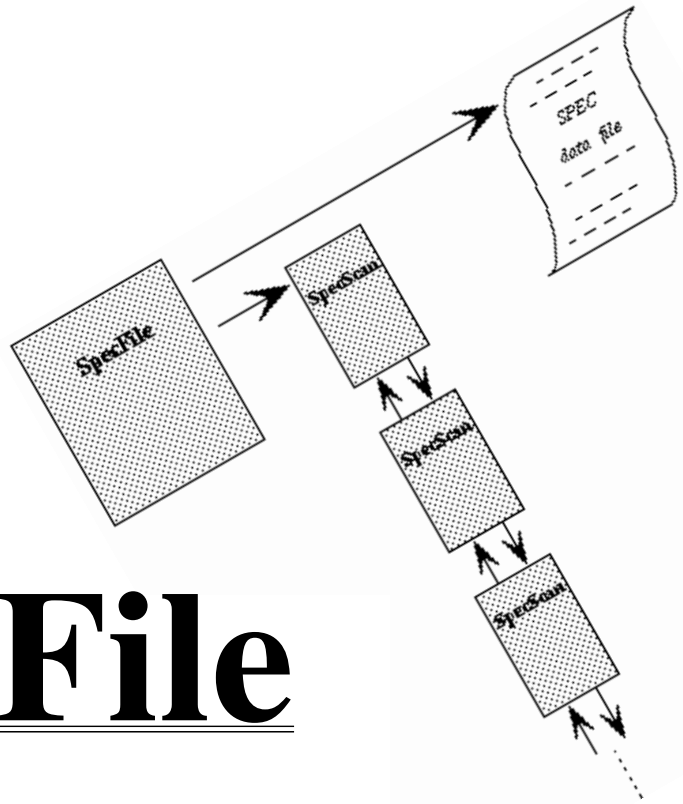
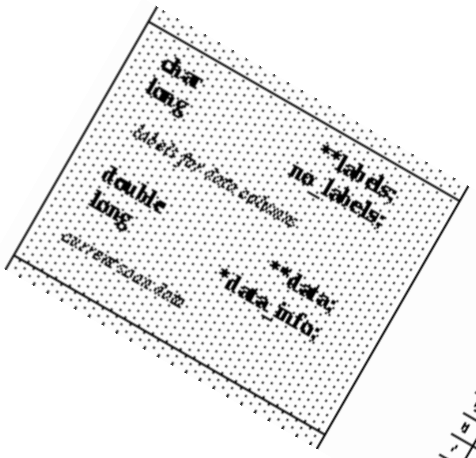
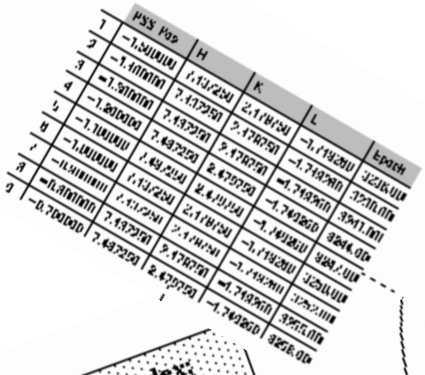



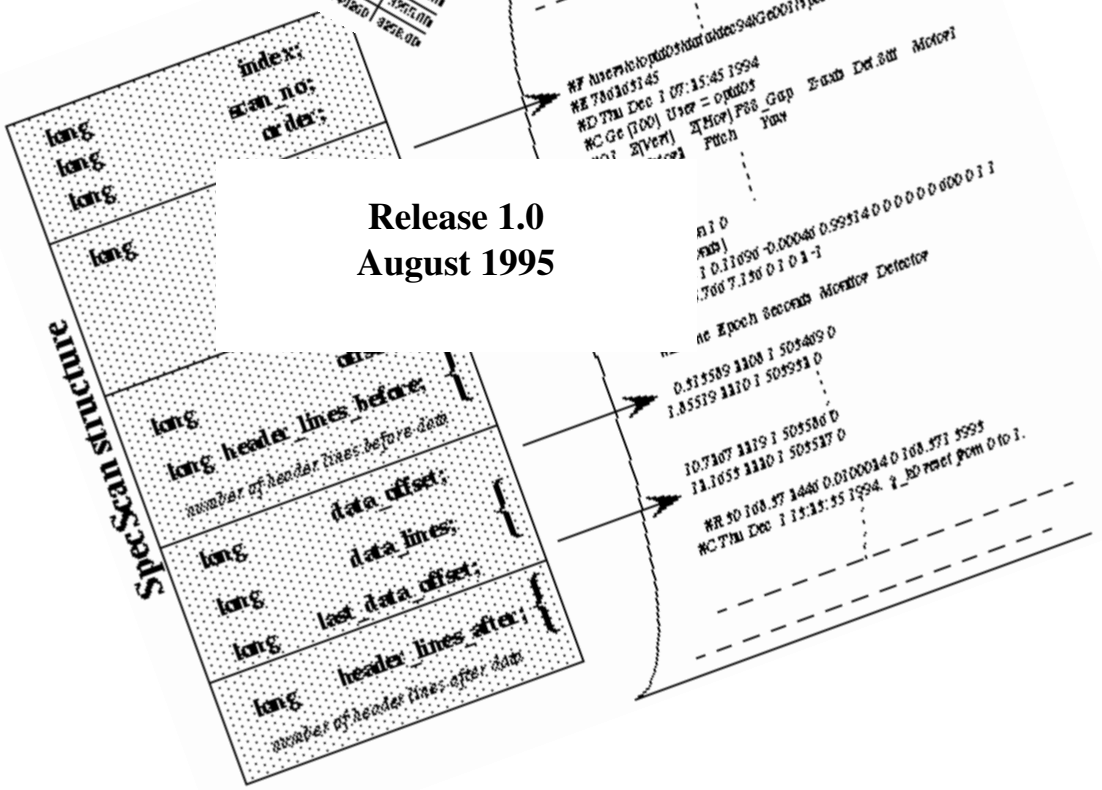

# SpecFile

## Library

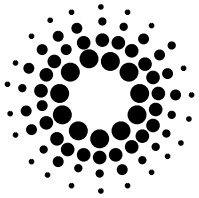
## Manual




**Release 1.0**
**August 1995**



# 1.0 Preface

This is a guide that describes how to use **SpecFile** functions inside C programs. It is also a complete reference to the full set of **SpecFile Library** functions.

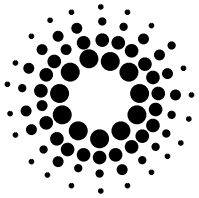
This document assumes that the reader has a good understanding of the C language [Kernigham].
It also assumes that the reader is familiar with SPEC [CSS], scans and data files in general.

We are conscious that as any big programming project may have errors, ours will not be an exception and we kindly ask the reader to report to *us the problem that may be encountered.

---

* *Postal Address :* *ESRF*
*Experiments Division Programming Group*
*BP220 - F-38043 Grenoble Cedex*
*France*

*e-mail:* *rey@esrf.fr*
*jack@foo.fh-furtwangen.de*





# 2.0 Introduction

## *2.1 SPEC*

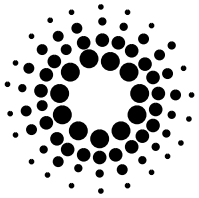
*SPEC is a commercial software for x-ray diffraction control and data acquisition. It is now used at over 60 locations in 120 laboratories all around the world. See [CSS] for a complete documentation. The most typical data acquisition sequence consists in moving a certain motor (or motors ) along a predefined set of positions and triggering a counting sequence at each of these points. This is called a *scan*. SPEC is able to save the obtained data, together with other values in a well defined format : the SPEC data file format.

## *2.2 SPEC Data Files*

SPEC data files are ASCII files. In the file, there are lines called *header lines.* All header lines start with a *'#'* followed by one or more characters identifying the information that follows and clearly described in SPEC documentation. For example *'#C'* begins a comment line, *'#P'* begins lines with motor positions etc. The list of all can be found in appendix A. Also other information can be introduced by the user by means of his own SPEC macro and using his own identifier.
*'#F'* and *'#S'* header lines define *file header* and *scan* blocks respectively. In this way a SPEC file can be seen as a sequence of blocks. A block starts with a *'#F'* or *'#S'* line and finishes when the next *'#F'* or *'#S'* is found.

---

* *SPEC is a trademark of Certified Scientific Software*



A *file header* consists only of *header lines*. It is created every time a "*newfile"* macro is issued in SPEC. It contains general file creation information:

*#F* - file name,
*#E* - epoch,
*#D* - file time and date,
*#C* - first comment includes SPEC title and SPEC user,
*#O* - motor names,

but also other lines may be added by the user. Normally only one file header block appears at the beginning of a file. It may occur that another file header appears somewhere else in the file or that there is no file header at all. In all cases the information contained in a file header covers all the scan blocks that follow up to the next file header.
A *scan* block starts with a *'#S'* line. This line shows also the scan number automatically assigned by SPEC plus the *command* that originated this scan. Other standard information that may immediately follow is:

*#D* - scan time and date,
*#G* - geometry values,
*#Q* - H, K, L values,
*#P* - positions of each motor,
*#N* - number of data columns
*#L* - column labels.

Typically after that , there is a block of *data lines* ( with no header or blank lines ) an another group of *header lines* untill the beginning of the next scan of file header.
A typical SPEC data file is shown in *fig.1*.

## *2.3 SpecFile Library*

The files generated in this way can become big and complex, with MBytes of data and thousands of scans. A quick access to a certain scan or even certain line in the file is not easy and represents always a lot of programming work. The **Specfile Library** allows to read any kind of information from SPEC data files in the fastest way.
The library creates an index of the file which is kept in memory. All functions of the library use it. In this way the access to a particular scan or commonly asked information is optimized.

An interesting utility provided by the library is the possibility to copy scans with associated file headers from the opened file into another one.

**FILE HEADER**

```
#F /users/c/opid03/data/dec94/Ge001/spec.Ge001_2x1
#E 786263145
#D Thu Dec  1 07:25:45 1994
#C Ge (100)  User = opid03
#O0    Delta    Theta      Chi      Phi       Mu     Gamma   GamTran    GamRot
#O1   X(Vert)   Z(Hor)     Mono   PSS_Pos   PSS_Gap   Z-axis  Det.Slit   Motor1
#O2    Motor2    Pitch      Yaw
```

**SCAN HEADER**

Scan 1

```
#S 2timescan 1 0
#D Thu Dec  1 08:02:32 1994
#T 1  (Seconds)
#G0 13 0 1 0.11696 -0.00046 0.99314 0 0 0 0 0 0 600 0 1 1
#G1 4.001 4.001 5.658 90 90 90 1.5704 1.5704 1.1105 90 90 90 1 0 1 1 0 3 18.016 -107.5 0.46 152.594 14 3.884 13.001 -53.06 0.46 152.594 14
43.482
#Q 7.43725 2.47975 -1.74926
#P0 100 0 0.38 109.2 0 0 0 0
#P1 0 -5.766 7.136 0 1 0 2 -1
#P2 0 0 0
#N 5
#L Time  Epoch  Seconds  Monitor  Detector
```

**SCAN DATA**

```
0.313589 2208 1 503469 0
1.85519 2210 1 503932 0
3.37135 2211 1 503905 0
4.84828 2213 1 503317 0
6.30889 2214 1 503496 0
7.78898 2216 1 503657 0
9.27692 2217 1 503343 0
10.7267 2219 1 503586 0
12.1653 2220 1 503527 0
                .
                .
                .
```

**SCAN HEADER LINES AFTER DATA**

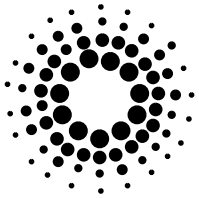
```
#R 30 168.37 2446 0.0100024 0 168.371 3993
#C Thu Dec  1 13:23:35 1994.  g_k0 reset from 0 to 1.
#C Thu Dec  1 13:23:35 1994.  g_l0 reset from 1 to 2.
```

**NEXT SCAN**

Scan 2

```
#S 4  ascan  pssp -1.5 1.5  30 1
#D Thu Dec  1 08:19:24 1994
#T 1  (Seconds)
#G0 13 0 1 0.11696 -0.00046 0.99314 0 0 0 0 0 0 600 0 1 1
                .
                .
                .
```

*fig.1*



# 3.0 Programming Guide

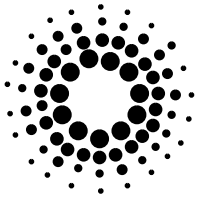
The **SpecFile Library** presents files as entities to which the programmer can ask for information. This information is organized in blocks that correspond to scans. To access a piece of information we will always need a "*handle"* to the file plus a scan *index*.

