EURO 3D

Promoting 3D spectroscopy in Europe

www.aip.de/Euro3D

RESEARCH TRAINING NETWORK

Sponsored by the EUROPEAN COMMISSION

Euro3D I/0 Libraries – v1.0a

Cookbooks

Issue 1.0b

14/04/2003

DRAFT

Authors P. Ferruit, A. Pécontal-Rousset and Y. Copin
Contact P. Ferruit (ferruit@obs.univ-lyon1.fr)

This page was intentionally left blank

Change Record

Issue	Date	Section affected	Reason/Initiation/Documents/Remarks
0.0	10/03/2003	all	Creation
1.0a	25/03/2003	all	Iteration inside the LYO team
1.0b	14/04/2003	all	Release of an α version of the document together with the Euro3D libraries v1.0a

This page was intentionally left blank

Contents

This page was intentionally left blank

1 Introduction

1.1 Purpose and scope

The ambition of this document is to provide simple examples of how to use the routines available in the Euro3D I/O libraries v1.0a. It will also include the frequently asked questions that will (without doubt) emerge as the libraries are used within the Euro3D network.

Warning: Some example programs are still missing...

1.2 Reference documents

[1]	*Euro3D format – Format Definition* M. Kissler-Patig et al.	issue ?.?, ??/??/??
[2]	*Euro3D I/O libraries v1.0a – Installation guide* A. Pécontal-Rousset, P. Ferruit and Y. Copin	issue 1.0a, 14/04/2003
[3]	*Euro3D I/O libraries v1.0b – Developer guide* Y. Copin, P. Ferruit and A. Pécontal-Rousset	issue 1.0b, 14/04/2003

Documents [1], [2] and [3] can be found on `http://www.aip.de/Euro3D`.

1.3 Abbreviations and acronyms

AD	Applicable Document
AIP	Astrophysikalisches Institut Postdam
DIT	Detector Integration Time
DQ	Data Quality
ESO	European Southern Observatory
FITS	Flexible Image Transport System
FWHM	Full Width at Half Maximum
HDU	Header and Data Unit
IFS	Integral Field Spectroscopy
IFU	Integral Field Unit
MPE	Max-Planck Institut für extraterrestrische Physik
NDIT	Number of DIT
NIR	Near Infra-Red
NOST	Nasa / Science Office of Standards and Technology
PA	Position Angle
PSF	Point Spread Function
RD	Reference Document
RON	Read-Out Noise
RMS	Root Mean Square
SPAXEL	SPAtial piXture ELement
S/N	Signal-to-Noise ratio
TBD	To Be Determined
WCS	World Coordinate System

2 Installation of the distribution of example programs

The example programs described in this document are available as part of compressed tar-ball (`E3D_Cookbooks-1.0b.tar.gz`), which can be downloaded from the internal pages of www.aip.de/Euro3D.

To install and compile these programs:

1. Install the Euro3D I/O libraries (*v1.0a*) if this not already done. These libraries can also be retrieved from the AIP site www.aip.de/Euro3D (internal pages). Follow the installation instructions provided with the distribution (see also the Installation Guide [2]).
2. Download the `E3D_Cookbooks-1.0b.tar.gz` file from the AIP web site.
3. Copy it to the root directory of the Euro3D libraries `$E3D_PATH`. This environment variable shall exist if you have followed the installation instructions of the Euro3D libraries (see step #1).
4. Uncompress and untar the file:
   - `tar xvzf E3D_Cookbooks-1.0b.tar.gz`

   This will create a `E3D_Cookbooks-1.0b/` directory.
5. Change directory to `E3D_Cookbooks-1.0b/`.
6. Type the commands:
   - `./configure`
   - `make`

You will find:

- The source files of the example programs under the `$E3D_PATH/E3D_Cookbooks-1.0b/pkg/Examples/source/` directory.
- The executables under the `$E3D_PATH/E3D_Cookbooks-1.0b/user/bin/` directory.
- The LaTeX sources of this document are available under the `$E3D_PATH/E3D_Cookbooks-1.0b/doc/Cookbooks/` directory.

3 Session handling

The Euro3D I/O library provides some functionalities to facilitate the handling of the input parameters of a program, as well as to handles error, warning and regular messages. We are going to detail the content of the example C-program `SessionHandling.c` provided with this documentation.

