for material 2030 with the array interface

the_handle = ses_open_f90(filename, 'R')
if (ses_is_valid_f90(the_handle) == SES_FALSE)  then
        !  invalid file handle – error
endif

the_mid = 2030
the_tid = 301

didit_setup = ses_setup_f90(the_handle, the_mid, the_tid)
if (didit_setup /= SES_NO_ERROR)  then
        !  did not set up properly – error
endif

!  data in the 301 table is ordered as follows:
!       nr
!       nt
!       density(1..nr)
!       temperature (1..nt)
!       pressure(1..nr)(1..nt)
!       energy(1..nr)(1..nt)
!       free energy(1..nr)(1..nt)

didit_read = SES_NO_ERROR;

didit_read = ses_read_number_f90(the_handle, pt_nr)
didit_read = ses_read_number_f90(the_handle, pt_ nt)

allocate (density(nr))
allocate (temperature(nt))

didit_read = ses_read_1D_f90(the_handle,  density, nr)
didit_read = ses_read_1D_f90(the_handle,  temperature, nt)

allocate(energy(nr*nt))
allocate(pressure(nr*nt))
allocate(free_energy(nr*nt))

didit_read = ses_read_2D_f90(the_handle, pressure, nr, nt)
didit_read = ses_read_2D_f90(the_handle, energy, nr, nt)
didit_read = ses_read_2D_f90(the_handle, free_energy, nr, nt)
```

6.3.2 Write

A file may be written in one of two ways: using the *iterator* interface, or the *array* interface. The file may not exist.

6.3.2.1 Write with Iterator Example

```
! writing a 301 table for material 2030 with the iterator interface

use ses_io

SES_FILE_HANDLE :: the_handle
SES_MATERIAL_ID :: the_mid
SES_TABLE_ID :: the_tid
SES_NUMBER :: nr, nt, ntab
SES_ERROR_FLAG :: didit_setup, didit_grid, didit_write
SES_NUMBER :: array_size
SES_LABEL :: the_label
SES_LABEL_STRING, target :: actual_label
SES_STRING :: filename
SES_OPEN_TYPE :: open_flags
SES_LARGE_STRING, target :: actual_filename
SES_WORD_REFERENCE :: data_buffer

the_label => actual_label
filename => actual_filename

actual_filename = “sesame_write”
open_flags = ‘W’

the_handle = ses_open_f90(filename, open_flags)
if (ses_is_valid_f90(the_handle) == SES_FALSE) then
        ! invalid file handle – error
endif

the_label => actual_label

the_mid = 2000
the_tid = 301
nr = 33
nt = 3
ntab = -1
didit_grid = ses_set_grid_f90(the_handle, nr, nt, ntab)
didit_setup = ses_setup_f90(the_handle, 2030, 301)
if (didit_setup != SES_NO_ERROR) then
        ! did not set up properly – error
endif

array_size = 0
didit_write = SES_NO_ERROR

do while (ses_has_next_f90(the_handle) == SES_TRUE)

        ! the user creates the data buffer and sets the array size
```

```
        !  the user puts data into the data_buffer here

        //  for each specific loop, change these
        the_label = "nr"
        data_buffer(1) = 3.285;    // etc

        didit_write = ses_write_next_f90(the_handle, data_buffer, array_size, the_label);
        if (didit_write  != SES_NO_ERROR) then
                /*  error – did not write correctly */
        endif
end do
```

6.3.2.2 Write with Array Interface Example

```
use ses_io

!  writing a 301 table for material 2030 with the array interface

the_handle = ses_open_f90(filename, 'W');
if (ses_is_valid(the_handle) == SES_FALSE) then
        !  invalid file handle – error
endif

nr = 33
nt = 3
ntab = -1
didit_grid = ses_set_grid_f90(the_handle, nr, ntab)

didit_setup = ses_setup_f90(the_handle, the_mid, the_tid)
if (didit_setup /= SES_NO_ERROR) then
        !  did not set up properly – error
endif

!  data in the 301 table is ordered as follows:
!       nr
!       nt
!       density(1..nr)
!       temperature (1..nt)
!       pressure(1..nr)(1..nt)
!       energy(1..nr)(1..nt)
!       free energy(1..nr)(1..nt)

didit_write = SES_NO_ERROR;

didit_write = ses_write_number_f90(the_handle, nr)
didit_write = ses_write_number_f90(the_handle, nt)
```

```
allocate(density(nr))
allocate(temperature(nt))

!  fill in the data to be written here
density(1) = .409  !  etc
temperature(1) = .2342  ! etc

didit_write = ses_write_1D_f90(the_handle, density, nr);
didit_write = ses_write_1D_f90(the_handle, temperature, nt);

allocate (pressure(nr*nt))
allocate (energy(nr*nt))
allocate (free_energy(nr*nt))

!  put some more values in the data to be written here

didit_write = ses_write_2D_f90(the_handle, pressure, nr, nt)
didit_write = ses_write_2D_f90(the_handle, energy, nr, nt)
didit_write = ses_write_2D_f90(the_handle, free_energy, nr, nt)
```

6.3.3 Append

A new material library may be appended to a file. The given file must exist, and must not contain the material specified in the ses_setup_f90 function.

If appending a new material to a Sesame ASCII file, the file must have a proper ASCII file format. If it does not, the new material might not be appended correctly. A Sesame ASCII file's last line must have the end of file record (" 2" at columns 1& 2, spaces until column 80 which contains a 2). The last 2 being the last character in the file.

Example:

```
! appending a 301 table for new material 2000 to an existing file

the_handle = ses_open_f90("sesame_append", 'A')

if (ses_is_valid_f90(the_handle) == SES_FALSE) then
        ! invalid file handle – error
endif

the_mid = 2000
the_tid = 301
nr = 3
nt = 3
ntab = -1
didit_grid = ses_set_grid_f90(the_handle, nr, nt, ntab);
didit_setup = ses_setup_f90(the_handle, the_mid, the_tid)
if (didit_setup /= SES_NO_ERROR) then
        ! did not set up properly – error
endif

array_size = 0
didit_write = SES_NO_ERROR;

do while (ses_has_next_f90(the_handle))
        array_size = ses_array_size_next_f90(the_handle)
        if (array_size <= 0) then
                ! array size incorrect – error
        endif

        ! the user puts data into the data_buffer here *
        allocate (data_buffer(array_size)
        data_buffer(1) = 3.285    ! etc
        actual_label = "pressure"
        didit_write = ses_write_next_f90(the_handle, data_buffer, array_size);

        if (didit_write /= SES_NO_ERROR) then
                ! error – did not write correctly
        endif
end do
```

6.3.4 Change

Data in an array in a sesame table may be changed. The given file must exist, and must contain the material/table pair specified in the setup function.

Example:

```
 !  changing data in the energy array in  a 301 table for material 2030

the_handle = ses_open_f90("sesame", 'C')

if (ses_is_valid_f90(the_handle) == SES_FALSE) then
        !  invalid file handle – error
endif

the_mid = 2030
the_tid = 301
didit_setup = ses_setup_f90(the_handle,  the_mid, the_tid)
if (didit_setup /= SES_NO_ERROR) then
        !  did not set up properly – error
endif

array_size = 0
didit_skip = SES_NO_ERROR
didit_write = SES_NO_ERROR

do while (ses_has_next_f90(the_handle) == SES_TRUE)
        the_label = ses_get_label_f90(the_handle)
        array_size = ses_array_size_next_f90(the_handle)
        if (the_label ==  "e – energy (MJ/kg)") then
                !  found the energy array
                !  put data into the buffer here, then write it to the file

                allocate(data_buffer(array_size))

                data_buffer(1) = 343.344    !etc
                didit_write = ses_change_next_f90(the_handle, data_buffer, array_size);

        else
                didit_skip = ses_skip_f90(the_handle);
        endif
end do
```

6.3.5 Setup Options

6.3.5.1 Array Order

SES_IO uses as default multi-dimension array order of column major, as in Fortran 90. It also reads and writes 2D and 3D arrays using an array reference, so the array order is important. The C language uses a row major order for multiple dimension arrays. The order of the elements, when reading from a sesame file in C, must be converted from column major order (on the file) to row major order (used in C). Likewise, when writing a sesame file in C the array must be converted from row major order (used in C) to column major order (on the file).

To specify that the file will be read into arrays in row major order, the user can set the array order with the ses_set_array_order_f90 function.

Example:

```
! set up to read multi-dimensional arrays into row-major order arrays

didit_setup = ses_setup_f90(the_handle, 2030, 301)
if (didit_setup /= SES_NO_ERROR) then
        ! did not set up properly – error
endif


didit_set_order= ses_set_array_order_f90(the_handle, 'R')

if (didit_set_order /= SES_NO_ERROR) then
        ! did not set the array order properly – error
endif
```

To specify that the file will be written into arrays in column major order, the user sets the array order to column-major order.

Example:

```
! set up to write row-major order multi-dimensional arrays into column-major order
!    arrays on the file

the_mid = 2030
the_tid = 301
didit_setup = ses_setup_f90(the_handle, the_mid, the_tid)
if (didit_setup /= SES_NO_ERROR) then
        !  did not set up properly – error
endif


 didit_set_order= ses_set_array_order_f90(the_handle, 'C');

if (didit_set_order /= SES_NO_ERROR) then
        !  did not set the array order properly – error
}
```

6.3.5.2 Significant digits

SES_IO contains the option to truncate data read in from or written to a sesame file to a certain number of digits followed by 0's in the remaining decimal places. This can help the user community avoid floating point arithmetic precision errors, such as in the case when a computation produces a floating-point number like 5.99999999e-12. Computations often provide the closest floating-point number to a computed number because not all real numbers are representable in a fixed number of bits. Note that this is not a true significant digit read or write, where only the significant digits would be read or written.

To set the numbers read-in and written to a file to a constant number of significant digits, the ses_set_significant_digits_f90 function is used.

Example:

```
! set up to read or write 3 significant digits to a file

the_mid = 2030
the_tid = 301
 didit_setup = ses_setup_f90(the_handle, the_mid, the_tid)
 if (didit_setup != SES_NO_ERROR) then
        ! did not set up properly – error
endif

sig = 3

 didit_set_digits= ses_set_significant_digits_f90(the_handle, sig);

if (didit_set_digits != SES_NO_ERROR) then
        ! did not set significant digits properly – error
endif
```

6.3.5.3 Validation

SES_IO contains the option to inform the user when data that is not valid is read from or written to a file.

To set this option, the ses_set_validate_f90 function is used.

Example:

```
! set up to read or write with validation

the_mid = 2030
the_tid = 301
didit_setup = ses_setup_f90(the_handle, the_mid, the_tid)
if (didit_setup /= SES_NO_ERROR) then
        ! did not set up properly – error
endif

 didit_set_valid= ses_set_valide_f90(the_handle)
if (didit_set_valid /= SES_NO_ERROR) then
        ! did not set validation properly – error
endif
```

6.3.5.4 Material order

The user may specify that when a file is written, it will be written with the material ids in ascending order. To set this option, the ses_set_material_order_f90 function is used.

Example:

```
! set up to write materials in order

didit_set_materials = ses_set_material_order_f90 (the_handle)
if (didit_set_materials /= SES_NO_ERROR) then
        ! did not set validation properly – error
endif
```

6.4 Close

When finished, a sesame file handle is closed with the ses_close_f90 function.

Example:

```
! close the valid file handle

didit_close = ses_close_f90(the_handle)
if (didit_close != SES_NO_ERROR) then
        ! file not closed correctly – error
endif
```

6.5 Querying Sesame Files

The SES_IO library provides several functions useful for querying information about a sesame file.

6.5.1 Example for ses_date_f90

The “ses_date_f90” function returns a ses_string with the date information for the file.

Example:

```
! query the file date

the_date = ses_date_f90(the_handle)
if (LEN(the_date) == 0) then
        ! function error – string not returned correctly
endif
```

6.5.2 Example for ses_version_f90

The "ses_version_f90" function returns a ses_string with the version information for the file.

Example:

```
! query the file version

ses_string the_version = ses_version_f90(the_handle)
if (LEN(the_version )== 0) then
        ! function error – string not returned correctly
endif
```

6.5.3 Example for ses_get_materials_f90

The "ses_get_materials_f90" returns a ses_material_id_reference, a reference to an array of ses_material_ids. This is a list of the materials on the file.

Example:

```
! return a list of the materials on the file

the_materials => ses_get_materials_f90(the_handle, pt_size);
if (SIZE(the_materials) == 0) then
        ! function error – list not returned correctly
endif
if (actual_size == 0) then
        ! function error – size not returned correctly
endif

! print the list
i = 0
do i=1, actual_size(1)
        print *, "material [", i, "] is " the_materials(i)
enddo
```

6.5.4 Example for ses_get_table_ids_f90

The "ses_get_table_ids_f90" returns a ses_table_id_reference, a reference to an array of ses_table_ids. This is a list of the materials on the file.

Example:

```
!  return a list of tables associated with a material on the file

actual_size = 0
the_mid = 2030
the_tables => ses_get_table_ids_f90(the_handle,  the_mid, pt_size)
if (SIZE(the_tables) == 0) then
        !  function error – list not returned correctly
endif
if (actual_size(1) == 0) then
!  function error – size not returned correctly
endif

! print the list

do i=1, size
        print *, “table  [“,i,”] for material 2030 is “, the_tables(i)
enddo
```

6.5.5 Example for ses_comments_f90

The “ses_comments” function returns a ses_string* with any comments (from the sesame 101 table) on the file.

Example:

```
!  after setup to 101 table

the_comments => actual_comments

! retrieve comments

didit_get_comments = ses_comments_f90(the_handle, the_comments)
if (didit_get_comments /= SES_NO_ERROR) then
        !  error – comments not retrieved
endif
```

6.6 Combining Sesame Files

The following function is used to combine two sesame files:

ses_error_flag ses_combine_f90(ses_file_handle file1, ses_file_handle file2, ses_string new_filename)

7.0 Java Examples

The Java interface to the SES_IO library is a java object. This object must be created and used in code that accesses the SES_IO library. The defines file "ses_defines_f90.h" is used with every file containing code that accesses ses_io.

7.1 Opening a File

A sesame file is opened with the SesOpen function. A ses_file_handle is returned.

Example:

```
import ses_defines.*;

// instantiate the SesIO library object

SesIO my_ses_io = new SesIO();

// open the file

ses_file_handle  the_handle = new ses_file_handle();
ses_string filename = new ses_string("sesame");
ses_open_type open_flags = new_ses_open_type('R');

the_handle.the_value = my_ses_io.SesOpen(filename.the_value, open_flags.the_value);
if ((my_ses_io.SesIsValid(the_handle.the_value) ==ses_boolean. SES_FALSE) {
        //  file not opened correctly – error
}
```

Note two things about this example: All parameters are passed with the ".the_value" qualifier. This accesses the data from the Java wrapped classes, and is the standard usage of all java functions in this library. Secondly, since everything in Java is an object, and every object in Java is a pointer, all sesame data types must be instantiated (given memory with 'new') before use.

The import ses_defines.* must be included at the top of every file that uses SES_IO.

7.2 Setup

Before it can be accessed, a sesame file handle is setup using the SesSetup function. Once setup, a ses_file_handle may be accessed. A file handle that is setup properly is associated with a sesame material and a sesame table.

Example:

```
//  setup the valid file handle  for material id 2030 and standard table id 301

ses_error_flag  didit_setup = new ses_error_flag();
ses_material_id the_mid = new ses_material_id(2030);
ses_table_id the_tid = new ses_table_id(301);

didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value,
                                           the_mid.the_value, the_tid.the_value);
if (didit_setup.the_value != ses_error_flag.SES_NO_ERROR) {
        //  file not setup correctly – error
}
```

Notice the error flag usage; all error codes are public variables of the ses_error_flag class, so they must be used with the "ses_error_flag.ERROR_CODE" usage used above.

7.3 Access

A file may be opened for access in one of ways: for Reading ('R'), Writing ('W'), Append ('A') or Change ('C').

7.3.1 Read

A file may be read in one of two ways: using the *iterator* interface, or the *array* interface. Upon setup, the sesame library is pointed at the table associated with the given material and table. The iterator interface allows the user to iterate over all arrays in the data table associated with that table id.

The *array* interface assumes that the user knows the structure of the table and allows the user to directly access the data in the table.

7.3.1.1 Read with Iterator example:

```
        //  reading a 301 table for material 2030 with the iterator interface

        ses_file_file the_handle = new ses_file_handle();
        ses_string filename = new ses_string("sesame");
        ses_open_type open_flags = new  ses_open_type('R');
        the_handle.the_value = my_ses_io.SesOpen(filename.the_value, open_flags.the_value);
        if (my_ses_io.SesIsValid(the_handle.the_value)).the_value ==  ses_boolean.SES_FALSE) {
                //  invalid file handle – error
        }

        ses_material_id the_mid = new ses_material_id(2030);
        ses_table_id the_tid = new ses_table_id(301);
        ses_error_flag didit_setup =new ses_error_flag();
        didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value,
                 the_mid.the_value, the_tid.the_value);
        if (didit_setup.the_value  != ses_error_flag.SES_NO_ERROR) {
                //  did not set up properly – error
        }

        ses_number array_size = new ses_number(0);

        while (my_ses_io.SesHasNext(the_handle.the_value)) {
                array_size.the_value = my_ses_io.SesArraySizeNext(the_handle.the_value);
                if (array_size.the_value <= 0) {
                        //  array size incorrect – error
                }

                ses_word_reference data_buffer = new ses_word_reference(array_size.the_value);
                data_buffer.the_value = my_ses_io.SesReadNext(the_handle.the_value,
                                                                (int) array_size.the_value);
                if (data_ buffer.the_value.length  <= 0) {
                        //  error – did not read correctly
                }

                //  here, the user may do what they like with the data
}
```

7.3.1.2 Read with Array Interface example:

```
// reading a 301 table for material 2030 with the array interface

ses_file_file the_handle = new ses_file_handle();
ses_string filename = new ses_string("sesame");
ses_open_type open_flags = new ses_open_type('R');
the_handle.the_value = my_ses_io.SesOpen(filename.the_value, open_types.the_value);
if (my_ses_io.SesIsValid(the_handle.the_value) == ses_boolean.SES_FALSE) {
        // invalid file handle – error
}

ses_error_flag didit_setup = new ses_error_flag();
ses_material_id the_mid = new ses_material_id(2030);
ses_table_id the_tid = new ses_table_id(301);
didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value,
                                           the_mid.the_value, the_tid.the_value);
if (didit_setup.the_value  != ses_error_flag. SES_NO_ERROR) {
        // did not set up properly – error
}

// data in the 301 table is ordered as follows:
//        nr
//        nt
//        density(1..nr)
//        temperature (1..nt)
//        pressure(1..nr)(1..nt)
//        energy(1..nr)(1..nt)
//        free energy(1..nr)(1..nt)

ses_error_flag didit_read = new ses_error_flag();
ses_number nr = new ses_number(0);
ses_number nt = new ses_number(0);

nr.the_value = my_ses_io.SesReadNumber(the_handle.the_value);
nt.the_value  = my_ses_io.SesReadNumber(the_handle.the_value);

ses_word _reference density = new ses_word_reference(nr.the_value);
ses_word _reference temperature = new ses_word_reference(nt.the_value);

density.the_value = my_ses_io.SesRead1D(the_handle.the_value, (int) nr.the_value);
temperature.the_value = my_ses_io.SesRead1D(the_handle.the_value,  (int)nt.the_value);

ses_word_reference  pressure = new ses_word_reference(nr.the_value*nt.the_value);
ses_word_reference  energy  = new ses_word_reference(nr.the_value*nt.the_value);
ses_word_reference free_energy  = new ses_word_reference(nr.the_value*nt.the_value);

pressure.the_value = my_ses_io.SesRead2D(the_handle, (int)nr.the_value, (int)nt.the_value);
energy.the_value = my_ses_io.SesRead2D(the_handle, (int)nr.the_value, (int)nt.the_value);
free_energy.the_value = my_ses_io.SesRead2D(the_handle,  (int)nr.the_value, (int)nt.the_value);
```

7.3.2 Write

A file may be written in one of two ways: using the *iterator* interface, or the *array* interface. The file may not exist.

7.3.2.1 Write with Iterator Example:

```
// writing a 301 table for material 2030 with the iterator interface

ses_file_handle the_handle = new ses_file_handle();
ses_string filename = new ses_string("sesame_write");
ses_open_type open_flags = new ses_open_type('W');
the_handle.the_value = my_ses_io.SesOpen(filename.the_value, open_flags.the_value);
if (my_ses_io.SesIsValid(the_handle.the_value) == ses_boolean.SES_FALSE) {
        // invalid file handle – error
}

ses_number nr = new ses_number(33);
ses_number nt = new ses_number(3);
ses_number ntab = new ses_number(-1);
ses_error_flag didit_grid = new ses_error_flag();
didit_grid.the_value = my_ses_io.SesSetGrid(the_handle.the_value, (int) nr.the_value,
                        (int)nt.the_value, (int)ntab.the_value);
ses_error_flag didit_setup = new ses_error_flag();

ses_material_id the_mid = new ses_material_id(2030);
ses_table_id the_tid = new ses_table_id(301);
didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value, the_mid.the_value,
                                        the_tid.the_value);
if (didit_setup.the_value !=ses_error_flag. SES_NO_ERROR) {
        // did not set up properly – error
}

ses_number array_size = new ses_number(0);
ses_error_flag didit_write = new ses_error_flag();
didit_write.the_value = ses_error_flag.SES_NO_ERROR;
ses_label the_label = new ses_label();

while (my_ses_io.SesHasNext(the_handle.the_value)) {

        // the user creates the data buffer and sets the array size

        // the user puts data into the data_buffer here

        ses_label the_label = new ses_label("nr");
        ses_word_reference data_buffer.the_value  = new ses_word_reference(array_size)
        data_buffer.the_value[0] = 3.285;   // etc

        didit_write.the_value = my_ses_io.SesWriteNext(the_handle.the_value,
```

```
                    data_buffer.the_value, (int) array_size.the_value, the_label.the_value);
            if (didit_write.the_value  != ses_error_flag.SES_NO_ERROR) {
                    //  error – did not write correctly
            }
}
```

7.3.2.2 Write with Array Interface Example:

```
//  writing a 301 table for material 2030 with the array interface

ses_file_handle the_handle = new ses_file_handle();
ses_string filename = new ses_string("sesame_write_array");
ses_open_type open_flags = new ses_open_type('W');
the_handle.the_value = my_ses_io.SesOpen(filename.the_value,  open_flags.the_value);
if (my_ses_io.SesIsValid(the_handle) == ses_boolean.SES_FALSE) {
        //  invalid file handle – error
}

ses_number nr = new ses_number(33);
ses_number nt = new ses_number(3);
ses_number ntab = new ses_number(-1);
ses_error_flag didit_grid = new ses_error_flag();
didit_grid.the_value = my_ses_io.SesSetGrid(the_handle.the_value, (int)nr.the_value,
         (int)nt.the_value, (int)ntab.the_value);

ses_material_id the_mid = new ses_material_id(2030);
ses_table_id the_tid = new ses_table_id(301);
ses_error_flag didit_setup = new ses_error_flag();
didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value,
        the_mid.the_value, the_tid.the_value);
if (didit_setup.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set up properly – error
}

//  data in the 301 table is ordered as follows:
//          nr
//          nt
//          density(1..nr)
//          temperature (1..nt)
//          pressure(1..nr)(1..nt)
//          energy(1..nr)(1..nt)
//          free energy(1..nr)(1..nt)

ses_error_flag didit_write = new ses_error_flag();
didit_write.the_value = ses_error_flag.SES_NO_ERROR;

didit_write.the_value = my_ses_io.SesWriteNumber(the_handle.the_value, (int)nr.the_value);
didit_write.the_value = my_ses_io.SesWriteNumber(the_handle.the_value, (int)nt.the_value);
```

```
ses_word_reference  density = new ses_word_reference((int)nr.the_value);
ses_word_reference temperature = new ses_word_reference((int)nt.the_value);

//  fill in the data to be written here
density.the_value[0] = .409;  // etc
temperature.the_value[0] = .2342;  /*etc*/

didit_write.the_value = my_ses_io.SesWrite1D(the_handle.the_value,
                                                density.the_value, (int)nr.the_value);
didit_write.the_value = my_ses_io.SesWrite1D(the_handle.the_value,
                                                temperature.the_value, (int)nt.the_value);

ses_word_reference  pressure = new ses_word_reference((int)nr.the_value*(int)nt.the_value);
ses_word_reference energy  = new ses_word_reference((int)nr.the_value*(int)nt.the_value);
ses_word_reference  free_energy  = new ses_word_reference((int)nr.the_value*(int)nt.the_value);

//  put some more values in the data to be written here

didit_write.the_value = my_ses_io.SesWrite2D(the_handle.the_value,
                                        pressure, (int)nr.the_value, (int)nt.the_value);
didit_write.the_value = my_ses_io.SesWrite2D(the_handle.the_value,
                                        energy, (int)nr.the_value, (int)nt.the_value);
didit_write.the_value = my_ses_io.SesWrite2D(the_handle.the_value,
                                        free_energy, (int)nr.the_value, (int)nt.the_value);
```

7.3.3 Append

A new material library may be appended to a file. The given file must exist, and may not contain the material specified in the SesSetup function.

Example:

```
//  appending  a 301 table for new material 2030 to an existing file

ses_file_file the_handle.the_value = SesOpen("sesame", 'A');
if (my_ses_io.SesIsValid(the_handle.the_value) == ses_boolean.SES_FALSE) {
        //  invalid file handle – error
}

ses_error_flag didit_grid = new ses_error_flag();
didit_grid = SesSetGrid(the_handle.the_value, 33, 3, -1);

ses_error_flag didit_setup = new ses_error_flag();
didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value, 2000, 301);
if (didit_setup.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set up properly – error
}

ses_number array_size = new ses_number(0);
```

```
ses_error_flag didit_write = new ses_error_flag();
didit_write. the_value = ses_error_flag.SES_NO_ERROR;
ses_label the_label = new ses_label();

while (my_ses_io.SesHasNext(the_handle.the_value) == ses_boolean.SES_TRUE) {
        array_size.the_value = my_ses_io.SesArraySizeNext(the_handle.the_value);
        if (array_size.the_value <= 0) {
                //  array size incorrect – error
        }

        //  the user puts data into the data_buffer here
        ses_word_reference data_buffer = new ses_word_reference(array_size.the_value);
        data_buffer.the_value[0] = 3.285;    //  etc

        didit_write.the_value = my_ses_io.SesWriteNext(the_handle.the_value,
                                        data_buffer.the_value, array_size.the_value);
        if (didit_write.the_value  != ses_error_flag.SES_NO_ERROR) {
                //  error – did not write correctly
        }
}
```

7.3.4 Change

Data in an array in a sesame table may be changed. The given file must exist, and must contain the material/table pair specified in the setup function.

Example:

```
// changing data in the energy array in a 301 table for material 2030

ses_file_file the_handle = new ses_file_handle();
the_handle.the_value = my_ses_io.SesOpen("sesame_change", 'C');
if (my_ses_io.SesIsValid(the_handle.the_value) == ses_boolean.SES_FALSE) {
        // invalid file handle – error
}

ses_error_flag didit_setup = new ses_error_flag();
didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value, 2030, 301);
if (didit_setup.the_value !=ses_error_flag. SES_NO_ERROR) {
        // did not set up properly – error
}

ses_number array_size = new ses_number(0);
ses_error_flag didit_skip = new ses_error_flag();
didit_skip.the_value = SES_NO_ERROR;
ses_label the_label = new ses_label();
ses_error_flag didit_write = new ses_error_flag();
didit_write.the_value = SES_NO_ERROR;

while (my_ses_io.SesHasNext(the_handle.the_value) == ses_boolean.SES_TRUE) {
        the_label.the_value = my_ses_io.SesGetLabel(the_handle.the_value);
        array_size.the_value = my_ses_io.SesArraySizeNext(the_handle.the_value);
        if (the_label.the_value.equals( "energy") == true) {
                // found the energy array
                // put data into the buffer here, then write it to the file */

        ses_word_reference data_buffer = new ses_word_reference(array_size.the_value);

                data_buffer.the_value[0] = 343.344; // etc
                didit_write.the_value = my_ses_io.SesChangeNext(the_handle.the_value,
                                data_buffer.the_value, array_size.the_value);
        }
        else {
                didit_skip.the_value = my_ses_io.SesSkip(the_handle.the_value);
        }
}
```

7.3.5 Setup Options

7.3.5.1 Array Order

SES_IO uses as default multi-dimension array order of column major, as in Fortran 90. It also reads and writes 2D and 3D arrays using an array reference, so the array order is important. The C language uses a row major order for multiple dimension arrays. The order of the elements, when reading from a sesame file in C, must be converted from column major order (on the file) to row major order (used in C). Likewise, when writing a sesame file in C the array must be converted from row major order (used in C) to column major order (on the file).

To specify that the file will be read into arrays in row major order, the user can set the array order with the SesSetArrayOrder function.

Example:

```
//  set up to read multi-dimensional arrays into row-major order arrays

ses_error_flag didit_setup = my_ses_io.SesSetup(the_handle.the_value, 2030, 301);
if (didit_setup.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set up properly – error
}


ses_error_flag didit_set_order = new ses_error_flag();

didit_set_order.the_value = my_ses_io.SesSetArrayOrder(the_handle.the_value, 'R');

if (didit_set_order.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set the array order properly – error
}
```

To specify that the file will be written into arrays in column major order, the user sets the array order to column-major order.

Example:

```
// set up to write row-major order multi-dimensional arrays into column-major order
    arrays on the file

ses_error_flag didit_setup = new ses_error_flag();
didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value, 2030, 301);
if (didit_setup.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set up properly – error
}


ses_error_flag didit_set_order = new ses_error_flag();

didit_set_order.the_value = my_ses_io.SesSetArrayOrder(the_handle.the_value, 'C');

if (didit_set_order.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set the array order properly – error
}
```

7.3.5.2 Significant digits

SES_IO contains the option to truncate data read in from or written to a sesame file to a certain number of digits followed by 0's in the remaining decimal places. This can help the user community avoid floating point arithmetic precision errors, such as in the case when a computation produces a floating-point number like 5.99999999e-12. Computations often provide the closest floating-point number to a computed number because not all real numbers are representable in a fixed number of bits. Note that this is not a true significant digit read or write, where only the significant digits would be read or written.

To set the numbers read-in and written to a file to a constant number of significant digits, the SesSetSignificantDigits function is used.

Example:

```
//  set up to read or write 3 significant digits to a  file

ses_error_flag didit_setup = new ses_error_flag();
didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value, 2030, 301);
if (didit_setup.the_value !=ses_error_flag.SES_NO_ERROR) {
        //  did not set up properly – error
}


ses_error_flag didit_set_digits = new ses_error_flag();

didit_set_digits.the_value = my_ses_io.SesSetSignificantDigits(the_handle.the_value, 3);

if (didit_set_digits.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set significant digits properly – error
}
```

7.3.5.3 Validation

SES_IO contains the option to inform the user when data that is not valid is read from or written to a file.

To set this option, the SesSetValidate function is used.

Example:

```
//  set up to read or write with validation

ses_error_flag didit_setup = new ses_error_flag();
didit_setup.the_value = my_ses_io.SesSetup(the_handle.the_value, 2030, 301);
if (didit_setup.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set up properly – error
}

ses_error_flag didit_set_valid = new ses_error_flag();
didit_set_valid.the_value = my_ses_io.SesSetValidate(the_handle.the_value);
if (didit_set_valid.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set validation properly – error
}
```

7.3.5.4 Material order

The user may specify that when a file is written, it will be written with the material ids in ascending order. To set this option, the SesSetMaterialOrder function is used.

Example:

```
//  set up to write materials in order, then call SesSetMaterialOrder

ses_error_flag didit_set_materials  = new ses_error_flag();
didit_set_materials.the_value = my_ses_io.SesSetMaterialOrder (the_handle.the_value);
if (didit_set_materials.the_value != ses_error_flag.SES_NO_ERROR) {
        //  did not set validation properly – error
}
```

7.4 Close

When finished, a sesame file handle is closed with the SesClose function.

Example:

```
// close the valid file handle

ses_error_flag didit_close = new ses_error_flag();
didit_close.the_value = my_ses_io.SesClose(the_handle.the_value);
if (didit_close.the_value != ses_error_flag.SES_NO_ERROR) {
        // file not closed correctly – error
}
```

7.5 Querying Sesame Files

The SES_IO library provides several functions useful for querying information about a sesame file.

7.5.1 Example for SesDate

The “SesDate” function returns a ses_string with the date information for the file.

Example:

```
// query the file date

ses_string the_date = new ses_string();
the_date.the_value  = my_ses_io.SesDate(the_handle.the_value);
if (the_date.the_value == (ses_string)NULL) {
        // function error – string not returned correctly
}
```

7.5.2 Example for SesVersion

The “SesVersion” function returns a ses_string with the version information for the file.

Example:

```
// query the file version

ses_string the_version = new ses_string();
the_version.the_value = my_ses_io.SesVersion(the_handle.the_value);
if (the_version.the_value == (ses_string)NULL) {
        // function error – string not returned correctly
}
```

7.5.3 Example for SesGetMaterials

The "SesGetMaterials" returns a ses_material_id_reference, a reference to an array of ses_material_ids. This is a list of the materials on the file.

Example:

```
//  return a list of the materials on the file

ses_material_id_reference the_materials = new ses_material_id_reference();
the_materials.the_value = my_ses_io.SesGetMaterials(the_handle.the_value);
if (the_materials.the_value == (ses_material_id_reference)NULL) {
        //  function error – list not returned correctly */
}
if (the_materials.the_value.length == 0) {
        //  function error – size not returned correctly */
}

//  print the list

for  (int i=0; i < the_materials.the_value.length; i++) {
        System.out.println("material ["+ i +"] is " + the_materials.the_value[i]);
}
```

7.5.4 Example for SesGetTableIds

The "SesGetTableIds" returns a ses_table_id_reference, a reference to an array of ses_table_ids. This is a list of the materials on the file.

Example:

```
//  return a list of tables associates with a material on the file – must be setup first

ses_table_id_reference the_tables = new ses_table_id_reference();
the_tables.the_value  = my_ses_io.SesGetTableIds(the_handle.the_value, 2030, size);
if (the_tables.the_value == (ses_table_id_reference)NULL) {
        /*  function error – list not returned correctly */
}
if (the_tables.the_value.size == 0) {
   //  function error – size not returned correctly
}

//  print the list

for  (int i=0; i < the_tables.the_value.length; i++) {
        System.out.println("table  [" + i + "] for material 2030 is " + the_tables[i]);
}
```

7.5.5 Example for SesComments

The "SesComments" function returns a ses_string with any comments (from the sesame 101 table) on the file.

Example:

```
// retrieve comments

ses_string the_comments = new ses_string("");
ses_string null_string = new ses_string("");

the_comments.the_value = my_ses_io.SesComments(the_handle.the_value);
if (the_comments.the_value.equals(null_string)) {
        //  error – comments not retrieved
}
```

7.6 Combining Sesame Files

The following function is used to combine two sesame files:

ses_error_flag SesCombine(ses_file_handle file1, ses_file_handle file2, ses_string new_filename);

8.0 Reference

8.1 Data Types

ses_file_handle

A ses_file_handle is used to access sesame files. All file access is done through the ses_file_handle. The function ses_is_valid returns a ses_boolean that indicates whether the file handle is valid.

ses_open_type

A ses_open type is one of four characters: 'R' for read-only, 'W' for write-only, 'C' for change an existing file, and 'A' for append to an existing file.

ses_string

A ses_string is a typical string, an array of characters.

ses_error_flag

A ses_error_flag indicates an error.

ses_material_id

A ses_material_id is a valid sesame material id.

ses_table_id

A ses_table_id is a valid sesame table id.

ses_array_order

A ses_array_order is a character that indicates array order: 'C' for column-major, 'R' for row-major.

ses_boolean

A ses_boolean has one of two values: SES_TRUE or SES_FALSE.

ses_label

A ses_label is a ses_string of generally small length.

ses_word

A ses_word is a floating-point word.

ses_word_reference

A ses_word_reference is a reference to a ses_word array.

ses_ascii_word_type

A ses_ascii_word_type is either ‘M’ for medium or ‘S’ for small. Default is ‘M’; it is the length (number of chars) of the word on the ASCII file.

ses_number

A ses_number is a positive integer > 0.

ses_number_reference

A ses_number_reference is a reference to a ses_number array.

ses_material_id_reference

A ses_material_id_reference is a reference to a ses_material_id array.

ses_table_id_reference

A ses_table_id reference is a reference to a ses_table_id array.

8.2 Function Reference

8.2.1 C Functions

ses_access_directory

```
long ses_access_directory(ses_file_handle the_handle,
                          ses_material_id_reference* return_matid,
                          long**nwds,
                          long** iadr,
                          long* date,
                          long* version);
```

This function accesses a sesame directory record directly and returns the information in that directory. See Appendix A for the information in a sesame directory record.

ses_access_table_index

```
long ses_access_table_index(ses_file_handle the_handle,
                            ses_table_id_reference* return_tblid,
                            long** nwds,
                            long** iadr,
                            long* date1,
                            long* date2,
                            long* version);
```

This function accesses a sesame table index record directly and returns the information in that index record. See Appendix A for the information in a sesame table index record.

ses_array_size_next

```
ses_number  ses_array_size_next(ses_file_handle the_handle);
```

This function is used with the read iterator interface. It returns the size of the next array at the current handle.

ses_change_next

```
ses_error_flag ses_change_next(ses_file_handle the_handle,
                               ses_word_reference the_buffer,
                               ses_number dim);
```

This function is used with the write iterator interface. It allows the user to change the next array to the values in “the_buffer”.

ses_close

```
ses_error_flag  ses_close(ses_file_handle the_handle);
```

This function closes the current file handle.

ses_combine

```
ses_error_flag  ses_combine(ses_file_handle file1,
                            ses_file_handle file2,
                            ses_string new_filename);
```

This function combines two sesame files into the new file “new_filename”.

ses_comments

```
ses_error_flag   ses_comments(ses_file_handle the_handle,
                              ses_string* the_string);
```

This function returns comments from the given file handle.

ses_date

```
ses_string  ses_date(ses_file_handle the_handle);
```

This function returns the date from the sesame file directory.

ses_define_table

```
ses_error_flag ses_define_table(ses_table_id tid,
                                ses_label description,
                                long nr,
                                long nt,
                                long num_arrays,
                                char** size_arrays,
                                ses_label* labels);
```

This function creates a user-defined table.

ses_delete_table

```
ses_error_flag   ses_delete_table(ses_file_handle the_handle);
```

This function deletes the table that has been “setup” (a material, table pair).

ses_exit

```
ses_boolean ses_exit( );
```

This routine cleans up and deletes all global memory. Use with care.

ses_format

```
ses_string ses_format(ses_file_handle the_handle);
```

This routine gets the format of the data on the current ses_file_handle.

ses_get_comments

```
ses_error_flag ses_get_comments(ses_file_handle the_handle,
                                ses_string* the_comments);
```

This function returns all the comments on the given ses_file_handle.

ses_get_date

```
long ses_get_date(ses_file_handle the_handle);
```

This function returns the date from the sesame file directory as a long integer.

ses_get_grid

```
ses_boolean  ses_get_grid(    ses_file_handle the_handle,
                              ses_material_id the_mid,
                              ses_table_id  the_tid,
                              long* nr,
                              long* nt,
                              long* ntab);
```

This function returns the grid (nr and nt and ntab) for the material, table pair on the given file handle.

ses_get_label

```
ses_label     ses_get_label(ses_file_handle the_handle);
```

This function is used with the iterator interface. It returns the label of the next array on the file handle.

ses_get_material_id

```
ses_material_id  ses_get_material_id(ses_string material_name);
```

This function returns the material id associated with the given material name.

ses_get_materials

```
ses_material_id_reference ses_get_materials(ses_file_handle the_handle,
                                            long* size);
```

This function returns an array of the material id's on the sesame file.

ses_get_table_ids

```
ses_table_id_reference   ses_get_table_ids(ses_file_handle the_handle,
                                           ses_material_id mid,
                                           long* size);
```

This function returns a list of the table ids on the file associated with a given material id.

ses_get_table_sizes

```
ses_table_sizes_reference  ses_get_table_sizes(ses_file_handle the_handle,
                                               ses_material_id mid,
                                               long* size);
```

This routine returns the table sizes for the given material.

ses_get_version

```
long ses_get_version(ses_file_handle the_handle);
```

This routine returns the version from the ses_directory as a long integer.

ses_has_next

```
ses_boolean ses_has_next(ses_file_handle the_handle);
```

This function is used in the iterator interface, and returns true if there are more arrays in the table.

ses_indicates_error