## *3.1 Initialization*

All **SpecFile** applications must include the file *<SpecFile.h>*. This file contains definitions that all applications need.
The first necessary operation on a SPEC file is creating the previously mentioned *"handle"*. The *SfOpen()* function gets the name of the file to open and returns a pointer to a structure *SpecFile* that will be needed for subsequent access. This structure contains not only file descriptor but also a list with indexes to scans to accelerate data access ( see Appendix B for exact definitions of all structures ).

```
#include <SpecFile.h>

main( argc, argv )
int                 argc;
char                *argv[];
{
    SpecFile        *sf;
    int             error=0;
    long            num_scans;
    long            scan_list;
                .
                .
                .
   /* Open a file specified by the first command line parameter. */
   if ( argc < 2 ) {
        fprintf( stderr,"Usage: %s file name\n", argv[0] );
        exit( 1 );
   }
   sf = SfOpen( argv[1], &error );

   /* Check if successful. */
   if ( sf == (SpecFile *)NULL ) {
        printf( "%s\n", SfError( error ) );
        exit( 1 );
   } else {
        if ( error ) {
        /* see SfOpen() for the explanation. */
        printf( "Could not read whole file !\n" );
        printf( "Occurred error :%s\n", SfError( error ) );
        error = 0;
        }
   }
   printf("Connection to the file %s successful\n", argv[1] );
```

*fig.2*





## *3.2 Scan Number and Scan Index*

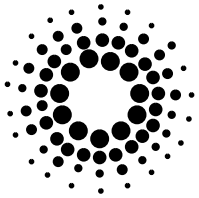
Ideally when a user creates a new SPEC file, the first scan is assigned *number* 1 and following scans have consecutive numbers. This is not always the case. SPEC allows the user to change the current scan number at any point. In this way it is not rare to find files in which the same scan number appears two or more times. For example a possible sequence could be:

| | |
|---|---|
| 3, 4, 5, 6, 7, 7, 6, 7, 9, 20, 10... | scan *number* |
| *1, 1, 1, 1, 1, 2, 2, 3, 1, 1, 1 ...* | scan *order* |
| 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ... | scan *index* |

If there are several scans with the same number in one file, scan *order* ( generated by the library ) helps to distinguish between them. The first scan with a certain number gets always the order 1. The next found with the same number has then the order 2 and so on ( see example above ).
The **SpecFile Library** generates an *index* for each scan in the file. The first scan in the file as assigned scan index 1 and the following scans have consecutive indexes. The *index*, and not the scan *number*, is the parameter that programmer must use to access a scan. The library provides the user with a set of functions that makes the link between indexes and scan numbers and orders.
*SfIndex()* returns scan index, the input values are scan number and order. To get scan number and order knowing its index *SfNumber()* and *SfOrder()* or *SfNumberOrder()* functions can be used.
*SfList()* is a function that returns a list of numbers of all scans in the file.
*SfCondList()* returns only numbers of aborted or not aborted scans or a list of numbers of scans with more data lines than a given value. The returned scan number list depends on the used condition.
The total number of scans in a file can be retrieved calling *SfScanNo()*.

```
                .
                .
                .
num_scans = SfScanNo( sf );
printf(" There are %li scans in the file %s .\n", num_scans, arg[1] );
scan_list = SfList( sf, &error );
if ( scan list == (long *)NULL ) {
        printf("%s\n", SfError( error ) );
        exit( 1 );
}
for ( i=0 ; i<num_scans ; i++ )
        printf( "%li scan has the number %li .\n", i+1, scam_list[i] );
free( scan_list );
                .
                .
                .
printf(" Enter scan index :");
fgets( buffer, 99, stdin );
index = atol( buffer );

printf(" %li scan has the number %li and the order %li .\n", index, SfNumber( sf, index ), SfOrder( sf, index ) );
```

*fig.3*



## *3.3 Header Lines*

Together with the data there is other information in a SPEC file : scan commands, geometry parameters, motor names and positions etc. This information is stored in *header lines*. For a particular scan index all lines before and after the data up to the next scan are assigned to it. They form a *scan header*. Some of the header information is common to all scans or a group of them. It appears in a so called *file header*. A file header consists of all header lines between a *'#F'* header line indicating file header beginning and a *'#S'* line, where a *scan* starts ( there can be also two file headers one after another. In this case the first file header is ignored ).
*SfHeader()* retrieves all *header lines* for a scan which match a certain pattern.
*SfFileHeader()* works in much the same way on the *file header* corresponding to a scan index.
Both functions add a *'#'* to the input pattern and compare it with the beginning of lines in the scan or file header. For example if the input pattern consists of *'C'*, the functions look for and return all lines in the *scan* or *file header* which begin with *'#C'*. Because of this fact *SfHeader()* and *SfFileHeader()* can also be used to retrieve user defined header lines. It is possible to get all file or scan header lines. The pattern must be set to *'\0'* or *(char *)NULL* then. Some information can be retrieved from the header lines straightforward with specialized library functions:
*SfUser()* allows to get the unix user name of the file creator.
*SfTitle()* returns a string which in most cases corresponds to the used SPEC version.
*SfEpoch()* retrieves the time that has elapsed since a certain point, the epoch.
*SfFileDate()* returns the creation date of the file,
*SfDate()* reads scan time and date.
*SfCommand()* returns the SPEC command that originated the scan, e.g."ascan th 2 10 3 1".
*SfHKL()* reads H, K and L value used in crystallography.
*SfGeometry()* returns all geometry lines which can be then interpreted by the user.
*SfNoHeaderBefore()* shows how many header lines there are before data lines.
*SfNoColumns()* retrieves number of data columns from the *'#N'* line.

Most values returned by the functions are strings. The memory for them is always allocated by the function and it should be freed by the application. *SfHeader()*, *SfFileHeader()* and *SfGeometry()* allocate space for an array of strings. To recover this space, the **SpecFile** convenience function *freeArrNZ()* can be used.

```
user = SfUser( sf, index, &error );
if ( user == (char *)NULL ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ ) {
                exit( 1 );
        } else {
                error = 0;
        }
} else {
        printf( "User : %S\n", user );
}
```

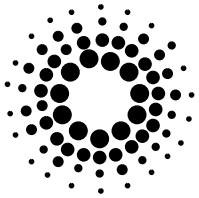
*fig.4*

## 3.4 Accessing Data

The *data* in a scan is organized in columns and rows. A column correspond to a motor position or a counter. Each row represents a data point. *SfData()* returns a 2-D array of double type numbers with all data for a certain scan. A single data line or column can be retrieved with the convenient function *SfDataLine(), SfDataCol()* or *SfDataColByName()*. One parameter or the return value of each of these functions shows how many items there are in the array. It is important to use these values to read the arrays and not the functions *SfNoColumns()* or *SfNoDataLines(),* because they retrieve the number of data lines and columns in a different way explaining possible differences. It could also happen that for some kind of exotic scan data are not regular or will consist of other information than pure numbers. In such cases *SfData()* will probably fail in giving the correct information. *SfDataAsString()* returns an array of strings, each of them containing one raw line in which no data extraction has been done. It is then the user who will analyze the data line.
All functions accessing data automatically allocate the required memory. The programmer is responsible to free this memory when it is not longer needed. The library provides with two functions *freePtr()* and *freeArrNZ()* to help with this task.

```
                    .
                    .
                    .
if ( SfData( sf, index, &data, &data_info, &error ) ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ ) {
                exit( 1 );
        } else {
                error = 0;
        }
} else {
        if ( !data_info[REG] ) {
                printf( "scan data :\n");
                for ( i=0 ; i<data_info[ROW] ; i++ ) {
                        for ( j=0 ; j<data_info[COL] ; j++ ) {
                                printf( "| %lf |", data[i][j] );
                        }
                printf("\n");
        } else {
                printf("Warning! \nData is not regular.");
        }
        freeArrNZ( (void ***)&data, data_info[ROW] );
        free( data_info );
}
                    .
                    .
                    .
```

*fig.5*



## *3.5 Column Labels*

In most cases data in a scan is organized in columns. Each column correspond to a motor position or a counter. Other information like machine current, epoch, and other can appear as columns of data. Each of these columns is assigned with a *label* that identifies it. It is the SPEC motor name, counter, or just a string describing the column.
*SfAllLabels()* returns an array of strings with all the column labels for a scan.
*SfLabel()* returns just one label. For example the program can ask for the label for the first column of data (1) or for the one before the last (-2). Negative values mean starting the counting from the last and going backwards. This is particularly useful because SPEC saves the current active motor and counter in the first and last column respectively. The monitor counts are saved in the column before the last. So if the user decides to change motor or counter that he works with, the program will still succeed in retrieving the relevant information.

```
num_labels = SfAllLabels( sf, index, &labels, &error );
if ( num_labels < 0 ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ ) {
                exit( 1 );
        } else {
                error = 0;
        }
} else {
        for ( i=0 ; i<num_labels ; i++ ) {
                printf(" label [%i] : %s\n", i, labels[i] );
        }
        freeArrNZ( (void ***)&labels, num_labels );
}
```

*fig.6*

## *3.6 Motor Names and Positions*

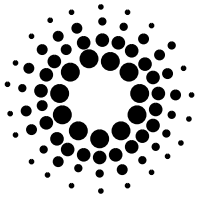
A *scan* means in most cases moving one or a few *motors* through a set of *positions*. The different positions reached by the motors are registered in the scan data. SPEC saves the positions of all motors, including those to which the scan concerns, in the scan header as they were at the moment the scan started. The motor names are stored in the file header in the same order as the positions in the scan header, so that it is possible to identify a motor name and its position.
*SfAllMotors()* returns an array with all motor names, *SfMotor()* returns one motor name.
*SfAllMotorPos(), SfMotorPos()* and *SfMotorPosByName()* are functions which allow to retrieve all or one motor position.
Like *SfLabel(), SfMotor()* and *SfMotorPos()* accept negative values to get motor name or its position starting the count from the last motor. There are two warnings the programmer must be aware of:

- the functions allow multi word motor names. The words of one name are separated then by a single blank and two blanks separate the names. Because of this fact the functions can conflict with older versions of SPEC which does not support multi word names and will separate them with a single blank. This causes the whole line to be read as a single motor name. To correct it, an additional blank must be inserted between all motor names (*'#O'* lines ) in the SPEC data file header.

- The motor configuration can be changed at any point by the user. Even if this is not common during an experiment, motor names or even the number of motors controlled by the SPEC application may change. In this case SPEC does not write a new file header into the file, and therefore motor positions and motor names do not correspond.

```
                    .
                    .
                    .
num_names = SfAllMotors( sf, index, &names, &error );
if ( num_names < 0 ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ ) {
                exit( 1 );
        } else {
                error = 0;
        }
}

num_positions = SfAllMotorPos( sf, index, &positions, &error );
if ( num_positions < 0 ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ ) {
                exit( 1 );
        } else {
                error = 0;
        }
}

if ( !num_names && !num_positions ) {
        printf(" Cannot find motor names and their positions !\n");
} else {
        if ( num_names > num_positions ) num_items = num_names;
        else num_items = num_positions;

        for ( i=0 ; i<num_items ; i++ ) {
                if ( i<num_names ) printf(" motor %s was ", names[i] );
                else printf(" motor ??? was ");

                if ( i<num_positions ) printf("at position %lf :\n", positions[i] );
                else printf(" at unknown position.\n");
        }

        freeArrNZ( void ***)&names, num_names );
        freeArrNZ( (void ***)&positions, num_positions );
}
```

*fig.7*



## *3.7 Writing Selected Scans*

A SPEC file contains often a large number of scans. Many of these are done for alignment or calibration purposes. After an experiment only a fraction of all scans in the file are interesting for data analysis. The **SpecFile Library** provides features to maintain a list of scans, that then can be extracted and written into a separate file. Scans, together with the associated file headers, are copied in the new file without any changes. First, *SfoInit()* must be called to get a pointer to a special list (*SpecFileOut*). This output list will keep the indexes of selected scans. To add a scan or several scans to the output list one of following functions can be used:

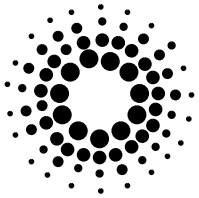
*SfoSelectOne()* adds one scan index to the list;
*SfoSelect()* adds scan indexes from a zero terminated list to the output list;
*SfoSelectRange()* adds a range of scan indexes to the output list.
*SfoSelectAll()* fills the output list with indexes of all scans in the file.

The opposite functions, which remove scan indexes from the output list are:
*SfoRemoveOne()* removes one scan index from the list;
*SfoRemove()* removes scan indexes included in a zero terminated list from the output list;
*SfoRemoveRange()* removes a range of scans from the output list.
*SfoRemoveAll()* empties the output list.

A certain scan will appear in the output list only once even if the user tries to select it several times.
The function *SfoWrite()* copies scans whose indexes are in the output list to another file.
It is the responsibility of the programmer to free the memory allocated for the *SpecFileOut* structure after copying the scans. The appropriate function is *SfoClose()*.

```
sf = SfOpen( file_name, &error );
        .
        .
sfo = SfoInit( sf, &error );
if ( sfo == (SpecFileOut *)NULL ) {
        printf("%s\n", SfError( error ) );
        exit( 1 );
}

SfoSelect...
SfoRemove...

        .
        .
        .
SfoWrite( sfo, output_file_name, &error );
        .
        .
        .
SfoClose( sfo );
SfClose( sf );
```

*fig.8*





## *3.8 Closing, Memory Management*

The **SpecFile Library** makes use of dynamic memory allocation. When this memory is not needed anymore, the programmer should take care of freeing it as it has been explained before. In some cases this memory is allocated for library's own use. It is the case of *SfOpen()* that allocates space for file and scan structures. The programmer must use *SfClose()* to free it. In other cases the memory is allocated for returning arrays ( e.g. *SfData()* ) strings ( *SfTitle()* ) etc. .
The programmer can use standard C routines or the utilities *freePtr()* and *freeArrNZ()* provided by the library to free the memory.
*freePtr()* is identical with C procedure free().
*freeArrNZ()* frees the memory used by a 2-D array and sets then the array pointer to NULL.

## *3.9 Errors and Debugging*

Almost every function uses a parameter to return an *error* code. An error will be reported every time an anomalous condition is found by the function, e.g.: trying to open a non existent file, asking for a negative scan index or many others. When the function executes normally, the contents of the error variable is not affected. Otherwise the library sets an error number that uniquely identifies the anomalous condition ( see Appendix C for full error list ). A simple string associated with the error code can be obtained with *SfError()*.

To help the programmer in the development phase, there exist two functions that switch debugging on and off: *SfDebugOn()*, *SfDebugOff()*. When degugging is on, the library prints messages showing entry points to library functions on *stderr*.

## *3.10 Compiling*

The figure below shows a possible *Makefile* for your programs.

```
CC          = gcc
FILE        = my_file
LFLAGS      = -lspecfile -lm
MAINDIR     = specfile_directory
LIBDIR      = -L$(MAINDIR)/lib
INCLDIRS    = -I$(MAINDIR)/include

$(FILE): $(FILE).c
$(CC) -c $(FILE).c $(INCLDIRS)
$(CC) -o $@ $(FILE).o $(LIBDIR) $(LFLAGS)
```

*fig.9*

# 4.0 Reference

# *SfAllLabels*

## Purpose

A function that reads all labels of scan data columns.