3.1 Key variables and routines

Session handling routines and variables

`set_version()`	used to specify the version number of the program
`set_arglist()`	used to define the list of input arguments
`init_session()`	session initialization routine
`get_argval()`	used to read the value of an argument (single value, no complex syntax)
`is_true()`	returns 1 if a "flag" argument is set to true, 0 else
`exit_session()`	used to exit the Euro3D session
`char **argval`	array of character strings containing the input values of the program arguments
`char **arglabel`	array of character strings containing the input labels of the program arguments

Message handling routines

`print_msg()`	print a message to the standard output
`print_error()`	print an error message to the standard output
`print_warning()`	print a warning message to the standard output

3.2 Example program

The program starts with a standard DIOXYGEN header which will allow to automatically build the documentation of the program.

```
/* ================================================================ */
/*!
\file       SessionHandling.c
\copyright  (c) 2003 - Euro3D
\date       13/03/2003
\author     Pierre Ferruit
\version    0.0 13/03/2003 PF Creation
\keywords   Session Handling, Cookbooks
\brief      Session handling example file

Example file provided with the documentation of the
Euro3D libraries (v6.0). This file provide an
example of how to use the session handling tools
(handling of the input arguments of a program)

*/
/* ================================================================ */
```

Then we include the standard Euro3D include file `E3D_io.h`.

```
#include        <IFU_io.h>
```

It is followed by the header of the `main` program and the variable declarations. The two character pointers `**argval` and `**arglabel` will be used to store the value of the arguments and their labels, respectively.

```
/* ================================================================ */
/*!
\page SessionHandling SessionHandling

Brief description.

\param [\c -name|nom] name of the computer
\param [\c -rank|rang] rank of the computer (defaulted to 1)
\param [\c -parameters] size and weight of the computer
(size,weight)
\param [\c -windows] if present, Zindoz is installed on your
computer
\param [\c -linux] if present, Linux is installed on your
computer
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================ */

int  main(int argc, char **argv)
{
char      **argval;              /* input arguments of the program  */
char      **arglabel;            /* corresponding labels            */

char      name[81];              /* name of the computer            */
long      rank;                  /* rank of the computer            */
float     size;                  /* size of the computer            */
float     weight;                /* weight of the computer          */
long      flag_windows;          /* =0 no windows, =1 windows       */
long      flag_linux;            /* =0 no linux, =1 linux           */

char      error_txt[161];        /* error messages                  */
char      msg_txt[161];          /* regular messages                */
char      warning_txt[161];      /* warning messages                */
```

First, you need to specifiy the version number of program (it will be printed each time it is executed) using the `set_version()` routine. To initialize the Euro3D session, you then need to define the list of the input arguments of the program. This is done using the `set_arglist()` routine. Each argument is in the form of `-ArgumentName` and is followed either by none (no default value, argument *must* be specified on input), or by a default value or by nothing (in which case the argument is considered as a flag). Different names can be used for the same input argument argument using the following syntax: `-Name1|Name2`. The name used by the user will be stored in the `arglabels` array.

The order in which the arguments appear in the `set_arglist` routine will set its index in the `**argval` and `**arglabel` arrays (first argument will have zero index).

```
/* ---------------------------------------------------------------- */
/* Initialization of the Euro3D session                             */
/* ---------------------------------------------------------------- */
```

```
set_version("1.0");
set_arglist("-name|nom none -rank|rang 1 -parameters none -windows -linux");
```

Then the session can be started using the `init_session()`, which will allocate and fill the `**argval` and `**arglabel` arrays.

```
init_session(argv, argc, &arglabel, &argval);
```

The values of the arguments can be accessed either through the `get_argval()` routine or directly using the standard `sscanf()` C-routine. For "flag" arguments like the `-windows` and `-linux` ones, the routine `is_set()` is available to test their presence in the command line. The Euro3D session handling software provides, by default, a `-debug` option (you do not need to specifiy it in your call to `set_arglist()`). If specified on input, it will set a global variable labelled `DEBUG` to 1. This allows you to isolate parts of the code you want to execute only in "debug" mode. In this example, this is the test on rank of your computer which is inside a `if (DEBUG) {}` block.