```
ses_boolean     ses_indicates_error(ses_error_flag the_ses_error_flag);
```

This flag returns true if the given error flag indicates an error has occurred.

ses_is_valid

```
ses_boolean     ses_is_valid(ses_file_handle the_handle);
```

This function returns true if the given file handle is connected correctly to a valid sesame file.

ses_open

```
ses_file_handle ses_open( ses_string filename,
                          ses_open_type open_flags);
```

This function opens the sesame file and associates it with a file handle.

ses_print_error_condition

```
ses_string      ses_print_error_condition(ses_file_handle the_handle);
```

This function prints the last error condition on the given file handle.

ses_print_error_message

```
ses_string ses_print_error_message(ses_error_flag the_error_flag);
```

This function prints the error string for the given error flag.

ses_read_1D

```
ses_error_flag  ses_read_1D(    ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim);
```

This function uses the array interface, and reads the next one-dimensional array of size dim into the buffer.

ses_read_2D

```
ses_error_flag ses_read_2D(     ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim1,
                                ses_number dim2);
```

This function uses the array interface, and reads the next two-dimensional array of size dim1*dim2 into the buffer.

ses_read_3D

```
ses_error_flag ses_read_3D(     ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim1,
                                ses_number dim2,
                                ses_number dim3);
```

This function uses the array interface, and reads the next three-dimensional array of size dim1*dim2*dim2 into the buffer.

ses_read_named_array

```
ses_error_flag ses_read_named_array(ses_file_handle the_handle,
                                    ses_label the_label,
                                    ses_word_reference buf1);
```

This function reads the array associated with the label from the given file handle into the buffer.

ses_read_next

```
ses_word_reference  ses_read_next(ses_file_handle the_handle);
```

This function uses the iterator interface, and reads the next array on the file handle into the buffer.

ses_read_number

```
ses_error_flag ses_read_number(      ses_file_handle the_handle,
                                     ses_number_reference the_buffer);
```

This function uses the array interface, and reads the next number on the file handle into the buffer.

ses_read_pairs

```
ses_error_flag ses_read_pairs(    ses_file_handle the_handle,
                                  ses_word_reference buf1,
                                  ses_word_reference buf2,
                                  ses_number dim);
```

This function uses the array interface, and reads the next paired data on the file handle into the given two buffers.

ses_read_word

```
ses_error_flag ses_read_word(   ses_file_handle the_handle,
                                ses_word_reference the_buffer);
```

This function uses the array interface, and reads the next word on the file handle.

ses_set_array_order

```
ses_error_flag  ses_set_array_order(ses_file_handle the_handle,
                                    ses_array_order the_order);
```

This function sets the array order of arrays that are read.

ses_set_date

```
ses_error_flag ses_set_date(      ses_file_handle the_handle,
                                  long the_date);
```

This function sets the date to be written into a sesame file.

ses_set_format

```
ses_error_flag ses_set_format(   ses_number the_num_parameters,
                                 ses_file_handle the_handle,
                                 ses_file_type the_file_type
                                 ses_ascii_word_type the_word_type);
```

This function sets the format of the output sesame file.

ses_set_grid

```
ses_error_flag ses_set_grid(    ses_file_handle the_handle,
                                ses_number nr,
                                ses_number nt,
                                ses_number ntab);
```

This function sets the grid size to be written into a sesame file. The variable ntab should be -1 unless you are dealing with a table with a variable number of arrays, in which case ntab should be the number of arrays in the table.

ses_set_label

```
ses_error_flag ses_set_label(ses_file_handle the_handle,
                             ses_label the_label);
```

This function sets the label of the next array on a sesame file. It is currently NOT WRITTEN to the file, it is meant to be used internally.

ses_set_material_order

```
ses_error_flag  ses_set_material_order(ses_file_handle the_handle);
```

This function signifies that materials on the file should be ordered by ascending material id.

ses_set_significant_digits

```
ses_error_flag  ses_set_significant_digits(ses_file_handle the_handle,
                                           ses_number number_digits);
```

This function sets the number of "significant digits" to be written to the sesame file (followed by zero's).

ses_set_table_index

```
ses_boolean ses_set_table_index(ses_file_handle the_handle, long date1, long date2, long version);
```

This function sets the dates and versions for the table indices.

ses_setup

```
ses_error_flag  ses_setup(      ses_file_handle the_handle,
                                ses_material_id the_mid,
                                ses_table_id the_tid);
```

This section "sets up" the file to point at the given material id and table id.

ses_set_validate

```
ses_error_flag  ses_set_validate(ses_file_handle the_handle);
```

This functions signifies that data read in should be validated to ensure no NaN's are read.

ses_set_version

```
ses_error_flag ses_set_version(  ses_file_handle the_handle,
                                 long the_version);
```

This function sets the version to be written to the sesame file.

ses_set_table_index

```
ses_error_flag ses_set_table_index (     ses_file_handle the_handle,
                                         long date1,
                                         long date2,
                                         long version);
```

This section allows the user to change the dates and version on the current table index.

ses_skip

```
ses_error_flag ses_skip(ses_file_handle the_handle);
```

This function skips the next array on the handle; it is used with the iterator interface.

ses_version

```
ses_string ses_version( ses_file_handle the_handle);
```

This function gets the version from the file handle.

ses_write_1D

```
ses_error_flag ses_write_1D(     ses_file_handle the_handle,
                                 ses_word_reference the_buffer,
                                 ses_number dim);
```

This function is used with the array interface and writes the given buffer into the next array.

ses_write_2D

```
ses_error_flag ses_write_2D(     ses_file_handle the_handle,
                                 ses_word_reference the_buffer,
                                 ses_number dim1,
                                 ses_number dim2);
```

This function is used with the array interface and writes the given buffer into the next array.

ses_write_3D

```
ses_error_flag ses_write_3D(    ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim1,
                                ses_number dim2,
                                ses_number dim3);
```

This function is used with the array interface, and writes the given buffer into the next array.

ses_write_comments

```
ses_error_flag ses_write_comments(      ses_file_handle the_handle,
                                        ses_string the_comments,
                                        ses_number dim);
```

This function writes the given string into the comments field on the sesame file.

ses_write_next

```
ses_error_flag ses_write_next(  ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim,
                                ses_label the_label);
```

This function is used with the iterator interface and writes the given buffer into the next array.

ses_write_number

```
ses_error_flag ses_write_number(ses_file_handle the_handle,
                                ses_number the_buffer);
```

This function is used with the array interface and writes the given number (as a double) into the next word.

ses_write_pairs

```
ses_error_flag ses_write_pairs(  ses_file_handle the_handle,
                                 ses_word_reference buf1,
                                 ses_word_reference buf2,
                                 ses_number dim);
```

This function is used with the array interface and writes the next pair of arrays, pairwise, into the next array.

ses_write_setup

```
ses_error_flag ses_write_setup ( ses_file_handle the_handle,
                                 ses_material_id the_mid,
                                 ses_table_id the_tid,
                                 ses_number nr,
                                 ses_number nt,
                                 ses_number ntab);
```

This function setup the file handle for write, including implicit gridding information.

ses_write_word

```
ses_error_flag ses_write_word(  ses_file_handle the_handle,
                                ses_word the_buffer);
```

This function is used with the array interface and writes the given buffer into the next word.

8.2.2 F90 Function Reference

The F90 functions that correspond to the previously listed C functions are listed below.

ses_access_directory_f90

```
ses_number  ses_access_directory_f90(  ses_file_handle the_handle,
                                       ses_material_id_reference return_matid,
                                       LONG_POINTER, dimension(:)  nwds,
                                       LONG_POINTER, dimension(:) iadr,
                                       long date,
                                       long version)
```

This function accesses a sesame directory record directly and returns the information in that directory. See Appendix A for the information in a sesame directory record.

ses_access_table_index_f90

```
ses_number  ses_access_table_index_f90(ses_file_handle the_handle,
                                       ses_table_id_reference, dimension(:)  return_tblid,
                                       LONG_POINTER, dimension(:) nwds,
                                       LONG_POINTER, dimension(:) iadr,
                                       long date1,
                                       long date2,
                                       long version)
```

This function accesses a sesame table index record directly and returns the information in that index record. See Appendix A for the information in a sesame table index record.

ses_array_size_next_f90

```
ses_number    ses_array_size_next_f90(ses_file_handle the_handle)
```

This function is used with the read iterator interface. It returns the size of the next array at the current handle.

ses_change_next_f90

```
ses_error_flag   ses_change_next_f90( ses_file_handle the_handle,
                                      ses_word_reference the_buffer,
                                      ses_number dim)
```

This function is used with the write iterator interface. It allows the user to change the next array to the values in “the_buffer”.

ses_close_f90

```
ses_error_flag   ses_close_f90(ses_file_handle the_handle)
```

This function closes the current file handle.

ses_combine_f90

```
ses_error_flag  ses_combine_f90(ses_file_handle h1,
                                ses_file_handle h2,
                                ses_string new_filename)
```

This function combines two sesame files into the new file “new_filename”.

ses_comments_f90

```
ses_error_flag   ses_comments_f90(ses_file_handle the_handle,
                                  ses_string the_string)
```

This function returns comments from the given file handle.

ses_date_f90

```
ses_large_string, target  ses_date_f90(ses_file_handle the_handle)
```

This function returns the date from the sesame file directory.

ses_define_table_f90

```
ses_error_flag ses_define_table_f90(  ses_table_id tid,
                                      ses_label description,
                                      long nr,
                                      long nt,
                                      long numarrays,
                                      character(len = 60), pointer, dimension(:) size_arrays,
                                      character(len = 60), pointer, dimension(:) labels)
```

This function creates a user-defined table.

ses_delete_table_f90

```
ses_error_flag   ses_delete_table_f90(ses_file_handle the_handle)
```

This function deletes the table that has been “setup” (a material, table pair).

ses_exit_f90

```
ses_boolean   ses_exit_f90( )
```

This routine cleans up and deletes all global memory. Use with care.

ses_format_f90

```
SES_LARGE_STRING, target ses_format_f90(ses_file_handle the_handle)
```

This routine returns the format of the given file handle.

ses_get_comments_f90

```
ses_error_flag ses_get_comments_f90(ses_file_handle the_handle,
                                    ses_string the_comments)
```

This routine gets all the comments on the file handle.

ses_get_date_f90