## Synopsis

**long SfAllLabels**( sf, index, labels, error )

```
SpecFile        *sf;
long            index;
char            ***labels;
int             *error;
```

## Description

This function reads all labels of scan data columns from *'#L'* lines in the scan header. It returns the number of labels stored in the array pointed to by *labels*. On failure ***-1*** is returned and the variable *error* contains the code of the error. The associated error string can be accessed by calling *SfError()*. The function allocates memory for the labels. It is important to free it e.g. by calling *freeArrNZ()*. The parameter *sf* is the pointer to the SPEC data file previously opened with *SfOpen()*. *Index* is the scan index created by *SfOpen()*. The index of the first scan is ***1***.
The function allows multi word labels. The words of one label are separated then by a single blank and two blanks separate the labels. Because of this fact the function can come in conflicts with older versions of SPEC which do not support multi word labels and separate the labels with a single blank. This causes that the whole line is read as a single label. To correct it, insert an additional blank between all labels (*'#L'* lines ) in the SPEC data file.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LINE_EMPTY.

## Related Information

*SfLabel(), SfOpen(), SfError(), freeArrNZ().*

**SfAllLabels**

# Example

```
/*******************************************************************************
* This program reads and displays all labels of scan data columns.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   char              *file_name;
   long              index ;
   int               i, error=0;
   char              **labels;
   long              num_labels;

                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   num_labels = SfAllLabels( sf, index, &labels, &error );
   if ( num_labels < 0 ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else {
      for ( i=0 ; i<num_labels ; i++ )
         printf(" label [%i] : %s\n", i+1, labels[i] );
      freeArrNZ( (void ***)&labels, num_labels );
   }

                .
                .
                .

   SfClose( sf );
}
```

# *SfAllMotors*

## Purpose

A function that reads all motor names.

## Synopsis

**long SfAllMotors**( sf, index, names, error )

```
SpecFile        *sf;
long            index;
char            ***names;
int             *error;
```

## Description

If successful, the function returns the number of read motor names. The variable *names* points to the array with motor names. The memory allocated for this array can be freed by calling freeArrNZ(). On failure, the function returns ***-1*** and sets *error* appropriately. The associated error string can be accessed by calling *SfError()*.The parameter *sf* is the pointer to the SPEC data file previously opened with *SfOpen()*. *Index* is the scan index created by *SfOpen()*. The index of the first scan is ***1***.
The function allows multi word motor names. The words of one name are separated then by a single blank and two blanks separate the names. Because of this fact the function can come in conflicts with older versions of SPEC which do not support multi word names and separate the names with a single blank. This causes that the whole line is read as a single motor name. To correct it, insert an additional blank between all motor names (*'#O'* lines ) in the SPEC data file.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_HEADER_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LINE_EMPTY.

## Related Information

*SfMotor(), SfMotorPos(), SfAllMotorPos(), SfMotorPosByName(), SfOpen(), SfError(), freeArrNZ().*

**SfAllMotors**

# Example

```
/*****************************************************************************
* This program reads and displays all motor names.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          index ;
   int           i, error=0;
   char          **names;
   long          num_names;

            .
            .
            .

   sf = SfOpen( file_name, &error );

            .
            .
            .

   num_names = SfAllMotors( sf, index, &names, &error );
   if ( num_names < 0 ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
             exit( 1 );
        else
             error = 0;
   } else {
        for ( i=0 ; i<num_names ; i++ )
             printf(" motor name [%i] : %s\n", i+1, names[i] );
        freeArrNZ( void ***)&names, num_names );
   }

                .
                .
                .

   SfClose( sf );
}
```

# *SfAllMotorPos*

## Purpose

A function that reads all motor positions

## Synopsis

**long SfAllMotorPos**( sf, index, pos, error )

```
SpecFile        *sf;
long            index;
double          **pos;
int             *error;
```

## Description

If successful, the function returns the number of read motor positions. The variable *pos* points then to the array with motor positions. The application is responsible for freeing the memory allocated for this array. On failure, the function returns ***-1*** and sets *error* appropriately. The associated error string can be accessed by calling *SfError()*. The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*. The variable *index* is the scan index created by *SfOpen()*. The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LINE_EMPTY.

## Related Information

*SfMotorPos(), SfMotorPosByName(), SfMotor(), SfAllMotors(), SfOpen(), SfError()*

**SfAllMotorPos**

# Example

```
/*****************************************************************************
* This program reads and displays all motor positions.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile   *sf;
   char       *file_name;
   long       index ;
   int        i, error=0;
   double     *positions;
   long       num_positions;

         .
         .
         .

   sf = SfOpen( file_name, &error );

         .
         .
         .

   num_positions = SfAllMotorPos( sf, index, &positions, &error );
   if ( num_positions < 0 ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else {
         for ( i=0 ; i<num_positions ; i++ )
               printf(" motor %i was at the position %lf\n", i+1, positions[i] );
         free( positions);
   }

         .
         .
         .

   SfClose( sf );
}
```

# *SfClose*

## Purpose

A function that closes the connection to a SPEC data file.

## Synopsis

**int SfClose**( sf )

SpecFile *sf;

## Description

The return value of *SfClose()* is ***0*** , upon the operation is successful and ***-1*** on failure.
The function frees all memory used by the *SpecFile* structure pointed to by *sf* and closes the SPEC data file. Every data file opened with *SfOpen()* should be later closed with *SfClose().*

## Errors

none is returned.

## Related Information

*SfOpen().*

# Example

```
/*****************************************************************************
* This program opens and closes the connection to a SPEC data file
*****************************************************************************/
#include <SpecFile.h>

main( argc, argv )

int         argc;
char        *argv[];
{
   SpecFile             *sf;
   int                  error=0;

    /* Open a file specified by the first command line parameter. */
   if ( argc < 2 ) {
   fprintf( stderr,"Usage: %s file name\n", argv[0] );
   exit( 1 );
    }

   sf = SfOpen( argv[1], &error );

    /* Check if successful. */
   if ( sf == (SpecFile *)NULL ) {
         printf( "%s\n", SfError( error ) );
         exit( 1 );
   } else {
         if ( error ) {
               /* see SfOpen() for the explanation. */
               printf( "Could not read whole file !\n" );
               printf( "Occurred error :%s\n", SfError( error ) );
               error = 0;
         }
   }
   printf("Connection to the file %s successful\n", argv[1] );

  /* Now you can use other library functions, e.g. get number of scans in the file. */
   no_scans = SfScanNo( sf );
   printf("There are %li scans in the file.\n", no_scans );
   .
   .
   .
    /* Close connection. */
   SfClose( sf );

}
```

# *SfCommand*

## Purpose

A function that reads scan command.

## Synopsis

**char *SfCommand**( sf, index, error )

```
SpecFile        *sf;
long            index;
int             *error;
```

## Description

This function returns the rest of the *'#S'* line after the scan number, the scan command.
If the execution of the function fails, the return value will be (char *)NULL. The variable *error* indicates then the reason of failure. *SfError()* returns the associated error string. The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*. The variable *index* is the scan index created by *SfOpen()*. The index of the first scan is ***1***.
The function allocates space for the return value. Freeing this space is caller's responsibility.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND.

## Related Information

*SfOpen(), SfError()*

# Example

```
/*****************************************************************************
* This program displays scan command.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   char              *file_name;
   long              index ;
   int               error=0;
   char              *command;
                  .
                  .
                  .

   sf = SfOpen( file_name, &error );

                  .
                  .
                  .

   command = SfCommand( sf, index, &error );
   if ( command == (char *)NULL ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   }
   printf(" The command of scan %li is : %s\n", index, command );

   free( command );

                  .
                  .
                  .

   SfClose( sf );
}
```

# *SfCondList*

## Purpose

A function that gets conditional list of scan numbers in a file.

## Synopsis

**long SfCondList**( sf, condition, scan_list, error )

```
SpecFile        *sf;
long            condition;
long            **scan_list;
int             *error;
```

## Description

This function creates a conditional list of scan numbers. The return value is the number of items in this list, on success, and ***-1*** if the execution fails. In the second case *error* includes the appropriate error code. The possible conditions for the list are:

| | | |
|---|---|---|
| *ABORTED* | or ***-1*** | : the list contains only numbers of aborted scans. |
| *NOT_ABORTED* | or ***0*** | : the list contains only numbers of not aborted scans. |
| *nn* | | : the list contains numbers of scans with more than *nn* data lines.( *nn* is a long integer number ). |

The function allocates memory for the list. It should be freed by the application.
The first argument, *sf,* is the pointer to the SPEC data file previously opened with *SfOpen()*.
*Scan_list* is the address of the returned list of found scan numbers.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfList(), SfScanNo(), SfOpen, SfError().*

**SfCondList**

# Example

```
/*****************************************************************************
* This program displays a list of scans, which fulfil given condition.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile     *sf;
   char         *file_name;
   long         condition;
   long         *list;
   long         num_scans;
   int          i, error=0;
                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   if ( (num_scans = SfCondList( sf, &list, condition, &error ) == -1 ) {
        printf("%s\n", SfError( error ) );
        exit( 1 );
   }
   if ( !num_scans ) {
        printf( " Could not find such scans ! \n");
   } else {
        if ( !condition )
             for ( i=0 ; i<num_scans ; i++ ) printf(" Scan number %li was not aborted.\n", list[i] );
        if ( condition < 0 )
             for ( i=0 ; i<num_scans ; i++ ) printf(" Scan number %li was aborted.\n", list[i] );}
        else
             for ( i=0 ; i<num_scans ; i++ ) printf(" Scan number %li has more than %li data lines.\n", condition, list[i] );

        free( list );
   }
                .
                .
                .

   SfClose( sf );
}
```

# *SfData*

## Purpose

A function that reads scan data lines and transforms them into an array of doubles.

## Synopsis

**int SfData**( sf, index, data, data_info, error )

```
SpecFile        *sf;
long            index;
double          ***data;
long            **data_info;
int             *error;
```

## Description

On success, the function returns ***0***, otherwise ***-1*** and sets the *error* variable appropriately. After a successful execution the parameter *data* points to a scan data array and *data_info to* an array of three elements :

**data_info[ROW]* - contains the number of data rows;
**data_info[COL]* - contains the number of data columns;
**data_info[REG]* - is ***0*** if data is regular (the number of columns is constant ); ***1*** if data is not regular ( **data_info[COL]* is then the maximal number of columns ).

The application is responsible for freeing the memory provided by this function to store the data and the data info. The data memory can be recovered by calling freeArrNZ().
The first argument, *sf,* is the pointer to the SPEC data file previously opened with *SfOpen()*. The variable *index* is the scan index created by *SfOpen()*. The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND.

## Related Information

*SfDataAsString(), SfDataLine(), SfDataCol(), SfDataColByName(), SfNoDataLines(), SfOpen(), freeArrNZ().*

# Example

```
/*****************************************************************************
* This program displays scan data.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   double        **data;
   long          *data_info;
   long          index;
   int           i, j;
   int           error=0;

                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   if ( SfData( sf, index, &data, &data_info, &error ) ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
             exit( 1 );
        else
             error = 0;
   } else {
        if ( !data_info[REG] ) {
             printf( "scan data :\n");
             for ( i=0 ; i<data_info[ROW] ; i++ ) {
                  for ( j=0 ; j<data_info[COL] ; j++ ) {
                       printf( "| %lf |", data[i][j] );
                  }
                  printf("\n");
             }
        } else {
             printf("Warning! \nData is not regular.");
        }
        freeArrNZ( (void ***)&data, data_info[ROW] );
        free( data_info );
   }

                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfDataAsString*

## Purpose

A function that reads scan data lines.

## Synopsis

**long SfDataAsString**( sf, index, data, error )

```
SpecFile    *sf;
long        index;
char        ***data;
int         *error;
```

## Description

On success, the function returns the number of data lines in the array pointed to by *data,* otherwise it returns ***-1*** and sets the *error* variable appropriately. The associated error string can be accessed by calling *SfError()*.
*SfDataAsString()* reads scan data lines without transforming it in any way. It can be used when the data are not regular ( number of columns not constant ) or when the data consist of other type than pure numbers. The usage of *SfData()* in these two cases will probably fail.
The application is responsible for freeing the memory provided by this function to store the data lines. The memory can be recovered by calling *freeArrNZ()*.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* is the scan index created by *SfOpen()*. The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND.

## Related Information

*SfData(), SfDataLine(), SfDataCol(), SfDataColByName(), SfNoDataLines(), SfOpen(), SfError().*

**SfDataAsString**

# Example

```
/*******************************************************************************
* This program displays scan data lines.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile     *sf;
   char         *file_name;
   char         **data;
   long         num_lines;
   long         index;
   int          i, error=0;

                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   if ( ( num_lines = SfDataAsString( sf, index, &data, &error ) ) == -1 ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
             exit( 1 );
        else
             error = 0;
   } else {
        printf( "scan data lines:\n");
        for ( i=0 ; i<num_lines ; i++ )
             printf( "line [%li] : %s\n", i+1, data[i]);
        freeArrNZ( (void ***)&data, num_lines );
   }
                .
                .
                .

   SfClose( sf );
}
```

# *SfDataCol*

## Purpose

A function that reads one scan data column.

## Synopsis

**long SfDataCol**( sf, index, col, data_col, error )

```
SpecFile        *sf;
long            index;
long            col;
double          **data_col;
int             *error;
```

## Description

This function returns the number of data values in the single data column specified by *col*. Negative values for *col* mean starting the counting from the last column and going backwards. E.g. when *col* is set to ***2***, the function will read the second column, when set to ***-2*** it will read the column before the last.
On failure it returns ***-1*** and sets *error* to the corresponding error code. The data column is pointed to by *data_col*. This function allocates the space necessary to hold the output value. Freeing this space is the caller's responsibility.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*. The variable *index* is the scan index created by *SfOpen()*. The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_COL_NOT_FOUND.

## Related Information

*SfData(), SfDataAsString(), SfDataColByName(), SfDataLine(), SfNoDataLines()*
*SfOpen(), SfError().*

# Example

```
/*****************************************************************************
* This program displays one scan data column.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   double        *data_col;
   long          col;
   long          num_values;
   long          index;
   int           i, error=0;

                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   if ( ( num_values = SfDataCol( sf, index, col, &data_col, &error ) ) == -1 ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else {
         printf( "Column %li:\n", col );
         for ( i=0 ; i<num_values ; i++ )
               printf( "%lf\n", data_col[i]);
         free( data_col );
   }

                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfDataColByName*

## Purpose

A function that reads one scan data column.