Note, in this example, the use of the the `print_msg()`, `print_warning()` and `print_error()` routines. We advise to use these routines instead of the classical `printf()` C-routine as they provide a transparent interface to a possible GUI. To exit the session, use the `exit_session()` routine instead of the classical `return()` C-routine.

```
/* --------------------------------------------------------------- */
/* Analysis of the input arguments                                 */
/* --------------------------------------------------------------- */
get_argval(0, "%s", name);

get_argval(1, "%ld", &rank);
if (DEBUG) {
if (rank > (long) 3) {
sprintf(warning_txt, "You computer rank (%ld) is too low !", rank);
print_msg(warning_txt);
}
}

if (sscanf(argval[2], "%f,%f", &size, &weight) != 2) {
print_error("Unable to read the size and weight of the computer.");
exit_session((int) -1);
}

if ((size <= (float) 0) || (weight <= (float) 0)) {
sprintf(error_txt, "Computer size   : %f australian nm", size);
print_error(error_txt);
sprintf(error_txt, "Computer weight : %f australian pounds", weight);
print_error(error_txt);
print_error("Size and weight shall be strictly positive...");
exit_session((int) -1);
}

flag_windows = is_true(argval[3]);
flag_linux = is_true(argval[4]);
```

No real comment on the core of this very sophisticated "Hell world" type program...

```
/* ---------------------------------------------------------------- */
/* Core of this extremly interesting program                        */
/* ---------------------------------------------------------------- */
print_msg("Hello dear human partner,");
sprintf(msg_txt, "My name is %s", name);
print_msg(msg_txt);
sprintf(msg_txt, "I am number %ld in my branch.", rank);
print_msg(msg_txt);
sprintf(msg_txt, "My size is %f australian nm", size);
print_msg(msg_txt);
sprintf(msg_txt, "and I weight %f australian pounds", weight);
print_msg(msg_txt);
if (flag_windows) {
print_msg("I have zindoz available.");
}
if (flag_linux) {
print_msg("I have linux available.");
}
```

It is now time to exit the program by ending the session. Note that if you compile this program, the compiler may complain about the lack of `return()` instruction (this is the case if you use `gcc` with the `-Wall` flag set). You can either ignore this warning or add a `return(0)` instruction just after the `exit_session(0)` line. It will never be executed (the program is exited when the `close_session()` instruction is executed), but the compiler will be satisfied...

```
/* ---------------------------------------------------------------- */
/* Session end                                                      */
/* ---------------------------------------------------------------- */
exit_session(0);
return(0);
}
```

3.3 Executing the example program

To test this program, you can execute the following command:
`$(E3D_PATH/E3D_CookBook-1.0/user/bin/SessionHandling -name Gino -rank 4 -parameters 0.4,10.5 -linux`
This should produce the following output:

```
Program: SessionHandling  Version 1.0-6.0       Obs. de Lyon
Hello dear human partner,
My name is Gino
I am number 4 in my branch.
My size is 0.400000 australian nm
and I weight 10.500000 australian pounds
zindoz is not available.
I have linux available.
End of SessionHandling
```

If you execute the program with the `-debug` option then it will also print:

```
You computer rank (4) is too low !
```

If you do something wrong (you forgot a mandatory argument or specified an unknown argument) as in the example below,
`$(E3D_PATH/E3D_CookBook-1.0/user/bin/SessionHandling -wrongargument`
you will get this type of message:

```
ERROR: FATAL error from routine parse_arg : option -wrongargument
ERROR: SessionHandling aborted - status code : -1
```

If you do not remember which arguments are accepted by a program, enter the command,
`$(E3D_PATH/E3D_CookBook-1.0/user/bin/SessionHandling -h` (or `-help`)
and you will get the following list:

```
available options :
-name|nom nodefault [-rank|rang 1 ] -parameters nodefault [-windows] [-linux]
[-h] [-help] [-inputformat] [-outputformat] [-version] [-debug] [-tk] [-quiet]
[-noask]
```

Note the presence of other standard input arguments, which will be explained in other section of this documents.