```
LONG ses_get_date_f90(ses_file_handle the_handle)
```

This routine gets the date from the directory for the file handle as a long integer.

ses_get_grid_f90

```
ses_boolean ses_get_grid_f90( ses_file_handle  the handle,
                              ses_material_id the_mid,
                              ses_table_id the_tid,
                              long nr,
                              long nt,
                              long ntab)
```

This function returns the grid (nr and nt and ntab) for the material, table pair on the given file handle.

ses_get_label_f90

```
ses_label_string, target   ses_get_label_f90(ses_file_handle the_handle)
```

This function is used with the iterator interface. It returns the label of the next array on the file handle.

ses_get_material_id_f90

```
ses_material_id ses_get_material_id_f90(ses_string the_material_name)
```

This function returns the material id associated with the given material name.

ses_get_materials_f90

```
ses_material_id_reference ses_get_materials_f90(ses_file_handle the_handle,
                                                LONG_ARRAY_POINTER pt_size)
```

This function returns an array of the material id’s on the sesame file.

ses_get_table_ids_f90

```
ses_table_id_reference, dimension(:)  ses_get_table_ids_f90(  ses_file_handle the_handle,
                                                              ses_material_id mid,
                                                              LONG_POINTER, DIMENSION(:) pt_size)
```

This function returns a list of the table ids on the file associated with a given material id.

ses_get_table_sizes_f90

```
ses_table_sizes_reference  ses_get_table_sizes_f90(ses_file_handle the_handle,
                                                   ses_material_id mid,
                                                   long size)
```

This routine returns the table sizes for the given material.

ses_get_version_f90

```
LONG ses_get_version_f90(ses_file_handle the_handle)
```

This routine gets the version from the directory for the file handle as a long integer.

ses_has_next_f90

```
ses_boolean ses_has_next_f90(ses_file_handle the_handle)
```

This function is used in the iterator interface, and returns true if there are more arrays in the table.

ses_indicates_error_f90

```
ses_boolean   ses_indicates_error_f90(ses_error_flag the_ses_error_flag)
```

This flag returns true if the given error flag indicates an error has occurred.

ses_is_valid_f90

```
ses_boolean   ses_is_valid_f90(ses_file_handle the_handle)
```

This function returns true if the given file handle is connected correctly to a valid sesame file.

ses_open_f90

```
ses_file_handle ses_open_f90(  ses_string filename,
                               ses_open_type open_flags)
```

This function opens the sesame file and associates it with a file handle.

ses_print_error_condition_f90

```
ses_large_string, target      ses_print_error_condition_f90(ses_file_handle the_handle)
```

This function prints the last error condition on the given file handle.

ses_print_error_message_f90

```
ses_boolean ses_print_error_message_f90(ses_error_flag the_error_flag,
                                        ses_string, INTENT(OUT) the_output,
                                        long, INTENT(OUT) :: dim1)
```

This routine produces the error string for the given error flag.

ses_read_1D_f90

```
ses_error_flag ses_read_1D_f90(ses_file_handle the_handle,
                               ses_word_reference the_buffer,
                               ses_number dim)
```

This function uses the array interface, and reads the next one-dimensional array of size dim into the buffer.

ses_read_2D_f90

```
ses_error_flag ses_read_2D_f90( ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim1,
                                ses_number dim2)
```

This function uses the array interface, and reads the next two-dimensional array of size dim1*dim2 into the buffer.

ses_read_3D_f90

```
ses_error_flag ses_read_3D_f90( ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim1,
                                ses_number dim2,
                                ses_number dim3)
```

This function uses the array interface, and reads the next three-dimensional array of size dim1*dim2*dim2 into the buffer.

ses_read_named_array_f90

```
ses_error_flag ses_read_named_array_f90( ses_file_handle the_handle,
                                         ses_label the_label,
                                         ses_word_reference buffer)
```

This function reads the array associated with the label from the given file handle into the buffer.

ses_read_next_f90

```
ses_number  ses_read_next_f90(ses_file_handle the_handle,
                              ses_word, dimension(dim1), target the_buffer,
                              ses_number dim1)
```

This function uses the iterator interface, and reads the next array on the file handle into the buffer.

ses_read_number_f90

```
ses_error_flag ses_read_number_f90(    ses_file_handle the_handle,
                                       ses_number_reference the_buffer)
```

This function uses the array interface, and reads the next number on the file handle into the buffer.

ses_read_pairs_f90

```
ses_error_flag ses_read_pairs_f90( ses_file_handle the_handle,
                                   ses_word_reference buf1,
                                   ses_word_reference buf2,
                                   ses_number dim)
```

This function uses the array interface, and reads the next paired data on the file handle into the given two buffers.

ses_read_word_f90

```
ses_error_flag ses_read_word_f90(ses_file_handle the_handle,
                                 ses_word_reference the_buffer)
```

This function uses the array interface, and reads the next word on the file handle.

ses_set_array_order_f90

```
ses_error_flag  ses_set_array_order_f90(ses_file_handle the_handle,
                                        ses_array_order the_order)
```

This function sets the array order of arrays that are read.

ses_set_date_f90

```
ses_error_flag ses_set_date_f90(ses_file_handle the_handle,
                                long the_date)
```

This function sets the date to be written into a sesame file.

ses_set_format_f90

```
ses_error_flag ses_set_format_f90(ses_number the_number_parameters,
                                  ses_file_handle the_handle,
                                  ses_file_type the_file_type,
                                  ses_ascii_word_type the_word_type)
```

This function sets the format of the output sesame file.

ses_set_grid_f90

```
ses_error_flag ses_set_grid_f90(ses_file_handle the_handle,
                                ses_number nr,
                                ses_number nt,
                                ses_number ntab)
```

This function sets the grid size to be written into a sesame file. The variable ntab should be -1 unless you are dealing with a table with a variable number of arrays, in which case ntab should be the number of arrays in the table.

ses_set_label_f90

```
ses_error_flag ses_set_label_f90(ses_file_handle the_handle,
                                 ses_label the_label)
```

This function sets the label of the next array on a sesame file. It is currently NOT WRITTEN to the file, it is meant to be used internally.

ses_set_material_order_f90

```
ses_error_flag  ses_set_material_order_f90(ses_file_handle the_handle)
```

This function signifies that materials on the file should be ordered by ascending material id.

ses_set_significant_digits_f90

```
ses_error_flag  ses_set_significant_digits_f90(ses_file_handle the_handle,
                                               ses_number number_digits)
```

This function sets the number of "significant digits" to be written to the sesame file (followed by zero's).

ses_set_table_index_f90

ses_error_flag ses_set_table_index (ses_file_handle the_handle, long date1, long date2, long version)

This function sets the table indices for the given file handle.

ses_setup_f90

```
ses_error_flag  ses_setup_f90(ses_file_handle the_handle,
                              ses_material_id the_mid,
                              ses_table_id the_tid)
```

This section "sets up" the file to point at the given material id and table id.

ses_set_validate_f90

ses_error_flag ses_set_validate_f90(ses_file_handle the_handle)

This functions signifies that data read in should be validated to ensure no NaN's are read.

ses_set_version_f90

```
ses_error_flag ses_set_version_f90( ses_file_handle the_handle,
                                    long the_version)
```

This function sets the version to be written to the sesame file.

ses_set_table_index_f90

```
ses_error_flag ses_set_table_index_f90 (    ses_file_handle the_handle,
                                            long date1,
                                            long date2,
                                            long version)
```

This section allows the user to change the dates and version on the current table index.

ses_skip_f90

ses_error_flag ses_skip_f90(ses_file_handle the_handle)

This function skips the next array on the handle; it is used with the iterator interface.

ses_version_f90

ses_large_string, target ses_version_f90(ses_file_handle the_handle)

This function gets the version from the file handle.

ses_write_1D_f90

```
ses_error_flag ses_write_1D_f90(ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim)
```

This function is used with the array interface and writes the given buffer into the next array.

ses_write_2D_f90

```
ses_error_flag ses_write_2D_f90(ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim1,
                                ses_number dim2)
```

This function is used with the array interface and writes the given buffer into the next array.

ses_write_3D_f90

```
ses_error_flag ses_write_3D_f90(ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                               ses_number dim1,
                               ses_number dim2,
                               ses_number dim3);
```

This function is used with the array interface, and writes the given buffer into the next array.

ses_write_comments_f90

```
ses_error_flag ses_write_comments_f90(ses_file_handle the_handle,
                                      ses_string the_comments,
                                      ses_number dim)
```

This function writes the given string into the comments field on the sesame file.

ses_write_next_f90

```
ses_error_flag ses_write_next_f90(ses_file_handle the_handle,
                                  ses_word_reference the_buffer,
                                  ses_number dim,
                                  ses_label the_label)
```

This function is used with the iterator interface and writes the given buffer into the next array.

ses_write_number_f90

```
ses_error_flag ses_write_number_f90(ses_file_handle the_handle,
                                    ses_number the_buffer)
```

This function is used with the array interface and writes the given number (as a double) into the next word.

ses_write_pairs_f90

```
ses_error_flag ses_write_pairs_f90(ses_file_handle the_handle,
                                   ses_word_reference buf1,
                                   ses_word_reference buf2,
                                   ses_number dim)
```

This function is used with the array interface and writes the next pair of arrays, pairwise, into the next array.

ses_write_setup_f90

```
ses_error_flag ses_write_setup_f90 ( ses_file_handle the_handle,
                                     ses_material_id the_mid,
                                     ses_table_id the_tid,
                                     ses_number nr,
                                     ses_number nt,
                                     ses_number ntab)
```

This function setup the file handle for write, including implicit gridding information.

ses_write_word_f90

```
ses_error_flag ses_write_word_f90(ses_file_handle the_handle,
                                  ses_word_reference the_buffer)
```

This function is used with the array interface and writes the given buffer into the next word.

8.2.3 Java Function Reference

The Java functions that correspond to the previously listed C functions are listed below.

SesAccessDirectory

```
ses_number      SesAccessDirectory(ses_file_handle the handle,
                                   ses_material_id_reference return_matid,
                                   long* nwds,
                                   long* iadr,
                                   long date,
                                   long version);
```

This function accesses a sesame directory record directly and returns the information in that directory. See Appendix A for the information in a sesame directory record.

SesAccessTableIndex