## Synopsis

**long SfDataColByName**( sf, index, label, data_col, error)

```
SpecFile    *sf;
long        index;
char        *label;
double      **data_col;
int         *error;
```

## Description

This function returns the number of data values in the single data column specified by column *label*. On failure it returns ***-1*** and sets *error* to the corresponding error code. The data column is pointed to by *data_col*. This function allocates the space necessary to hold the output value. Freeing this space is the caller's responsibility.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LINE_EMPTY,
SF_ERR_COL_NOT_FOUND.

## Related Information

*SfData(), SfDataAsString(), SfDataCol(), SfDataLine(), SfNoDataLines()*
*SfOpen(), SfError().*

**SfDataColByName**

# Example

```
/*****************************************************************************
* This program displays one scan data column.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   char          *label;
   double        *data_col;
   long          num_values;
   long          index;
   int           i, error=0;

                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   if ( ( num_values = SfDataColByName( sf, index, label, &data_col, &error ) ) == -1 ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else {
         printf( "Column %s:\n", label );
         for ( i=0 ; i<num_values ; i++ )
               printf( "%lf\n", data_col[i]);
         free( data_col );
   }
                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfDataLine*

## Purpose

A function that reads one data line.

## Synopsis

**long SfDataLine**( sf, index, line, data_line, error )

```
SpecFile    *sf;
long        index;
long        line;
double      **data_line;
int         *error;
```

## Description

This function returns the number of data values in the single data line specified by *line* number. If *line* is negative, line counting will begin from the last and will go backwards.
E.g. when *line* is set to ***2*** , the function will read the second line, when set to ***-1*** it will read the last one. On failure the function returns ***-1*** and sets *error* to the corresponding error code. The data line is pointed to by *data_line*. This function allocates the space necessary to hold the output value. Freeing this space is the caller's responsibility.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND.

## Related Information

*SfData(), SfDataAsString(), SfDataCol(), SfDataColByName(), SfNoDataLines()*
*SfOpen(), SfError().*

# Example

```
/*****************************************************************************
* This program displays one scan data line.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   double        *data_line;
   long          line;
   long          num_values;
   long          index;
   int           i, error=0;

                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   if ( ( num_values = SfDataLine( sf, index, line, &data_line, &error ) ) == -1 ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else {
         printf( "Line %li:\n", line );
         for ( i=0 ; i<num_values ; i++ )
               printf( "| %lf |", data_line[i]);
         printf("\n");
         free( data_line );
   }
                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfDate*

## Purpose

A function that returns scan date.

## Synopsis

**char *SfDate**( sf, index, error )

```
SpecFile        *sf;
long            index;
int             *error;
```

## Description

This function returns date from the scan header line ( *'#D'* line ). If an error occurs inside the function, (char *)NULL will be returned and the variable *error* will contain the appropriate error code. The memory used by the returned string should be freed by the application.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND.

## Related Information

*SfFileDate(), SfOpen(), SfError().*

# Example

```
/*****************************************************************************
* This program displays scan date.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   char          *date;
   long          index;
   int           error=0;

                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   if ( (date = SfDate( sf, index, &error ) ) == (char *)NULL ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else {
         printf("Date of scan %li : %s \n", index, date );
         free( date );
   }

                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfDebugOff, SfDebugOn*

## Purpose

Functions to switch the debugging on or off.

## Synopsis

**void SfDebugOff()**

**void SfDebugOn()**

## Description

After *SfDebugOn()* has been called, the name of every entered library function ( internal functions too ) is printed on **stderr**.
This allows to follow the program flow and to detect possible bugs.

*SfDebugOff()* suspends the action of *SfDebugOn()*.

## Errors

none is returned.

## Related Information

**SfDebugOff,SfDebugOn**

# Example

```
/*****************************************************************************
* This program displays the name of every entered library function.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;

                 .
                 .
                 .

   SfDebugOn();
   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   SfClose( sf );
   SfDebugOff();
}
```

# *SfEpoch*

## Purpose

A function that returns the epoch of the file.

## Synopsis

**long SfEpoch**( sf, index, error )

```
SpecFile        *sf;
long            index;
int              *error;
```

## Description

This function returns epoch from the *'#E'* line in the last file header. If an error occurs inside the function ***-1*** will be returned and the variable *error* will contain the appropriate error code. Use *SfError()* to get the associate error message .
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_HEADER_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND.

## Related Information

*SfOpen(), SfError().*

# Example

```
/*******************************************************************************
* This program displays scan epoch.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          index;
   long          epoch;
   int           error=0;

                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   if ( (epoch = SfEpoch( sf, index, &error ) == -1) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else
         printf("Epoch of scan %li : %li \n", index, epoch);

                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfError*

## Purpose

A function that returns error message.

## Synopsis

**char *SfError( error)**

**int error;**

## Description

This function returns a string which describes the error reason specified by *error* code.
It does not allocate any space, so do not worry about the memory.

The variable error in your program should be set to ***0*** after every error checking operation following the library functions.

## Errors

none is returned.

## Related Information

*Appendix E*

# Example

```
*******************************************************************************
* This program shows how to deal with library errors.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile     *sf;
   char         *file_name;
   long         index;
   long         epoch;
   int          error=0;

                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   if ( (epoch = SfEpoch( sf, index, &error ) == -1) {
        switch ( error ) {
        case SF_ERR_LINE_NOT_FOUND:
                        printf("Cannot find any #E line in the file header !\n");
                        error = 0;
                        break;
        case SF_ERR_HEADER_NOT_FOUND :
                        printf("This scan has no associated file header !\n");
                        error = 0;
                        break;
        case SF_ERR_SCAN_NOT_FOUND:
                        printf("There is no scan with such index : %li\n", index );
                        error = 0;
                        break;
        default :
                        printf("Fatal error : %s\n", SfError( error ) );
                        exit( 1 );
        }
   } else
        printf("Epoch of scan no. %li : %s \n", index, epoch);

                .
                .
                .

   SfClose( sf );
}
```

# *SfFileDate*

## Purpose

A function that returns file date.

## Synopsis

**char *SfFileDate**( sf, index, error )

```
SpecFile        *sf;
long            index;
int             *error;
```

## Description

This function returns date from the file header ( *'#D'* line ). If an error occurs inside the function, (char *)NULL will be returned and the variable *error* will contain the appropriate error code. The memory used by the returned string should be freed by the application.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_HEADER_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LINE_EMPTY.

## Related Information

*SfDate(), SfOpen(), SfError().*

**SfFileDate**

# Example

```
/*****************************************************************************
* This program displays file date.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   char          *file_date;
   long          index;
   int           error=0;

                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   if ( (file_date = SfFileDate( sf, index, &error ) ) == (char *)NULL ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else {
         printf("File date :%s \n", file_date );
         free( file_date );
   }
                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfFileHeader*

## Purpose

A function that reads file header lines.

## Synopsis

**long SfFileHeader**( sf, index, string, lines, error )

```
SpecFile        *sf;
long            index;
char            *string;
char            ***lines;
int             *error;
```

## Description

This function allows to read all or only specified file header lines. The line type can be specified by setting *string* pattern appropriately. A '*#*' is added automatically. E.g. to read all comment lines from the last file header set *string* to "C\0". To read all file header lines set *string* to "\0" or (char *)NULL. The function returns the number of found file header lines. If an error has occurred it returns ***-1*** and the variable *error* contains the error code. The associated error message can be accessed by calling *SfError()*. The function allocates memory for the file header lines which should be freed by the application ( e.g. using *freeArrNZ()* ).
The first argument, *sf,* is the pointer to the previously with *SfOpen()* opened SPEC data file. The parameter *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*. The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_HEADER_NOT_FOUND,
SF_ERR_SCAN_NOT_FOUND.

## Related Information

*SfHeader(), SfOpen(), SfError(), freeArrNZ().*

**SfFileHeader**

# Example

```
/*******************************************************************************
* This program displays all file header lines.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile     *sf;
   char         *file_name;
   char         string[1] = "\0";
   char         **lines;
   long         num_lines;
   long         index;
   int          i, error=0;

                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   if ( ( num_lines = SfFileHeader( sf, index, string, &lines, &error ) ) == -1 ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
             exit( 1 );
        else
             error = 0;
   } else {
        printf( "File header lines:\n");
        for ( i=0 ; i<num_lines ; i++ )
             printf( "line [%li] : %s\n", i+1, lines[i] );
        freeArrNZ( (void ***)&lines, num_lines );
   }
                .
                .
                .

   SfClose( sf );
}
```

# *SfGeometry*

## Purpose

A function that reads all scan header lines with geometry informations.

## Synopsis

**long SfGeometry**( sf, index, lines, error )

```
SpecFile        *sf;
long            index;
char            ***lines;
int             *error;
```

## Description

This function reads all scan *'#G'* lines which contain the geometry values and returns the number of read lines. If an error occurs inside the function ***-1*** will be returned and the variable *error* will contain the appropriate error code. The associated error message can be accessed by calling *SfError().*
The function allocates memory for the geometry lines. It can be recovered by calling *freeArrNZ()*. The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen().* The variable *index* represents the scan index created by *SfOpen()*. The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND.

## Related Information

*freeArrNZ(), SfHeader(), SfOpen(), SfError().*

# Example

```
/*****************************************************************************
* This program displays geometry lines.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile     *sf;
   char         *file_name;
   char         **lines;
   long         num_lines;
   long         index;
   int          i, error=0;

                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   if ( ( num_lines = SfGeometry( sf, index, &lines, &error ) ) == -1 ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
             exit( 1 );
        else
             error = 0;
   } else {
        printf( "Scan geometry lines :\n");
        for ( i=0 ; i<num_lines ; i++ )
             printf( "line [%li] : %s\n", i+1, lines[i]);
        freeArrNZ( (void ***)&lines, num_lines );
   }
                .
                .
                .

   SfClose( sf );
}
```

# *SfHKL*

## Purpose

A function that returns the H,K and L values.

## Synopsis

**double *SfHKL**( sf, index, error )

```
SpecFile        *sf;
long            index;
int             *error;
```

## Description

The return value of this function is an array with *H, K* and *L* values. They are retrieved from the *'#Q'* line of the scan haeder. Instead of an array index, the letters H, K and L can be used when reading the array ( e.g. array[H] contains the H value ). In case of an error (double *)NULL will be returned and the variable *error* will contain the appropriate error code. This function allocates memory for the returned array. It should be freed by the application.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LINE_EMPTY.

## Related Information

*SfOpen(), SfError().*

# Example

```
/*****************************************************************************
* This program displays H, K and L values.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile     *sf;
   char         *file_name;
   double       *hkl_array;
   long         index;
   int          error=0;

                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   if ( ( hkl_array = SfHKL( sf, index, &error ) ) == (double *)NULL ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
             exit( 1 );
        else
             error = 0;
   } else {
        printf( "Scan index : %li\n", index );
        printf( "\tH value :\n", hkl_array[H] );
        printf( "\tK value :\n", hkl_array[K] );
        printf( "\tL value :\n", hkl_array[L] );

        free( hkl_array );
   }
                .
                .
                .

   SfClose( sf );
}
```

# *SfHeader*

## Purpose

A function that reads scan header lines.

## Synopsis

**long SfHeader**( sf, index, string, lines, error )

```
SpecFile        *sf;
long            index;
char            *string;
char            ***lines;
int             *error;
```

## Description

This function allows to read all or only specified scan header lines. The line type can be specified by setting *string pattern* appropriately .A '*#*' is added automatically. E.g. to read all geometry lines of the scan set string to "G\0". To read all scan header lines set string to "\0" or (char *)NULL. The function returns the number of found scan header lines. If an error has occurred it returns ***-1*** and the variable *error* contains the error code. The associated error message can be accessed by calling *SfError()*. The function allocates memory for the header lines which should be freed by the application ( e.g. using *freeArrNZ()* ).
The parameter *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND.

## Related Information

*SfNoHeaderBefore(), SfFileHeader(), SfOpen(), SfError(), freeArrNZ().*

**SfHeader**

# Example

```
/*****************************************************************************
* This program displays all scan geometry lines.( #G )
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile     *sf;
   char         *file_name;
   char         string[2] = "G\0";
   char         **lines;
   long         num_lines;
   long         index;
   int          i, error=0;

                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   if ( ( num_lines = SfHeader( sf, index, string, &lines, &error ) ) == -1 ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
             exit( 1 );
        else
             error = 0;
   } else {
        if ( !num_lines ) {
             printf( " no geomerty lines !\n");
        } else {
             printf( "Scan geometry :\n");
             for ( i=0 ; i<num_lines ; i++ )
                  printf( "line [%li] : %s\n", i+1, lines[i]);
             freeArrNZ( (void ***)&lines, num_lines );
        }
   }
                .
                .
                .

   SfClose( sf );
}
```

# *SfIndex*

## Purpose

A function that returns scan index.

## Synopsis

**long SfIndex**( sf, number, order )

```
SpecFile        *sf;
long            number;
long            order;
```

## Description

This function returns scan *index* created by *SfOpen()*, which is used by other SpecFile functions to access scan related informations. The index of the first scan in the file is ***1***. If there is no scan with corresponding *number* and *order*, ***-1*** will be returned.
If there are several scans with identical *number* , the variable *order* helps to distinguish between them. The first found scan with a certain *number* has always the *order* ***1***, the next one with the same *number* the *order* ***2*** and so on.
*Sf* is the pointer to the SPEC data file previously opened with *SfOpen()*.

## Errors

none returned.

## Related Information

*SfNumber(), SfOrder(), SfNumberOrder(), SfOpen().*

# Example

```
/*****************************************************************************
* This program displays scan index.
*****************************************************************************/
#include <SpecFile.h>

long input();

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          number=1;
   long          order;
   long          index;
   int           error=0;

                 .
                 .
                 .
   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   while ( number ) {
         number = input( "Enter scan number : ");
         order = input( "Enter scan order : ");
         printf("\n");

         if ( (index = SfIndex( sf, number, order ) ) == -1) {
               printf("Scan not found ! \n");
         } else {
               printf("index = %li \n", index );
         }
   }
                 .
                 .
                 .