3.4 FAQ

4 Manipulating 1D images (spectra)

4.1 The IMAGE1D/SPECTRUM structure

In the following, we list the most important fields of the IMAGE1D/SPECTRUM structure (i.e. those a regular user may have to access while developing programs under the Euro3D environment). It is declared in the `items.h` include file of the `E3D_io_LCL-1.0a` library package and is available both as IMAGE1D and SPECTRUM.

```
typedef struct
{
char   name[lg_name+1];     /* name of the spectrum          */
...
char   history[lg_hist+1]; /* history of the spectrum       */
...
char   ident[lg_ident+1];   /* identifier of the spectrum    */
char   cunit[lg_unit+1];    /* units of the axes             */
int    npts;                /* number of pixels              */
double start;               /* start coordinate              */
double end;                 /* end coordinate                */
double step;                /* step size                     */
...
double min;                 /* min & max of intensity        */
double max;
...
} IMAGE1D;
```

The `lg_name`, `lg_hist`, `lg_ident` and `lg_unit` lengths are actually aliases defined in one of the standard Euro3D include files (`gendef.h`) and are available through the mandatory `E3D_io.h` include file.

> **Warning:** The `min` and `max` fields are only updated if you are *opening* an existing spectrum or if you make a call to the `spec_minmax()` routine provided with the libraries.

4.2 Key variables and routines

Routines to manipulate spectra (1D images)

`create_spec()`	create a new spectrum
`open_spec()`	open an existing spectrum
`close_spec()`	close a spectrum
`RD_spec()`	read the value of a given pixel
`RD_qspec()`	read the quality value of a given pixel
`WR_spec()`	write a value into a given pixel
`WR_qspec()`	write the quality of a given pixel
`coord_spec()`	return the coordinate of a given pixel
`pixel_spec()`	return the index of the pixel corresponding to a given coordinate

Routines to manipulate spectra in memory

`alloc_spec_mem()`	equivalent to `open_spec()` but the spectrum is not associated to an actual file
`free_spec_mem()`	free the memory allocated for a spectrum data

Warning: The first pixel of a spectrum corresponds to index 0 and the last one to index spectrum.npts - 1

4.3 Example program #1: creating and filling a 1D image (spectrum)

In this first example, we will describe how to create a new spectrum, fill it and then close it. The example program `SpectrumHandling1.c` starts with the usual headers and includes the mandatory `E3D_io.h` include file.

```
/* ================================================================= */
/*!
\file        SpectrumHandling1.c
\copyright   (c) 2003 - Euro3D
\date        14/03/2003
\author      Pierre Ferruit
\version     1.0 14/03/2003 PF Creation
\keywords    Spectrum Handling, Cookbooks
\brief       Spectrum handling example file #1

Example file provided with the documentation of the
Euro3D libraries (v6.0). This file provide an
example of how to create and fill a 1D image (spectrum)
within the Euro3D environment.

*/
/* ================================================================= */

#include        <math.h>
#include        <IFU_io.h>


/* ================================================================= */
/*!
\page SpectrumHandling1 SpectrumHandling1

Brief description.

\param [\c -name] name of the output spectrum
\param [\c -npix] number of pixels of the spectrum
(defaulted to 100)
\param [\c -start] start value of the spectrum
(defaulted to 0.)
\param [\c -step] step value of the spectrum
(defaulted to 1.)
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================= */

int  main(int argc, char **argv)
```

```
{
```

Among the variables we define a IMAGE1D-type (or equivalently SPECTRUM-type) one which will be used throughout the program.

```
char      **argval;           /* input arguments of the program  */
char      **arglabel;         /* corresponding labels            */

IMAGE1D   spectrum;           /* spectrum structure              */
char      name[81];           /* name of the spectrum            */
long      npix;               /* number of pixels                */
double    start;              /* start coordinate                */
double    step;               /* step size                       */
char      units[lg_unit+1];   /* spectrum unit string            */
char      ident[lg_ident+1];  /* identifier string               */

long      i;                  /* work variable                   */
float     x,y;                /* work variables                  */
```

We initialize the Euro3D session, including the list of input parameters for our program. Once this is done, we read the contents of the input arguments and store them in the corresponding variables.

```
/* --------------------------------------------------------------- */
/* Initialization of the Euro3D session                            */
/* --------------------------------------------------------------- */
set_version("1.0");
set_arglist("-name none -npix 100 -start 0. -step 1.");
init_session(argv, argc, &arglabel, &argval);

/* --------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)    */
/* --------------------------------------------------------------- */
get_argval(0, "%s", name);
get_argval(1, "%ld", &npix);
get_argval(2, "%lf", &start);
get_argval(3, "%lf", &step);
```