```
ses_number      SesAccessTableIndex( ses_file_handle the_handle,
                                     ses_table_id_reference return_tid,
                                     long* nwds,
                                     long* iadr,
                                     long date1,
                                     long date2,
                                     long version);
```

This function accesses a sesame table index record directly and returns the information in that index record. See Appendix A for the information in a sesame table index record.

SesArraySizeNext

```
ses_number      SesArraySizeNext(ses_file_handle the_handle);
```

This function is used with the read iterator interface. It returns the size of the next array at the current handle.

SesChangeNext

```
ses_error_flag SesChangeNext( ses_file_handle the_handle,
                              ses_word_reference the_buffer,
                              ses_number dim);
```

This function is used with the write iterator interface. It allows the user to change the next array to the values in “the_buffer”.

SesClose

```
ses_error_flag  SesClose(ses_file_handle the_handle);
```

This function closes the current file handle.

SesCombine

```
ses_error_flag  SesCombine(     ses_file_handle h1,
                                ses_file_handle h2,
                                ses_string new_filename);
```

This function combines two sesame files into the new file “new_filename”.

SesComments

```
ses_string SesComments(ses_file_handle the_handle);
```

This function returns comments from the given file handle.

SesDate

```
ses_string  SesDate(ses_file_handle the_handle);
```

This function returns the date from the sesame file directory.

SesDefineTable

```
ses_error_flag SesDefineTable( ses_table_id tid,
                               ses_string description,
                               long nr,
                               long nt,
                               long num_arrays,
                               ObjectArray sizearrays,
                               ObjectArray labels);
```

This function creates a user-defined table.

SesDeleteTable

```
ses_error_flag   SesDeleteTable(ses_file_handle the_handle);
```

This function deletes the table that has been “setup” (a material, table pair).

SesExit

```
ses_boolean SesExit();
```

This routine cleans up and deletes all global memory. Use with care.

SesGetGrid

```
ses_boolean SesGetGrid(     ses_file_handle the_handle,
                            ses_material_id the_mid,
                            ses_table_id the_tid,
                            long* nr,
                            long* nt);
```

This function returns the grid (nr and nt) for the material, table pair on the given file handle.

SesGetLabel

```
ses_label    SesGetLabel(ses_file_handle the_handle);
```

This function is used with the iterator interface. It returns the label of the next array on the file handle.

SesGetMaterialId

```
ses_material_id SesGetMaterialId(ses_string material_name);
```

This function returns the material id associated with the given material name.

SesGetMaterials

```
ses_material_id_reference SesGetMaterials(ses_file_handle the_handle, long* size);
```

This function returns an array of the material id's on the sesame file.

SesGetTableIds

```
ses_table_id_reference  SesGetTableIds(ses_file_handle the_handle,
                                       ses_material_id mid);
```

This function returns a list of the table ids on the file associated with a given material id.

SesGetTableSizes

```
ses_table_sizes_reference SesGetTableSizes(   ses_file_handle the_handle,
                                              ses_material_id mid,
                                              long* size);
```

This routine returns the table sizes for the given material.

SesHasNext

```
ses_boolean SesHasNext(ses_file_handle the_handle);
```

This function is used in the iterator interface, and returns true if there are more arrays in the table.

SesIndicatesError

```
ses_boolean     SesIndicatesError(ses_error_flag the_ses_error_flag);
```

This flag returns true if the given error flag indicates an error has occurred.

SesIsValid

```
ses_boolean     SesIsValid(ses_file_handle the_handle);
```

This function returns true if the given file handle is connected correctly to a valid sesame file.

SesOpen

```
ses_file_handle SesOpen(ses_string filename,
                        ses_open_type open_flags);
```

This function opens the sesame file and associates it with a file handle.

SesPrintErrorCondition

```
ses_string      SesPrintErrorCondition(ses_file_handle the_handle);
```

This function prints the last error condition on the given file handle.

SesRead1D

```
doubleArray SesRead1D(ses_file_handle the_handle,
                      ses_number dim);
```

This function uses the array interface, and reads the next one-dimensional array of size dim into the buffer.

SesRead2D

```
doubleArray SesRead2D(ses_file_handle the_handle,
                      ses_number dim1,
                      ses_number dim2);
```

This function uses the array interface, and reads the next two-dimensional array of size dim1*dim2 into the buffer.

SesRead3D

```
doubleArray SesRead3D(ses_file_handle the_handle,
                      ses_number dim1,
                      ses_number dim2,
                      ses_number dim3);
```

This function uses the array interface, and reads the next three-dimensional array of size dim1*dim2*dim2 into the buffer.

SesReadNamedArray

```
doubleArray SesReadNamedArray(ses_file_handle the_handle,
                              ses_label the_label,
                              int dim);
```

This function reads the array associated with the label from the given file handle into the buffer.

SesReadNext

```
doubleArray  SesReadNext(ses_file_handle the_handle,
                         int dim);
```

This function uses the iterator interface, and reads the next array on the file handle into the buffer.

SesReadNumber

```
ses_number  SesReadNumber(ses_file_handle the_handle);
```

This function uses the array interface, and reads the next number on the file handle into the buffer.

SesReadPairs

```
ses_error_flag SesReadPairs(    ses_file_handle the_handle,
                                ses_word_reference buf1,
                                ses_word_reference buf2,
                                ses_number dim);
```

This function uses the array interface, and reads the next paired data on the file handle into the given two buffers.

SesReadWord

```
ses_word SesReadWord(ses_file_handle the_handle);
```

This function uses the array interface, and reads the next word on the file handle.

SesSetArrayOrder

```
ses_error_flag  SesSetArrayOrder(ses_file_handle the_handle,
                                 ses_array_order the_order);
```

This function sets the array order of arrays that are read.

SesSetDate

```
ses_error_flag  SesSetDate(ses_file_handle the_handle,
                           long the_date);
```

This function sets the date to be written into a sesame file.

SesSetFormat

```
ses_error_flag SesSetFormat(  ses_number the_num_parameters,
                                  ses_file_handle the_handle,
                              ses_file_type the_file_type
                                ses_ascii_word_type the_word_type);
```

This function sets the format of the output sesame file.

SesSetGrid

```
ses_error_flag  SesSetGrid(ses_file_handle the_handle,
                           ses_number nr,
                           ses_number nt,
                           ses_number ntab);
```

This function sets the grid size to be written into a sesame file. The variable ntab should be -1 unless you are dealing with a table with a variable number of arrays, in which case ntab should be the number of arrays in the table.

SesSetLabel

```
ses_error_flag SesSetLabel(ses_file_handle the_handle,
                           ses_label the_label);
```

This function sets the label of the next array on a sesame file. It is currently NOT WRITTEN to the file, it is meant to be used internally.

SesSetMaterialOrder

```
ses_error_flag  SesSetMaterialOrder(ses_file_handle the_handle);
```

This function signifies that materials on the file should be ordered by ascending material id.

SesSetSignificantDigits

```
ses_error_flag  SesSetSignificantDigits(ses_file_handle the_handle,
                                        ses_number number_digits);
```

This function sets the number of "significant digits" to be written to the sesame file (followed by zero's).

SesSetValidate

```
ses_error_flag  SesSetValidate(ses_file_handle the_handle);
```

This functions signifies that data read in should be validated to ensure no NaN's are read.

SesSetVersion

```
ses_error_flag  SesSetVersion(ses_file_handle the_handle,
                              long version);
```

This function sets the version to be written to the sesame file.

SesSetup

```
ses_error_flag  SesSetup(ses_file_handle the_handle,
                         ses_material_id the_mid,
                         ses_table_id the_tid);
```

This section "sets up" the file to point at the given material id and table id.

SesSetTableIndex

```
ses_error_flag SesSetTableIndex (       ses_file_handle the_handle,
                                        long date1, long date2, long version);
```

This section allows the user to change the dates and version on the current table index.

SesSkip

```
ses_error_flag SesSkip(ses_file_handle the_handle);
```

This function skips the next array on the handle; it is used with the iterator interface.

SesVersion

```
ses_string SesVersion(ses_file_handle the_handle);
```

This function gets the version from the file handle.

SesWrite1D

```
ses_error_flag SesWrite1D(ses_file_handle the_handle,
                          ses_word_reference the_buffer,
                          ses_number dim);
```

This function is used with the array interface and writes the given buffer into the next array.

SesWrite2D

```
ses_error_flag SesWrite2D(ses_file_handle the_handle,
                          ses_word_reference the_buffer,
                          ses_number dim1,
                          ses_number dim2);
```

This function is used with the array interface and writes the given buffer into the next array.

SesWrite3D

```
ses_error_flag SesWrite3D(ses_file_handle the_handle,
                          ses_word_reference the_buffer,
                          ses_number dim1,
                          ses_number dim2,
                          ses_number dim3);
```

This function is used with the array interface, and writes the given buffer into the next array.

SesWriteComments

```
ses_error_flag  SesWriteComments(ses_file_handle the_handle,
                                 ses_string comments,
                                 long dim);
```

This function writes the given string into the comments field on the sesame file.

SesWriteNext

```
ses_error_flag SesWriteNext(    ses_file_handle the_handle,
                                ses_word_reference the_buffer,
                                ses_number dim,
                                ses_string label);
```

This function is used with the iterator interface and writes the given buffer into the next array.

SesWriteNumber

```
ses_error_flag SesWriteNumber(ses_file_handle the_handle,
                              ses_number the_buffer);
```

This function is used with the array interface and writes the given number (as a double) into the next word.

SesWritePairs

```
ses_error_flag SesWritePairs(    ses_file_handle the_handle,
                                 ses_word_reference buf1,
                                 ses_word_reference buf2,
                                 ses_number dim);
```

This function is used with the array interface and writes the next pair of arrays, pairwise, into the next array.

SesWriteSetup

```
ses_error_flag SesWriteSetup(    ses_file_handle the_handle,
                                 ses_material_id the_mid,
                                 ses_table_id the_tid,
                                 ses_number nr,
                                 ses_number nt,
                                 ses_number ntab);
```

This function setup the file handle for write, including implicit gridding information.

SesWriteWord