   SfClose( sf );
}

/*****************************************************************************/
long
input( string )

char         *string;
{
   char     buffer[99];

   printf( string );
   fgets( buffer, 99, stdin );
   return( atol( buffer ) );
}
```

# *SfLabel*

## Purpose

A function that returns label of a scan data column.

## Synopsis

**char *SfLabel**( sf, index, column, error )

```
SpecFile        *sf;
long            index;
long            column;
int             *error;
```

## Description

This function returns the label of a scan data column specified by *column* number. If this number is negative, column counting will begin from the last one and will go backwards. E.g. when *column* is set to ***2*** , the function will return the second label, when set to ***-1***, it will return the last one. On failure (char *)NULL is returned and the variable *error* contains the code of the error. The associated error string can be accessed by calling *SfError()*. The function allocates memory for the label. It is important to free it.
*Sf* is the pointer to the SPEC data file previously opened with *SfOpen()*. *Index* is the scan index created by the same function. The index of the first scan is ***1***.

The function allows multi word labels. The words of one label are separated then by a single blank and two blanks separate the labels. Because of this fact the function can come in conflicts with older versions of SPEC which do not support multi word labels and separate the labels with a single blank. This causes that the whole line is read as a single label. To correct it, insert an additional blank between all labels (*'#L'* lines ) in the SPEC data file.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_EMPTY,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LABEL_NOT_FOUND.

## Related Information

*SfAllLabels(), SfOpen(), SfError().*

# Example

```
/*****************************************************************************
* This program displays one column label.
*****************************************************************************/
#include <SpecFile.h>
long input();

main()
{
   SpecFile      *sf;
   char          *file_name;
   char          *label;
   long          index=1;
   long          column;
   int           error=0;
                 .
                 .
                 .
   sf = SfOpen( file_name, &error );
                 .
                 .
                 .
   while ( index ) {
        index = input( "Enter scan index : ");
        column = input( "Enter data column number : ");
        printf("\n");

        if ( (label = SfLabel( sf, index, column, &error) ) == (char *)NULL) {
             printf( "Error : %s\n", SfError( error ) );
             if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
                  exit( 1 );
             else
                  error = 0;
        } else {
             printf("label = %s \n", label);
             free( label );
        }
   }
                 .
                 .
                 .
   SfClose( sf );
}

/*****************************************************************************/
long
input( string )

char          *string;
{
   char       buffer[99];

   printf( string );
   fgets( buffer, 99, stdin );
   return( atol( buffer ) );
}
```

# *SfList*

## Purpose

A function that returns a list of numbers of all scans in the file.

## Synopsis

**long *SfList**( sf, error )

```
SpecFile        *sf;
int             *error;
```

## Description

*SfList()* allows the user to get a list of numbers of all scans in the file. This function allocates memory for the returned list which should be freed by the application.
If there is no enough memory for the list , the function will return (long *)NULL and the variable *error* will be set to SF_ERR_MEMORY_ALLOC.
The number of items in the list can be obtained calling *SfScanNo()*.
The parameter *sf* is the pointer to the SPEC data file previously opened with *SfOpen()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfCondList(), SfScanNo(), SfOpen(), SfError().*

# Example

```
/*****************************************************************************
* This program displays scan index and number of each scan in the file.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          *scan_list;
   int           i, error=0;

                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   if ( ( scan_list = SfList( sf, &error ) ) == (long *)NULL )
         printf("%s\n", SfError( error ) );
         exit( 1 );
   }
   for ( i = 0 ; i<SfScanNo( sf ) ; i++ ) {
         printf( "Scan index : %li, \t\tnumber : %li\n" , i+1, scan_list[i] );
   }
   free( scan_list );

                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfMotor*

## Purpose

A function that returns one motor name.

## Synopsis

**char *SfMotor**( sf, index, number, error )

```
SpecFile        *sf;
long            index;
long            number;
int             *error;
```

## Description

If successful, the function returns the name of the motor specified by *number*. If this number is negative, the counting will begin from the last motor name and will go backwards. E.g. when *number* is set to ***3***, the function will return the name of the third motor, when set to ***-1*** it will return that of the last one. The memory allocated for the return value should be freed by the application. On failure, the function returns (char *)NULL and sets *error* appropriately. The associated error string can be accessed by calling *SfError()*. The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*. The variable *index* represents the scan index created by *SfOpen()*. The index of the first scan is ***1***.
The function allows multi word motor names. The words of one name are separated then by a single blank and two blanks separate the names. Because of this fact the function can come in conflicts with older versions of SPEC which do not support multi word names and separate them with a single blank. This causes that the whole line is read as a single motor name. To correct it, insert an additional blank between all motor names (*#O* lines ) in the SPEC data file.

## Errors

SF_ERR_MEMORY_ALLOC, SF_ERR_FILE_READ, SF_ERR_LINE_EMPTY, SF_ERR_HEADER_NOT_FOUND, SF_ERR_SCAN_NOT_FOUND, SF_ERR_LINE_NOT_FOUND, SF_ERR_MOTOR_NOT_FOUND.

## Related Information

*SfAllMotors(), SfMotorPos(), SfMotorPosByName(), SfAllMotorPos(), SfOpen(), SfError().*

# Example

```
/*****************************************************************************
* This program displays the name of chosen motor.
*****************************************************************************/
#include <SpecFile.h>
long input();

main()
{
   SpecFile      *sf;
   char          *file_name;
   char          *motor_name;
   long          motor_number;
   long          index=1;
   int           error=0;
                 .
                 .
                 .
   sf = SfOpen( file_name, &error );
                 .
                 .
                 .
   while ( index ) {
        index = input( "Enter scan index : ");
        column = input( "Enter motor number : ");
        printf("\n");

        if ( (motor_name = SfMotor( sf, index, motor_number, &error) ) == (char *)NULL) {
             printf( "Error : %s\n", SfError( error ) );
             if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
                  exit( 1 );
             else
                  error = 0;
        } else {
             printf("\t\tMotor name is : %s \n", motor_name );
             free( motor_name );
        }
   }
                 .
                 .
                 .
   SfClose( sf );
}

/*****************************************************************************/
long
input( string )

char          *string;
{
   char       buffer[99];

   printf( string );
   fgets( buffer, 99, stdin );
   return( atol( buffer ) );
}
```

# *SfMotorPos*

## Purpose

A function that returns motor position.

## Synopsis

**double SfMotorPos**( sf, index, number, error )

```
SpecFile        *sf;
long            index;
long            number;
int             *error;
```

## Description

The function returns the position of a motor specified by *number*. If this number is negative, the counting will begin from the last motor and will go backwards. E.g. when *number* is set to ***2***, the function will return the position of the second motor, when set to ***-1*** it will return that of the last one. If a position could not be read, it returns ***+HUGE_VAL*** and sets *error* as needed. The associated error string can be accessed by calling *SfError()*.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LINE_EMPTY,
SF_ERR_POSITION_NOT_FOUND.

## Related Information

*SfMotorPosByName(), SfAllMotorPos(), SfAllMotors(), SfMotor(), SfOpen(), SfError().*

**SfMotorPos**

# Example

```
/*****************************************************************************
* This program displays motor position.
*****************************************************************************/
#include <SpecFile.h>
long input();

main()
{
   SpecFile      *sf;
   char          *file_name;
   double        motor_pos;
   long          motor_number;
   long          index=1;
   int           error=0;
                 .
                 .
                 .
   sf = SfOpen( file_name, &error );
                 .
                 .
                 .
   while ( index ) {
        index = input( "Enter scan index : ");
        column = input( "Enter motor number : ");
        printf("\n");

        if ( (motor_pos = SfMotorPos( sf, index, motor_number, &error) ) == +HUGE_VAL ) {
             printf( "Error : %s\n", SfError( error ) );
             if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
                  exit( 1 );
             else
                  error = 0;
        } else {
             printf("\t\tPosition of the motor %li was : %lf \n", motor_pos );
        }
   }
                 .
                 .
                 .
   SfClose( sf );
}

/*****************************************************************************/
long
input( string )

char          *string;
{
   char       buffer[99];

   printf( string );
   fgets( buffer, 99, stdin );
   return( atol( buffer ) );
}
```

# *SfMotorPosByName*

## Purpose

A function that returns motor position.

## Synopsis

**double SfMotorPosByName**( sf, index, name, error )

```
SpecFile        *sf;
long            index;
char            *name;
int             *error;
```

## Description

The function returns the position of a motor specified by its *name*. If the position could not be read, it returns ***+HUGE_VAL*** and sets *error* as needed. The associated error string can be accessed by calling *SfError()*. The parameter *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*. The variable *index* represents the scan index created by *SfOpen()*. The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LINE_EMPTY,
SF_ERR_POSITION_NOT_FOUND.

## Related Information

*SfMotorPos(), SfAllMotorPos(), SfAllMotors(), SfMotor(), SfOpen(), SfError().*

**SfMotorPosByName**

# Example

```
/*****************************************************************************
* This program displays motor position.
*****************************************************************************/
#include <SpecFile.h>
long input();

main()
{
   SpecFile      *sf;
   char          *file_name;
   char          motor_name[99];
   double        motor_pos;
   long          index=1;
   int           error=0;
                 .
                 .
                 .
   sf = SfOpen( file_name, &error );
                 .
                 .
                 .
   while ( index ) {
        index = input( "Enter scan index : ");
        printf( "Enter motor name : ");
        fgets( motor_name, 99, stdin );
        printf("\n");

        if ( (motor_pos = SfMotorPosByName( sf, index, motor_name, &error) ) == +HUGE_VAL) {
             printf( "Error : %s\n", SfError( error ) );
             if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
                  exit( 1 );
             else
                  error = 0;
        } else {
             printf("\t\tPosition of the motor %s was : %lf \n", motor_name, motor_pos );
        }
   }
                 .
                 .
                 .
   SfClose( sf );
}

/*****************************************************************************/
long
input( string )

char          *string;
{
   char       buffer[99];

   printf( string );
   fgets( buffer, 99, stdin );
   return( atol( buffer ) );
}
```

# *SfNoColumns*

## Purpose

A function that returns number of scan data columns.

## Synopsis

**long SfNoColumns**( sf, index, error )

```
SpecFile        *sf;
long            index;
int             *error;
```

## Description

This function returns number of data columns in the scan ( from the *'#N'* line !). On failure ***-1*** is returned and the variable *error* contains the appropriate error code. The retrieved number of columns does not always correspond to the real value and should not be used in functions dealing with scan data. Use *data_info[COL]* instead ( see *SfData()* ).
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND.

## Related Information

*SfNoDataLines(), SfData(), SfOpen(), SfError().*

**SfNoColumns**

# Example

```
/*****************************************************************************
* This program displays number of data columns in the scan.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          index ;
   long          num_columns;
   int           error=0;
                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   num_columns = SfNoColumns( sf, index, &error );
   if ( num_columns == -1 ) {
        printf("%s\n", SfError( error ) );
             if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
                  exit( 1 );
             else
                  error = 0;
   } else
        printf(" Number of data columns in this scan is : %li\n", num_columns );

                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfNoDataLines*

## Purpose

A function that returns number of scan data lines.

## Synopsis

**long SfNoDataLines**( sf, index, error )

```
SpecFile        *sf;
long            index;
int             *error;
```

## Description

This function returns the number of scan data lines. If the scan cannot be found the return value is ***-1*** , and *error* is set to SF_ERR_SCAN_NOT_FOUND.
The returned value can differ from the real number of lines in case of aborted scans( ***+1*** ). Therefore it should not be used when dealing with scan data. Use *data_info[ROW]* instead ( see *SfData()* ).
The parameter *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*. The variable *index* represents the scan index created by *SfOpen()*. The index of the first scan is 1.

## Errors

SF_ERR_SCAN_NOT_FOUND.

## Related Information

*SfNoColumns(), SfData(), SfOpen(), SfError().*

**SfNoDataLines**

# Example

```
/*******************************************************************************
* This program displays number of data lines in the scan.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          index ;
   long          num_lines;
   int           error=0;

                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   num_lines = SfNoDataLines( sf, index, &error );
   if ( num_lines == -1 ) {
         printf("%s\n", SfError( error ) );
   } else
         printf(" Number of data lines in this scan is : %li\n", num_lines );

                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfNoHeaderBefore*

## Purpose

A function that returns the number of scan header lines before data.

## Synopsis

**long SfNoHeaderBefore**( sf, index, error )

```
SpecFile        *sf;
long            index;
int             *error;
```

## Description

The return value of this function is the number of scan header lines before scan data block. It is at least ***1*** , because there is always a *'#S*' line indicating a scan. If the scan cannot be found the function will return ***-1*** and *error* will be set to SF_ERR_SCAN_NOT_FOUND.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_SCAN_NOT_FOUND.

## Related Information

*SfHeader(), SfOpen(), SfError().*

**SfNoHeaderBefore**

# Example

```
/*******************************************************************************
* This program displays the first scan header line after data.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile     *sf;
   char         *file_name;
   char         string[2] = "\0";
   char         **lines;
   long         num_lines;
   long         num_lines_before;
   long         index;
   int          error=0;

                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   if ( ( num_lines = SfHeader( sf, index, string, &lines, &error ) ) == -1 ) {
        printf("%s\n", SfError( error ) );
        if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
             exit( 1 );
        else
             error = 0;
   } else {
        num_lines_before = SfNoHeaderBefore( sf, index, &error );

        if ( num_lines == num_lines_before ) {
             printf( "No header lines after data !\n");
        } else {
             printf( "First header line after data : %s\n", lines[num_lines_before] );
        }
        freeArrNZ( (void ***)&lines, num_lines );
   }

                .
                .
                .

   SfClose( sf );
}
```

# *SfNumber*

## Purpose

A function that returns scan number.

## Synopsis

**long SfNumber**( sf, index )

```
SpecFile        *sf;
long            index;
```

## Description

This function returns scan number. If the scan cannot be found, ***-1*** will be returned.
The parameter *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
*Index* represents the scan index created by the same function. The index of the first scan is ***1***.

## Errors

none is returned.

## Related Information

*SfIndex(), SfOrder(), SfNumberOrder(), SfOpen().*

**SfNumber**

# Example

```
/*****************************************************************************
* This program displays scan index and number.
*****************************************************************************/
#include <SpecFile.h>

long input();

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          number;
   long          index=1;
   int           error=0;

                 .
                 .
                 .
   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   while ( index ) {
         index = input( "Enter scan index : ");
         printf("\n");

         if ( (number = SfNumber( sf, index) ) == -1) {
               printf("Scan not found ! \n");
         } else {
               printf("Scan index = %li , \t\t number = %li \n", index, number );
         }
   }
                 .
                 .
                 .

   SfClose( sf );
}

/*****************************************************************************/
long
input( string )

char          *string;
{
   char       buffer[99];

   printf( string );
   fgets( buffer, 99, stdin );
   return( atol( buffer ) );
}
```

# *SfNumberOrder*

## Purpose

A function that returns scan number and order.

## Synopsis

**int SfNumberOrder**( sf, index, number, order )

```
SpecFile        *sf;
long            index;
long            *number;
long            *order;
```

## Description

This function gets scan number and order. If the operation is successful ( scan found ) ***0*** will be returned, otherwise ***-1***.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
*Index* represents the scan index created by the same function. The index of the first scan is ***1***.

## Errors

none is returned.

## Related Information

*SfIndex(), SfNumber(), SfOrder(), SfOpen().*

**SfNumberOrder**

# Example

```
/*****************************************************************************
* This program displays scan index, number and order.
*****************************************************************************/
#include <SpecFile.h>

long input();

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          number;
   long          order;
   long          index=1;
   int           error=0;

                 .
                 .
                 .
   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   while ( index ) {
         index = input( "Enter scan index : ");
         printf("\n");

         if ( SfNumberOrder( sf, index, &number, &order ) ) {
               printf("Scan not found ! \n");
         } else {
               printf("Scan index = %li , \t\t number = %li , \t\t order = %li\n", index, number , order );
         }
   }
                 .
                 .
                 .

   SfClose( sf );
}

/*****************************************************************************/
long
input( string )

char          *string;
{
   char       buffer[99];

   printf( string );
   fgets( buffer, 99, stdin );
   return( atol( buffer ) );
}
```

# *SfOpen*

## Purpose

A function that opens connection to a SPEC data file.

## Synopsis

**SpecFile *SfOpen**( name, error )

```
char            *name;
int             *error;
```

## Description

This function opens the file named by *name* and returns a pointer to a structure associated with it and necessary for other access to **Specfile Library** functions. *SfOpen()* initializes the SpecFile structure and goes through the whole file to create an index with scan numbers and other useful information. If the open is not successful a NULL pointer is returned and the variable *error* is initialized with an error code. A string associated with this error code can be obtained by calling *SfError()*.
If only a part of the file could be read in consequence of file read or memory problems, a valid SpecFile pointer will be returned and error will show either SF_ERR_FILE_READ or SF_ERR_MEMORY_ALLOC. The user can decide what to do in this case.

## Errors

SF_ERR_FILE_OPEN,
SF_ERR_FILE_READ,
SF_ERR_MEMORY_ALLOC.

## Related Information

*SfClose(), SfError().*

**SfOpen**

# Example

```
/*****************************************************************************
* This program opens and closes the connection to a SPEC data file
*****************************************************************************/
#include <SpecFile.h>

main( argc, argv )

int             argc;
char            *argv[];
{
   SpecFile     *sf;
   char         answer[2];
   long         no_scans;
   int          error=0;


   /* Open a file specified by the first command line parameter. */
   if ( argc < 2 ) {
        fprintf( stderr,"Usage: %s file name\n", argv[0] );
        exit( 1 );
   }

   sf = SfOpen( argv[1], &error );
   if ( sf == (SpecFile *)NULL ) {
        printf( "%s\n", SfError( error ) );
        exit( 1 );
   } else {
        if ( error == SF_ERR_MEMORY_ALLOC ) {
             printf( "Not enough memory !\n" );
             exit( 1 );
        }
        if ( error == SF_ERR_FILE_READ ) {
             printf( "File read error occurred !\n Check the file storage medium !\n");
             printf("\t\t\t Continue ? [ y/n ] :");
             fgets( answer, 2, stdin );
             if ( *answer != 'y' ) exit (1 );
        }
        error = 0;
   }

    /* Now you can use other functions, e.g. get number of scans in the file. */
   no_scans = SfScanNo( sf );
   printf("There are %li scans in the file.\n", no_scans );

                .
                .
                .

   /* Close the connection */
   SfClose( sf );
}
```

# *SfOrder*

## Purpose

A function that returns scan order.

## Synopsis

**long SfOrder**( sf, index )

```
SpecFile        *sf;
long            index;
```

## Description

This function returns scan order. If the scan cannot be found the return value will be ***-1***.
The parameter *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
*Index* represents the scan index created by the same function. The index of the first scan is ***1***.

## Errors

none returned.

## Related Information

*SfIndex(), SfNumber(), SfNumberOrder(), SfOpen().*

# Example

```
/*****************************************************************************
* This program displays scan index, number and order.
*****************************************************************************/
#include <SpecFile.h>

long input();

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          number;
   long          order;
   long          index=1;
   int           error=0;

                 .
                 .
                 .
   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   while ( index ) {
         index = input( "Enter scan index : ");
         printf("\n");

         if ( (number = SfNumber( sf, index) ) == -1) {
               printf("Scan not found ! \n");
         } else {
               printf("Scan index = %li , \t\t number = %li , \t\t order = %li\n", index, number , SfOrder( sf, index ) );
         }
   }
                 .
                 .
                 .

   SfClose( sf );
}

/*****************************************************************************/
long
input( string )

char          *string;
{
   char       buffer[99];

   printf( string );
   fgets( buffer, 99, stdin );
   return( atol( buffer ) );
}
```

# *SfScanNo*

## Purpose

A function that returns number of scans in the file.

## Synopsis

**long SfScanNo**( sf )

SpecFile *sf;

## Description

The return value of this function is the number of scans in the SPEC data file associated with *sf. Sf* is returned by *SfOpen()*.

## Errors

none returned.

## Related Information

*SfList(), SfCondList(), SfOpen().*

**SfScanNo**

# Example

```
/*******************************************************************************
* This program displays number of scans in the file.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile     *sf;
   char         *file_name;
   long         index ;
   int          error=0;
                .
                .
                .

   sf = SfOpen( file_name, &error );

                .
                .
                .

   printf("There are %li scans in file %s.\n", SfScanNo( sf ), file_name );

                .
                .
                .

   SfClose( sf );
}
```

# *SfTitle*

## Purpose

A function that returns SPEC title.

## Synopsis

**char *SfTitle**( sf, index, error )

```
SpecFile        *sf;
long            index;
int             *error;
```

## Description

This function returns SPEC title from the first comment line in the last file header. If an error occurs (char *)NULL will be returned and the variable *error* will contain the appropriate error code. The associated error string can be obtained by calling *SfError()*.
The memory used by the returned title should be freed by the application.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_HEADER_NOT_FOUND,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_LINE_EMPTY.

## Related Information

*SfOpen(), SfError().*

# Example

```
/*******************************************************************************
* This program displays SPEC title.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          index ;
   int           error=0;
   char          *title;
                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   title = SfTitle( sf, index, &error );
   if ( title== (char *)NULL ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else {
         printf(" The title is : %s\n", title );
         free( title );
   }

                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfUser*

## Purpose

A function that returns SPEC user.

## Synopsis

**char * SfUser**( sf, index, error )

```
SpecFile        *sf;
long            index;
int             *error;
```

## Description

This function returns a string which contains the SPEC user information ( from the first comment line (*'#C'*) in the last file header ). If an error occurs inside the function (char *)NULL will be returned and the variable *error* will contain the appropriate error code. The associated error string can be obtained by calling *SfError()*.
This function allocates memory for the SPEC user. The application should free this memory.
The first argument, *sf*, is the pointer to the SPEC data file previously opened with *SfOpen()*.
The variable *index* represents the scan index created by *SfOpen()*.
The index of the first scan is ***1***.

## Errors

SF_ERR_MEMORY_ALLOC,
SF_ERR_FILE_READ,
SF_ERR_HEADER_NOT_FOUND,
SF_ERR_SCAN_NOT_FOUND,
SF_ERR_LINE_NOT_FOUND,
SF_ERR_USER_NOT_FOUND.

## Related Information

*SfOpen(), SfError().*

# Example

```
/*****************************************************************************
* This program displays SPEC user.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile      *sf;
   char          *file_name;
   long          index ;
   int           error=0
   char          *user;
                 .
                 .
                 .

   sf = SfOpen( file_name, &error );

                 .
                 .
                 .

   user = SfUser( sf, index, &error );
   if ( user== (char *)NULL ) {
         printf("%s\n", SfError( error ) );
         if ( error == SF_ERR_MEMORY_ALLOC || error == SF_ERR_FILE_READ )
               exit( 1 );
         else
               error = 0;
   } else {
         printf(" The user is : %s\n", user );
         free( user );
   }
                 .
                 .
                 .

   SfClose( sf );
}
```

# *SfoClose*

## Purpose

A function that closes the output structure.

## Synopsis

**void SfoClose**( sfo )

SpecFileOut *sfo;

## Description

*SfoClose()* closes the SpecFileOut structure pointed to by *sfo* and frees all memory used by it. This function should be used after all write and related operations.

## Errors

none returned.

## Related Information

*SfoInit(), SfoWrite().*

# Example

```
/*****************************************************************************
* This program shows how and where to initialize and close the output structure.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   SpecFileOut       *sfo;
   char              *file_name;
   int               error=0;
                     .
                     .
                     .

   sf = SfOpen( file_name, &error );
   /* Check here if open successful*/

   sfo = SfoInit( sf, &error );
   /* Check here if init successful*/

                     .
                     .
                     .

   /* Here, you can use all library read and write functions */

                     .
                     .
                     .

   SfoClose( sfo );
   SfClose( sf );
}
```

# *SfoGetList*

## Purpose

A function that gets the list of scans in the output list.

## Synopsis

**long SfoGetList**( sfo, list, error )

```
SpecFileOut     *sfo;
long            **list;
int             *error;
```

## Description

This function makes a copy of the list of selected scans ( see *SfoSelect(), SfoRemove()* functions ). It returns the number of scans in this list. If there are no selected scans, ***0*** will be returned and the list will be set to (long *) NULL. In case of an error, the function will return ***-1*** and *error* will show the reason of the failure. The function allocates memory for the output value. The caller is responsible for freeing it.
*Sfo* is the pointer to the output structure initialized with *SfoInit()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfoWrite(), SfoSelect(), SfoRemove(), SfoOpen().*

**SfoGetList**

# Example

```
/*******************************************************************************
* This program shows how to get a list of selected scans.
*******************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   SpecFileOut       *sfo;
   char              *file_name;
   char              *output_file_name;
   long              *list;
   long              num_selected;
   int               i, error=0;
                     .
                     .
                     .

   sf = SfOpen( file_name, &error );
   /* Check here if open successful*/

   sfo = SfoInit( sf, &error );
   /* Check here if init successful*/

                     .
                     .
                     .

   SfoSelect( sfo, index_list, &error );

                     .
                     .
                     .

   num_selected = SfoGetList( sfo, &list, &error );
   if ( num_selected < 0 ) {
         printf( "Error : %s\n", SfError( error ) );
         exit( 1 );
   }
   for ( i =0 ; i<num_selected ; i++ )
         printf("Index of selected scan %li is %li \n", i+1, list[i] );
   free( list );

                     .
                     .
                     .
   SfoWrite( sfo, output_file_name, &error );
                     .
                     .
                     .

   SfoClose( sfo );
   SfClose( sf );
}
```

# *SfoInit*

## Purpose

A function that initializes the output structure.

## Synopsis

**SpecFileOut *SfoInit**( sf, error )

```
SpecFile        *sf;
int             *error;
```

## Description

This function initializes an output structure associated with opened SPEC data file which allows to copy scans from the opened file into another one. The return value of the function is a pointer to initialized SpecFileOut structure. On failure, NULL pointer is returned and *error* contains the appropriate error code. The associated error string can be obtained by calling *SfError()*.
The first argument, *sf,* is the SPEC data file previously opened with *SfOpen()* from which the selected scans will be copied.
After all write and related operations the memory used by the structure should be freed using *SfoClose()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfoClose(), SfoWrite(), SfoSelect(), SfoRemove(), SfOpen(), SfError().*

**SfoInit**

# Example

```
/*****************************************************************************
* This program shows how to initialize the output structure.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   SpecFileOut       *sfo;
   char              *file_name;
   int               error=0;
                  .
                  .
                  .

   sf = SfOpen( file_name, &error );
   /* Check here if open successful*/

   sfo = SfoInit( sf, &error );
   if ( sfo == (SpecFileOut *)NULL ) {
         printf(" Error during initialization of the output structure : %s\n", SfError( error ) );
         exit( 1 );
   }

                  .
                  .
                  .

   /* Here, you can use all library read and write functions */

                  .
                  .
                  .