```
ses_error_flag SesWriteWord(     ses_file_handle the_handle,
                                 ses_word the_buffer);
```

This function is used with the array interface and writes the given buffer into the next word.

Appendix A: The Sesame Format

Sesame Format, Version 1

Sesame Library := Ascii_Library | Binary_Library

Ascii_Library := Sequential_Directory_File (Sequential_Material_File)+ EOF_Mark
EOF_Mark

Sequential_Directory File := Sequential_Record_1 Sequential_Record_2 File_Mark
Sequential_Record_1 := Record_1 Record_Mark
Sequential_Record_2 := Record_2 Record_Mark

Sequential_Material_File := Sequential_Index_Record (Sequential_Data_Record)+ File_Mark
Sequential_Index_Record := Index_Record Record_Mark
Sequential_Data_Record = Data_Record Record_Mark

Binary_Library := Directory_File (Material_File)+

Directory File := Record_1 Record_2

Record_1 := Nfiles Date Version

Nfiles := number of files in this library
Date := date of the version used
Version := version number of the version used

Record_2 := Matid (1..N) NWDS (1..N) IADR (1..N)
Note: Record_2 is dependent on the value of N is Record_1.

Matid(I) := Material id # for file number I
NWDS(I) := Number of words in file number I
IADR(I) := address of the index record for material file I

Material_File := Index_Record (Data_Record)+

Index_Record := Matid Date1 Date2 Vers Nrec TblId(1..Nrec) Nwds (1..Nrec) Iadr (1..Nrec)
Note: Matid here must be the same as the Matid listed in Record_2 (it must correspond to the Matid(I) for the Ith File in this Sesame Library). The KIND of the Data Records that follow must correspond to the TableID's in the list of the Index Record.

Data_Record := 101_Data_Record | 102_Data_Record | 201_Data_Record | 301_Data_Record |
303_Data_Record | 304_Data_Record | 305_Data_Record | 306_Data_Record |
401_Data_Record | 411_Data_Record | 412_Data_Record | 431_Data_Record |
501_Data_Record | 502_Data_Record | 503_Data_Record | 504_Data_Record |
505_Data_Record | 601_Data_Record | 602_Data_Record | 603_Data_Record |
604_Data_Record | 605_Data_Record

101_Data_Record := Comments
102_Data_Record := Comments

201_Data_Record := Zbar, Abar, Rho0, B0, Cex (201 Atomic Number, Atomic Mass, Normal Density)

Question: What are the Units?

301_Data_Record := Low_300_Data_Record (301 Total EOS (304 + 305 + 306))
303_Data_Record := Low_300_Data_Record (303 Ion EOS Plus Cold Curve (305 + 306))
304_Data_Record := Low_300_Data_Record (304 Electron EOS)
305_Data_Record := Low_300_Data_Record (305 Ion EOS (Including Zero Point))

Low_300_Data_Record := NR NT R(1..NR) T(1..NT) P((1..NR)(1..NT)) E((1..NR)(1..NT))
A((1..NR)(1..NT))

NR = number of densities
NT = number of temperatures

R = density array, Mg/m3
T = temperature array, K
P = pressure array, GPa
E = energy array, MJ/kg
A = free energy array, MJ/kg

306 Cold Curve (No Zero Point)

306_Data_Record := NR 1 R(1..NR) 0 P(1..NR) E(1..NR) A(1..NR)

NR = number of densities
NT = number of temperatures = 1
R = density array, Mg/m3
T = temperature array = 0
P = pressure array, Gpa
E = energy array, MJ/kg
A = free energy array, MJ/kg

401 Vaporization Table

401_Data_Record := NT P(1..NT) T(1..NT) RG(1..NT) RL(1..NT) EG(1..NT) EL(1..NT) AG(1..NT)

NT = number of temperatures

P = Vapor pressure, Gpa
T = Temperature, K
RG = Vapor density on coexistence line, Mg/m3
RL = Density of liquid or solid on coexistence line, Mg/m3
EG = Internal energy of vapor on coesistence lin, MJ/kg
EL = Internal energy of liquid or solid on coexistence line, MJ/kg
AG = Free energy of vapor on coexistence line, MJ/kg
AL = Free energy of liquid or solid on coexistence line, MJ/kg

411 Solid Melt Table

411_Data Record := NR 1 RS(1..NR) 0 TM(1..NR) PM(1..NR) ES(1..NR) AS(1..NR)

NR = Number of densities
NT = Number of temperatures = 1

RS = Density of solid on melt, Mg/m3
T = temperature, K = 0
TM = melt temperature, K
PM = melt pressure, Gpa
ES = Internal energy of solid on the melt line, MJ/kg
AS = Free energy of solid on the melt line, MJ/kig

412 Liquid Melt Table

412_Data_Record := NR 1 RS(1..NR) 0 TM(1..NR) PM(1..NR) ES(1..NR) AS(1..NR)

NR = number of densities
NT = number of temperatures = 1

RS = Density of solid on melt, Mg/m3
T = Temperature, K = 0
TM = Melt temperature, K
PM = Melt pressure, Gpa
ES = Internal energy of liquid on the melt line, MJ/kg
AS = Free energy of liquid on the melt line, MJ/kg

431 Shear Modulus Table

431_Data_Record := NR 1 R(1..NR) 0 G(1..NR)

NR = number of densities
NT = number of temperatures = 1
R = density array, Mg/m3
T = temperature array, T=0
G = shear modulas array, Gpa

501 Opacity Grid Boundary: Calculated vs. Interpolated

501_Data_Record := N (T(I), R(I), I=1..N)

N = number of temperature-density pairs
T = temperature, K
R = density, Mg/m3

502_Data_Record := 600_Data_Record (502 Rosseland Mean Opacity (cm2/g))
503_Data_Record := 600_Data_Record (503 Electron Conductive Opacity1 (cm2/g))
504_Data_Record := 600_Data_Record (504 Mean Ion Charge1 (free electrons per atom))
505_Data_Record := 600_Data_Record (505 Planck Mean Opacity (cm2/g))

601_Data_Record := 600_Data_Record (601 Mean Ion Charge2 (free electrons per atom))
602_Data_Record := 600_Data_Record (602 Electrical Conductivity (per sec))
603_Data_Record := 600_Data_Record (603 Thermal Conductivity (per cm-sec))
604_Data_Record := 600_Data_Record (604 Thermoelectric Coefficient (per cm-sec))
605_Data_Record := 600_Data_Record (605 Electron Conductive Opacity2 (cm2/g))

Note:

1 = Opacity Model (Hubbard-Lampe)

2 = Conductivity Model (Ziman)

600_Data_Record := NR NT R(1..NR) T(1..NT) (X(I,J), I=1..NR, J=1..NT)

NR = number of densities
NT = number of temperatures
R = LOG10 density array, g/cm3
T = LOG10 density array, eV
X = LOG10 quantity

Appendix B – Format for User-Define Table Strings

Strings in an extended (includes expression) json format are as follows:

{ members , members* }

meaning one or more name, value pairs, separated by commas, surrounded by braces.

The grammar is as follows:

object : { members, members* }
members := pair | members*
pair : = name : value
name := characters
value : = object | array | string | number | boolean | null | expression
array : [element , element *]
element : = string | number | expression
string := " characters "
number : = integer | floating_point
boolean := true | false
null := null
expression := (name)

Examples of valid table definitions are as follows:

{ tid: 801, number_arrays: 2, sizes:[(nr), (nt)], labels:["label for nr", "label for nt"] }

The ses_io library uses these strings to define standard sesame tables; for example, the 301 table is defined by

{ tid: 301, num_arrays : 7, num_independent : 2, description : "Total EOS (304 + 305 + 306)",
sizes : [(1), (1), (nr), (nt), (nr*nt), (nr*nt), (nr*nt)] ,
labels: ["nr (number densities)","nt (number temperatures)", "r - density (Mg/m3)", "t - temperature (K)", "p - pressure (GPa)", "e - energy (MJ/kg)", "a - free energy (MJ/kg)"] }

Appendix C – Sesame Type Definitions

Name	C Type	Fortran Type	Default Value
ses_array_order	int	integer(kind=4)	
ses_ascii_word_size	int	integer(kind=4)	22 (for medium)
ses_ascii_word_type	char	character(len=1)	M (for medium)
ses_boolean	int	integer(kind=4)	
ses_error_flag	int	integer(kind=4)	
ses_file_format_type	char	character(len=1)	B (for binary)
ses_file_handle	int	integer(kind=4)	
ses_file_format_type	char	character(len=1)	
ses_label	char *	character(len=60), pointer	
ses_material_id	long	integer(kind=8)	
ses_material_id_reference	long *	integer(kind=8), dimension(:), pointer	
ses_number	long	Integer(kind=8)	
ses_number_reference	long *	integer(kind=8), dimension(:), pointer	
ses_open_type	char	character(len=1)	
ses_string	char *	character(len=270), pointer	
ses_table_id	long	integer(kind=8)	
ses_table_id_reference	long *	integer(kind=8), dimension(:), pointer	
ses_table_sizes_references	long *	integer(kind=8), pointer	
ses_word	double	real(kind=8)	
ses_word_reference	double *	real(kind=8), dimension(:), pointer	

Appendix D – Sesame Error Values

Return Value	Keyword
0	SES_NO_ERROR
1	SES_ARRAY_SIZE_ERROR
2	SES_MEMPRY_ALLOCATION_ERROR
3	SES_OBJECT_CONSTRUCTION_ERROR
4	SES_NULL_OBJECT_ERROR
5	SES_OBJECT_COPY_ERROR
6	SES_OBJECT_DESTRUCTION_ERROR
7	SES_FUNCTION_FAIL_ERROR
8	SES_INVALID_FILE_HANDLE
9	SES_INVALID_MID
10	SES_OBJECT_READY_ERROR
11	SES_OPEN_ERROR
12	SES_INVALID_TID
13	SES_OBJECT_OUT_OF_RANGE
14	SES_SETUP_ERROR
15	SES_CLOSE_ERROR
16	SES_FILE_READY_ERROR
17	SES_FILE_WRITE_ERROR
18	SES_READ_ERROR
19	SES_WRITE_ERROR
20	SES_CHANGE_ERROR
21	SES_COMBINE_ERROR
22	SES_COMMENTS_ERROR
23	SES_DELETE_ERROR
24	SES_NOT_IMPLEMENTED
25	SES_INVALID_OPEN_TYPE
26	SES_DOUBLE_SIZE_ERROR
27	SES_TEST_ERROR
28	SES_APPEND_ERROR
29	SES_NO_DATA_ERROR
30	SES_INVALID_FILE_FORMAT_TYPE
31	SES_INVALID_ASCII_WORD_TYPE
32	SES_INVALID_NUM_PARAMETERS
33	SES_INVALID_SESAME_ASCII