   SfoClose( sfo );
   SfClose( sf );
}
```

# *SfoRemove*

## Purpose

A function that removes several scan indices from the output list.

## Synopsis

**long SfoRemove**( sfo, list, error )

```
SpecFileOut     *sfo;
long            *list;
int             *error;
```

## Description

This function removes several scan indices from the output list. The output list is a part of SpecFileOut structure pointed to by *sfo*.
The *list* of indices of scans to be removed must be zero terminated ! ( the last element must be a ***0*** ).
The function returns the number of scans remaining in the output list.
On failure, ***-1*** is returned and *error* reflects the reason. The associated error string can be accessed by calling *SfError()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfoRemoveOne(), SfoRemoveRange(), SfoRemoveAll(),*
*SfoSelect(), SfoSelectOne(), SfoSelectRange(), SfoSelectAll(), SfoWrite(), SfError().*

**SfoRemove**

# Example

```
/*****************************************************************************
* This program shows how to remove several scan indices from the output list.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   SpecFileOut       *sfo;
   char              *file_name;
   char              *output_file_name;
   char              buffer[99];
   long              *index_list;
   long              index=1;
   long              num_selected;
   int               i=0, error=0;
                     .
                     .
                     .
   sf = SfOpen( file_name, &error );
                     .
                     .
   sfo = SfoInit( sf, &error );
                     .
                     .
   SfoSelect . . .
                     .
                     .
   index_list = (long *)malloc( sizeof(long) );
   while( index ) {
         index_list = realloc( index_list, sizeof(long) * ( i+1 ) );
         printf( " Enter index of scan to be removed from the output list ( last = 0 ) : ");
         fgets( buffer, 99, stdin );
         index = atol( buffer );
         index_list[i] = index;
         i++;
   }
   if ( (num_selected = SfoRemove( sfo, index_list, &error ) ) < 0 ) {
         printf( "Error : %s\n", SfError( error ) );
         exit( 1 );
   } else {
         printf("There are still %li scans in the output list .\n", num_selected );
   }
   free( index_list );
                     .
                     .
                     .
   SfoWrite( sfo, output_file_name, &error );
                     .
                     .
                     .
   SfoClose( sfo );
   SfClose( sf );
}
```

# *SfoRemoveAll*

## Purpose

A function that removes all scan indices from the output list.

## Synopsis

**long SfoRemoveAll**( sfo, error )

```
SpecFileOut     *sfo;
int             *error;
```

## Description

This function removes all scan indices from the output list. The output list is a part of SpecFileOut structure pointed to by *sfo*.
The function always returns ***0***.
The parameter *error* is only a dummy to keep the same syntax with other write functions ( particularly with *SfoSelectAll()* ).

## Errors

none can occur ( see description ).

## Related Information

*SfoRemove(), SfoRemoveOne(), SfoRemoveRange (),*
*SfoSelect(), SfoSelectOne(), SfoSelectRange(), SfoSelectAll(), SfoWrite().*

**SfoRemoveAll**

# Example

```
/*****************************************************************************
* This program shows how to remove all scan indices from the output list.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   SpecFileOut       *sfo;
   char              *file_name;
   char              *output_file_name;
   int               error=0;
                     .
                     .
                     .
   sf = SfOpen( file_name, &error );
                     .
                     .
   sfo = SfoInit( sf, &error );
                     .
                     .
   SfoSelect . . .
                     .
                     .
                     .

   SfoRemoveAll( sfo, &error );

                     .
                     .
                     .
   SfoSelect . . .
   SfoWrite( sfo, output_file_name, &error );
                     .
                     .
                     .
   SfoClose( sfo );
   SfClose( sf );
}
```

# *SfoRemoveOne*

## Purpose

A function that removes one scan index from the output list.

## Synopsis

**long SfoRemoveOne**( sfo, index, error )

```
SpecFileOut     *sfo;
long            index;
int             *error;
```

## Description

This function removes one scan *index* from the output list. The output list is a part of SpecFileOut structure pointed to by *sfo*.
The function returns the number of scans remaining in the output list.
On failure, ***-1*** is returned and *error* reflects the reason. The associated error string can be accessed by calling *SfError()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfoRemove(), SfoRemoveRange (), SfoRemoveAll(),*
*SfoSelect(), SfoSelectOne(), SfoSelectRange(), SfoSelectAll(), SfoWrite().*

**SfoRemoveOne**

# Example

```
/*****************************************************************************
* This program shows how to remove one scan index from the output list.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   SpecFileOut       *sfo;
   char              *file_name;
   char              *output_file_name;
   long              num_selected;
   long              index;
   int               error=0;
                     .
                     .
                     .
   sf = SfOpen( file_name, &error );
                     .
                     .
   sfo = SfoInit( sf, &error );
                     .
                     .
   SfoSelect . . .
                     .
                     .
                     .

   if ( (num_selected = SfoRemoveOne( sfo, index, &error )) < 0 ) {
         printf("%s\n", SfError( error ) );
         exit( 1 );
   } else {
         printf("There are still %li scans in the output list .\n", num_selected );
   }

                     .
                     .
                     .
   SfoWrite( sfo, output_file_name, &error );
                     .
                     .
                     .
   SfoClose( sfo );
   SfClose( sf );
}
```

# *SfoRemoveRange*

## Purpose

A function that removes scan index range from the output list.

## Synopsis

**long SfoRemoveRange**( sfo, begin, end, error )

```
SpecFileOut     *sfo;
long            begin;
long            end;
int             *error;
```

## Description

This function removes scan indices from *begin* to *end* from the output list ( *begin* can be greater than *end* ). The output list is a part of SpecFileOut structure pointed to by *sfo*.
If one of the borders ( *begin* to *end* ) is smaller than ***1*** or greater than the number of scans in the file, nothing will be removed.
The function returns the number of scans remaining in the output list.
On failure, ***-1*** is returned and *error* reflects the reason. The associated error string can be accessed by calling *SfError()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfoRemove(), SfoRemoveOne (), SfoRemoveAll(),*
*SfoSelect(), SfoSelectOne(), SfoSelectRange(), SfoSelectAll(), SfoWrite().*

**SfoRemoveRange**

# Example

```
/*****************************************************************************
* This program shows how to remove a range scan indices from the output list.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   SpecFileOut       *sfo;
   char              *file_name;
   char              *output_file_name;
   long              num_selected;
   long              begin;
   long              end;
   int               error=0;
                     .
                     .
                     .
   sf = SfOpen( file_name, &error );
                     .
                     .
   sfo = SfoInit( sf, &error );
                     .
                     .
   SfoSelect . . .
                     .
                     .
                     .

   if ( (num_selected = SfoRemoveRange( sfo, begin, end, &error )) < 0 ) {
                     printf("%s\n", SfError( error ) );
                     exit( 1 );
   } else {
                     printf("There are still %li scans in the output list .\n", num_selected );
   }

                     .
                     .
                     .
   SfoWrite( sfo, output_file_name, &error );
                     .
                     .
                     .
   SfoClose( sfo );
   SfClose( sf );
}
```

# *SfoSelect*

## Purpose

A function that adds several scan indices to the output list.

## Synopsis

**long SfoSelect**( sfo, list, error )

```
SpecFileOut     *sfo;
long            *list;
int             *error;
```

## Description

This function adds scan indices specified in *list* to the output list. The output list is a part of SpecFileOut structure pointed to by *sfo*.
The list of scan indices must be zero terminated ! ( the last element must be a ***0*** ). Each scan will appear only once in the list.
The function returns the number of scans in the output list. On failure, ***-1*** is returned and *error* reflects the reason. The associated error string can be accessed by calling *SfError()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfoSelectOne(), SfoSelectRange(), SfoSelectAll()*
*SfoRemove(), SfoRemoveOne(), SfoRemoveRange(), SfoRemoveAll(), SfoWrite().*

# Example

```
/*****************************************************************************
* This program shows how to add several scan indices to the output list.
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   SpecFileOut       *sfo;
   char              *file_name;
   char              *output_file_name;
   long              *index_list;
   long              index=1;
   long              num_selected;
   int               error=0;
                     .
                     .
                     .
   sf = SfOpen( file_name, &error );
                     .
                     .
   sfo = SfoInit( sf, &error );
                     .
                     .
                     .

   index_list = (long *)malloc( sizeof(long) );
   while( index ) {
         index_list = realloc( index_list, sizeof(long) * ( i+1 ) );
         printf( " Enter index that should be added to the output list ( last = 0 ) : ");
         fgets( buffer, 99, stdin );
         index = atol( buffer );
         index_list[i] = index;
         i++;
   }
   if ( (num_selected = SfoSelect( sfo, index_list, &error ) ) < 0 ) {
         printf( "Error : %s\n", SfError( error ) );
         exit( 1 );
   } else {
         printf("Now, there are %li scans in the output list .\n", num_selected );
   }
   free( index_list );

                     .
                     .
                     .
   SfoWrite( sfo, output_file_name, &error );
                     .
                     .
                     .
   SfoClose( sfo );
   SfClose( sf );
}
```

# *SfoSelectAll*

## Purpose

A function that adds all scan indices to the output list.

## Synopsis

**long SfoSelectAll**( sfo, error )

```
SpecFileOut     *sfo;
int             *error;
```

## Description

This function adds all scan indices to the output list. The output list is a part of SpecFileOut structure pointed to by *sfo*.
The function returns the number of scans in the output list.
On failure, ***-1*** is returned an *error* reflects the reason. The associated error string can be accessed by calling *SfError()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfoSelect(), SfoSelectOne(), SfoSelectRange(),*
*SfoRemove(), SfoRemoveOne(), SfoRemoveRange(), SfoRemoveAll(), SfoWrite().*

**SfoSelectAll**

# Example

```
/*****************************************************************************
* This program shows how to add all scan indices to the output list.
*****************************************************************************/
#include <SpecFile.h>

main()
{
    SpecFile        *sf;
    SpecFileOut     *sfo;
    char            *file_name;
    char            *output_file_name;
    long            num_selected;
    int             error=0;
            .
            .
            .
    sf = SfOpen( file_name, &error );
            .
            .
    sfo = SfoInit( sf, &error );
            .
            .
            .

    if ( (num_selected = SfoSelectAll( sfo, &error )) < 0 ) {
            printf("%s\n", SfError( error ) );
            exit( 1 );
    } else {
            printf("Now, there are %li scans in the output list .\n", num_selected );
    }

            .
            .
            .
    SfoWrite( sfo, output_file_name, &error );
            .
            .
            .
    SfoClose( sfo );
    SfClose( sf );
}
```

# *SfoSelectOne*

## Purpose

A function that adds one scan index to the output list.

## Synopsis

**long SfoSelectOne**( sfo, index, error )

```
SpecFileOut     *sfo;
long             index;
int             *error;
```

## Description

This function adds one scan *index* to the output list. The output list is a part of SpecFileOut structure pointed to by *sfo*. Each scan will appear only once in the list.
The function returns the number of scans in the output list.
On failure, ***-1*** is returned and *error* reflects the reason. The associated error string can be accessed by calling *SfError()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfoSelect(), SfoSelectRange(), SfoSelectAll(),*
*SfoRemove(), SfoRemoveOne(), SfoRemoveRange(), SfoRemoveAll(), SfoWrite().*

# Example

```
/*******************************************************************************
* This program shows how to add one scan index to the output list.
*******************************************************************************/
#include <SpecFile.h>

main()
{
    SpecFile            *sf;
    SpecFileOut         *sfo;
    char                *file_name;
    char                *output_file_name;
    long                num_selected;
    long                index;
    int                 error=0;
            .
            .
            .
    sf = SfOpen( file_name, &error );
            .
            .
    sfo = SfoInit( sf, &error );
            .
            .
            .

    if ( (num_selected = SfoSelectOne( sfo, &error )) < 0 ) {
            printf("%s\n", SfError( error ) );
            exit( 1 );
    } else {
            printf("Now, there are %li scans in the output list .\n", num_selected );
    }

            .
            .
            .
    SfoWrite( sfo, output_file_name, &error );
            .
            .
            .
    SfoClose( sfo );
    SfClose( sf );
}
```

# *SfoSelectRange*

## Purpose

A function that adds scan index range to the output list.

## Synopsis

**long SfoSelectRange**( sfo, begin, end, error )

```
SpecFileOut     *sfo;
long            begin;
long            end;
int             *error;
```

## Description

This function adds scan indices from *begin* to *end* to the output list ( begin can be greater than end ). The output list is a part of SpecFileOut structure pointed to by *sfo*.
If *begin* or *end* is smaller than ***1*** or greater than the number of scans in the file, nothing will be added.
The function returns the number of scans in the output list.
On failure, ***-1*** is returned and *error* reflects the reason. The associated error string can be accessed by calling *SfError()*.

## Errors

SF_ERR_MEMORY_ALLOC.

## Related Information

*SfoSelect(), SfoSelectOne(), SfoSelectAll(),*
*SfoRemove(), SfoRemoveOne(), SfoRemoveRange(), SfoRemoveAll(), SfoWrite().*

# Example

```
/*****************************************************************************
* This program shows how to add a range scan indices to the output list.
*****************************************************************************/
#include <SpecFile.h>

main()
{
    SpecFile            *sf;
    SpecFileOut         *sfo;
    char                *file_name;
    char                *output_file_name;
    long                num_selected;
    long                begin;
    long                end;
    int                 error=0;
            .
            .
            .
    sf = SfOpen( file_name, &error );
            .
            .
    sfo = SfoInit( sf, &error );
            .
            .
            .

    if ( (num_selected = SfoSelectRange( sfo, begin, end, &error )) < 0 ) {
            printf("%s\n", SfError( error ) );
            exit( 1 );
    } else {
            printf("Now, there are %li scans in the output list .\n", num_selected );
    }

            .
            .
            .
    SfoWrite( sfo, output_file_name, &error );
            .
            .
            .
    SfoClose( sfo );
    SfClose( sf );
}
```

# *SfoWrite*

## Purpose

A function that copies selected scans from one file into another one.

## Synopsis

**long SfoWrite**( sfo, name, error )

```
SpecFileOut     *sfo;
char            *name;
int             *error;
```

## Description

This function writes ( appends ) all selected scans ( see *SfoSelect ...() ,SfoRemove ...()* functions ) into the file specified by *name*. The scans and the associated file headers are copied without any changes.
The indices of the selected scans are stored in an output list which is a part of SpecFileOut structure pointed to by *sfo*.
The function returns the number of written scans. On failure ***-1*** is returned and the appropriate *error* code shows the reason of the failure. The associated error string can be obtained by calling *SfError()*.

## Errors

SF_ERR_FILE_OPEN,
SF_ERR_FILE_CLOSE,
SF_ERR_FILE_WRITE,
SF_ERR_FILE_READ.

## Related Information

*SfoInit(), SfoClose(), SfoSelect(), SfoRemove(), SfError().*

**SfoWrite**

# Example

```
/*****************************************************************************
* This program shows how to write scans which indices are in the output list into another file..
*****************************************************************************/
#include <SpecFile.h>

main()
{
   SpecFile          *sf;
   SpecFileOut       *sfo;
   char              *file_name;
   char              *output_file_name;
   int               error=0;
         .
         .
         .
   sf = SfOpen( file_name, &error );
         .
         .
   sfo = SfoInit( sf, &error );
         .
         .
         .
   SfoSelect...(...);
   SfoRemove...(...);
         .
         .
         .

   if ( SfoWrite( sfo, output_file_name, &error ) < 0 ) {
         printf( "%s\n", SfError( error );
         exit( 1 );
   }

         .
         .
         .
   SfoClose( sfo );
   SfClose( sf );
}
```

# *freeArrNZ , freePtr*

## Purpose

Functions that free allocated memory.

## Synopsis

**void freeArrNZ**( ptr, no_lines )
void ***ptr;
long no_lines;

**void freePtr**( ptr )
void *ptr

## Description

***freeArrNZ()*** deallocates the space of an array ( with number of lines specified by *lines* ) pointed to by *ptr. Ptr* should be a cast to (void ***). If **ptr* is a NULL pointer, or *lines* < ***1*** no action occurs. **ptr* is set to NULL after this operation.

***freePtr()*** = *free() (#include <stdlib.h>)*, deallocates the memory pointed to by *ptr* (a pointer to a block previously allocated by malloc(), realloc(), or calloc()) and makes the space available for further allocation. If *ptr* is a NULL pointer, no action occurs.

## Errors

none returned.

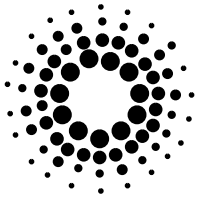
## Related Information

*free() (#include <stdlib.h>)*

**freeArrNZ , freePtr**

# Example

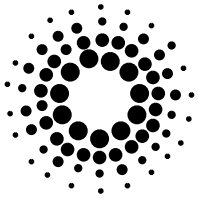
```
/*******************************************************************************
* This program allocates memory for an array and then frees it .
*******************************************************************************/
#include <SpecFile.h>

main()
{
   char         **array;
   char          buffer[500];
   long          no_lines=0;
   long          i;


   array = (char **)malloc( sizeof ( char *) );
   if ( array == (char **)NULL ) {
         printf("Memory allocation error !\n");
         exit( 1 );
   }
   while ( 1 ) {
         printf("Enter %li line ( empty line = end ) :", no_lines+1 );
         fgets( buffer , 500, stdin );
         if ( buffer == (char *)NULL || *buffer =='\n' ) break;

         array = realloc( array, sizeof ( char *) * (no_lines +1) );
         if ( array == (char **)NULL ) {
               printf("Memory allocation error !\n");
               for ( ; no_lines ; no_lines-- ) freePtr( array[no_lines-1] );
               exit( 1 );
         }

         array[no_lines] = (char *)malloc( sizeof (char) * ( strlen( buffer ) +1) );
         if ( array[no_lines] == (char *)NULL ) {
               printf("Memory allocation error !\n");
               freePtr( array );
               exit( 1 );
         }
         memcpy( array[no_lines], buffer, sizeof(char) * ( strlen( buffer )+1) );
         no_lines++;
   }

   printf( "You entered following lines :\n");
   for ( i=0 ; i<no_lines ; i++ ) {
         printf( "Line %li : %s", i+1, array[i] );
   }
   freeArrNZ( (void ***)&array, no_lines );
   freePtr( array );
}
```



# Header Line Descriptors

Each header line begins with '#' followed by a pattern describing the line type:

| | |
|---|---|
| #define SF_COMMENT | 'C' |
| #define SF_DATE | 'D' |
| #define SF_EPOCH | 'E' |
| #define SF_FILE_NAME | 'F' |
| #define SF_GEOMETRY | 'G' |
| #define SF_INTENSITY | 'I' |
| #define SF_LABEL | 'L' |
| #define SF_MON_NORM | 'M' |
| #define SF_COLUMNS | 'N' |
| #define SF_MOTOR_NAMES | 'O' |
| #define SF_MOTOR_POSITIONS | 'P' |
| #define SF_RECIP_SPACE | 'Q' |
| #define SF_RESULTS | 'R' |
| #define SF_SCAN_NUM | 'S' |
| #define SF_TIME_NORM | 'T' |
| #define SF_USER_DEFINED | 'U' |
| #define SF_TEMPERATURE | 'X' |
| #define SF_MCA_DATA | '@' |





# Structures

## *SpecFile*

```
typedef struct _SpecFile{
  FILE                          *fd;
  struct _ListHeader             list;
  long int                       no_scans;
  ObjectList                    *current;
  double                        **data;
  long                          *data_info;
  char                          **labels;
  long int                       no_labels;
  double                        *motor_pos;
  long int                       no_motor_pos;
  char                          **motor_names;
  long int                       no_motor_names;
  long int                       header_offset;
  long int                       epoch;
} SpecFile;
```

**fd**

is the pointer to the opened SPEC data file;

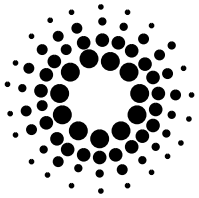
**list**

contains the first and the last element of the linked list of SpecScan structures.

```
typedef struct _ListHeader {
 struct _ObjectList             *first;
 struct _ObjectList             *last;
} ListHeader;
```

The list is made of following elements, whereby contents is a pointer to SpecScan structure :

```
typedef struct _ObjectList {
 struct _ObjectList             *next;
 struct _ObjectList             *prev;
 void                           *contents;
} ObjectList;
```



**no_scans**
contains the total number of scans in the SPEC data file;

**current**
points to the current list element ( current scan );

**data**
contains a data array of the current scan. It is filled up when calling one of the data reading functions. Next call of a function of this type causes reading data from this array and not from the file , what improves the program performance;

**data_info**

| | |
|---|---|
| data_info[ROW] | - number of data rows, |
| data_info[COL] | - number of data columns, |
| data_info[REG] | - 0 if data regular |
| | - 1 if data not regular ( number of columns not constant ); |

**labels**
is an array with data column labels of the current scan. It is filled up when calling one of the label reading functions. Next call of a function of this type causes reading the labels from this array and not from the file , what improves the program performance;

**no_labels**
number of labels in the array;

**motor_pos**
contains motor positions of the current scan after one of motor position reading functions has been called. Next call of a function of this type causes reading the positions from this array and not from the file , what improves the program performance;

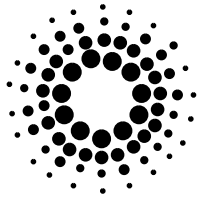
**no_motor_pos**
number of motor positions;

**motor_names**
stores motor names from the last file header. It is filled up after one of motor names reading functions has been called. Next call of a function of this type causes reading the names from this array and not from the file , what improves the performance of the program;

**no_motor_names**
number of motor names in the array;

**header_offset**
contains the offset of the last file header;

**epoch**
is the epoch from the last file header;





# *SpecScan*

```
typedef struct _SpecScan {
  long int                    index;
  long int                    scan_no;
  long int                    order;
  long int                    offset;
  long int                    data_offset;
  long int                    last_data_offset;
  long int                    last_header;
  long int                    data_lines;
  long int                    header_lines_before;
  long int                    header_lines_after;
} SpecScan;
```

**index**
scan index created by *SfOpen()*. First scan in the file has the index 1;

**scan_no**
scan number;

**order**
scan order. If there are more than one scans with the same number, the first found in the file will get the order 1, the second 2 and so on. If there is only one scan with certain number, the order is always 1;

**offset**
represents the offset of the first scan line ( *'#S'* ) in the file;

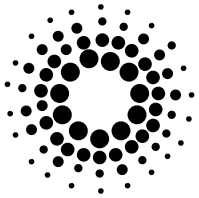
**data offset**
**last_data_offset**
is the offset of the first / last data line of a scan in the file;

**last_header**
contains th offset of the last file header in the file;

**data_lines**
number of data lines in the scan;

**header_lines_before**
number of scan header lines (lines beginning with '*#*' ) before scan data.
It is at least 1, because there must be always a *'#S'* line indicating new scan.

**header_lines_after**
number of scan header lines after scan data.



# *SpecFileOut*

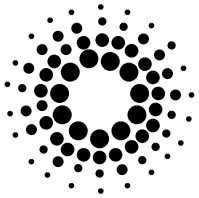
```
typedef struct _SpecFileOut{
  SpecFile                    *sf;
  long                        *list;
  long                         list_size;
  long                         last_header;
} SpecFileOut;
```

**sf**
: pointer to a SpecFile structure of a file from which the selected scans are to be copied;

**list**
: output list which contains indices of selected scans;

**list_size**
: number of selected scans in the output list.

**last_header**
: offset of the last written file header in the input file.





# Errors

**ERROR CODE** *ERROR STRING*
DESCRIPTION

---

**SF_ERR_COL_NOT_FOUND** *"Column not found error ( SpecFile )"*
column number is out of range( SfDataCol() ), or there is no column with such name ( SfDataColByName() ).

**SF_ERR_FILE_CLOSE** *"File close error ( SpecFile )"*
error occurred during fclose() in SfoWrite().

**SF_ERR_FILE_OPEN** *"File open error ( SpecFile )"*
file could not be opened. Make sure that file exists ( SfOpen() ).

**SF_ERR_FILE_READ** *"File read error ( SpecFile )"*
error during one of following file reading functions :
- feof();
- ftell();
- fseek();

**SF_ERR_FILE_WRITE** *"File write error ( SpecFile )"*
could not write a character with fputc() in SfoWrite();

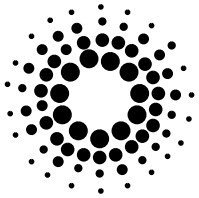
**SF_ERR_HEADER_NOT_FOUND** *"Header not found error ( SpecFile )"*
there is no file header before this scan. It is more an information than an error.

**SF_ERR_LABEL_NOT_FOUND** *"Label not found error ( SpecFile )"*
there is no label for this data column ( SfLabel() );

**SF_ERR_LINE_EMPTY** *"Line empty or wrong data error ( SpecFile )"*
header line does not contain expected information or empty.

**SF_ERR_LINE_NOT_FOUND** *"Line not found error ( SpecFile )"*
there is no such header line.

**SF_ERR_MEMORY_ALLOC** *"Memory allocation error ( SpecFile )"*
could not allocate ( malloc() ) or reallocate ( realloc() ) memory.



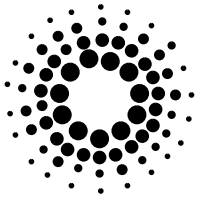
**SF_ERR_MOTOR_NOT_FOUND** *"Motor not found error ( SpecFile )"*
there is no motor name with this number ( SfMotor() ). It is more an information than an error.

**SF_ERR_POSITION_NOT_FOUND** *"Position not found error ( SpecFile )"*
there is no corresponding position for this motor ( SfMotorPos(), SfMotorPosByName() ). It is more an information than an error.

**SF_ERR_SCAN_NOT_FOUND** *"Scan not found error ( SpecFile )"*
scan with such index does not exist. It is more an information than an error.

**SF_ERR_USER_NOT_FOUND** *"User not found error ( SpecFile )"*
the first comment line in the last file header does not contain following string : "User =". It is more an information than an error.

**SF_ERR_NO_ERRORS** *"OK ( SpecFile )"*
no errors has occurred.

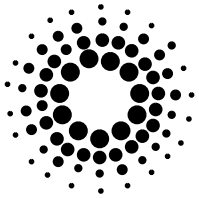




# Functions - Short List

**Alphabetical list :**

SfAllLabels();
SfAllMotors();
SfAllMotorPos();
SfClose();
SfCommand();
SfCondList();
SfData();
SfDataAsString();
SfDataCol();
SfDataColByName();
SfDataLine();
SfDate();
SfDebugOff();
SfDebugOn();
SfEpoch();
SfError();
SfFileDate();
SfFileHeader();
SfGeometry();
SfHKL();
SfHeader();
SfIndex();
SfLabel();
SfList();
SfMotor();
SfMotorPos();
SfMotorPosByName();
SfNoColumns();
SfNoDataLines();
SfNoHeaderBefore();
SfNumber();
SfNumberOrder();
SfOpen();
SfOrder();
SfScanNo();
SfTitle();
SfUser();
SfoClose();
SfoGetList();
SfoInit();
SfoRemove();
SfoRemoveAll();
SfoRemoveOne();
SfoRemoveRange();
SfoSelect();
SfoSelectAll();
SfoSelectOne();
SfoSelectRange()
SfoWrite();
freeArrNZ();
freePtr();



**Functions listed by groups:**

*Specfile:*

SfOpen();
SfClose();

*Scan lists:*

SfScanNo();
SfList();
SfCondList();

*Scan number, order, index:*

SfIndex();
SfNumber();
SfOrder();
SfNumberOrder();

*Header info:*

*file header :*

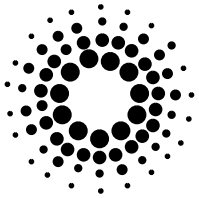
SfUser();
SfTitle();
SfEpoch();
SfFileDate();
SfFileHeader();

*scan header:*

SfCommand();
SfDate();
SfNoColumns();
SfHKL();
SfGeometry();
SfHeader();
SfNoHeaderBefore();

*Data:*

SfNoDataLines();
SfData();
SfDataAsString();
SfDataLine();
SfDataCol();
SfDataColByName();





*Data column labels:*

SfAllLabels();
SfLabel();

*Motor info:*

SfAllMotors();
SfMotor();
SfAllMotorPos();
SfMotorPos();
SfMotorPosByName();

*Specfile output:*

SfoInit();
SfoClose();

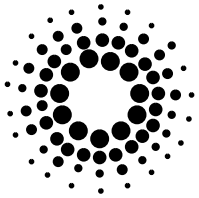
*Output selection:*

SfoSelectOne();
SfoSelect();
SfoSelectRange();
SfoSelectAll();
SfoRemoveOne();
SfoRemove();
SfoRemoveRange();
SfoRemoveAll();
SfoWrite();
SfoGetList();

*Utilities:*

SfError();
SfDebugOn();
SfDebugOff();

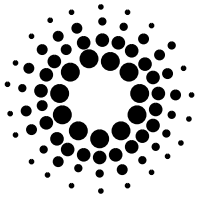
freeArrNZ();
freePtr();



# Bibliography


[Kerningham] Kerningham,B.W and D.M.Ritchie, *The C Programming Language*, Prentice Hall, 1978.

[CSS] *SPEC, X-Ray Diffraction Software - User Manual and Tutorials*, Certified Scientific Software.



# Index