Now we create the spectrum using the `create_spec()` routine. On input, we provide: the pointer to the `IMAGE1D / SPECTRUM` structure (`&spectrum`); the parameters of the spectrum (number of points, start coordinate and step size); the data type (SHORT, LONG, INT, FLOAT or DOUBLE); the identifier of the spectrum; the units of the axes (???).

```
/* --------------------------------------------------------------- */
/* Creating the spectrum using the open_spec() routine             */
/* --------------------------------------------------------------- */

sprintf(ident, "Test spectrum");
sprintf(units, "Units x-axis      Units y-axis");

create_spec(&spectrum, name, npix, start, step, FLOAT, ident, units);
```

Once the spectrum created, we fill it with an arbitrary function (namely a Gaussian profile). The loop goes from 0 (index of the first pixel) to `spectrum.npts - 1`. To know the coordinate of a given

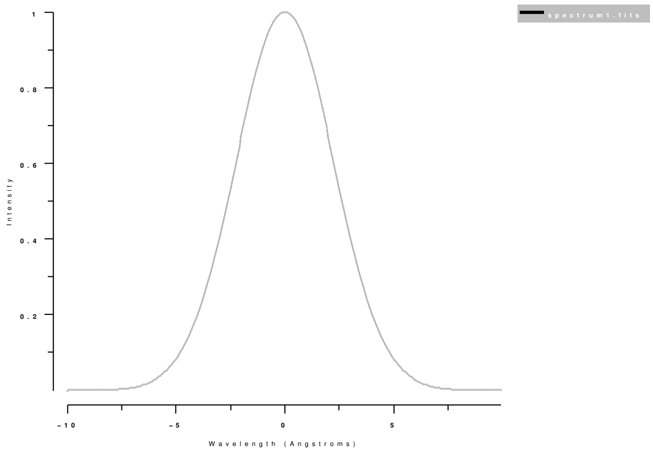

Figure 1: Plot of the spectrum1.fits spectrum created by the SpectrumHandling1 program.

pixel, we use the function `coord_spec()` (inversly, the index of a pixel corresponding to a given coordinate can be obtained using the function `pixel_spec()`). To write the pixel value and quality, we use the `WR_spec()` and `WR_qspec()` routines, respectively.

```
/* ----------------------------------------------------------------- */
/* Filling the spectrum with y=f(x)= exp(-x**2/10.)                  */
/* and set the quality to 0 (OK)                                     */
/* ----------------------------------------------------------------- */
for (i = 0 ; i < spectrum.npts ; i++) {
x = coord_spec(&spectrum, i);
y = exp(- x * x / 10.);
WR_spec(&spectrum, i, y);
WR_qspec(&spectrum, i, (long) 0);
}
```

Last we close the spectrum (everything will be written to disk) and exit the Euro3D session.

```
/* ----------------------------------------------------------------- */
/* Closing the spectrum                                              */
/* ----------------------------------------------------------------- */
close_spec(&spectrum);

/* ----------------------------------------------------------------- */
/* Session end                                                       */
/* ----------------------------------------------------------------- */
exit_session(0);
}
```

The program can be executed by typing:

```
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/SpectrumHandling1 -name spectrum1 -npix 200 -start
-10 -step 0.1
```

The messages on the screen will be minimalist as we did not print any message in the program:

```
Program: SpectrumHandling1  Version 1.0-6.0       Obs. de Lyon
End of SpectrumHandling1
```

A plot of the output spectrum is displayed in Fig. 4.3. If the same command is executed a second time, the user will be prompted wether to overwrite the spectrum. To avoid that, it is possible to specify the `-noask` option.

By default the spectrum is created in the format specified in the `E3D_DEFAULT_FMT` environment variable. To change that, it is possible to specify the `-outputformat`.

4.4 Example program #2: opening and reading a 1D image (spectrum)

In this second example, we show how to open an existing spectrum and read its data.

```
/* ================================================================== */
/*!
\file       SpectrumHandling2.c
\copyright  (c) 2003 - Euro3D
\date       14/03/2003
\author     Pierre Ferruit
\version    1.0 14/03/2003 PF Creation
\keywords   Spectrum Handling, Cookbooks
\brief      Spectrum handling example file #2

Example file provided with the documentation of the
Euro3D libraries (v6.0). This file provide an
example of how to open and read a 1D image (spectrum)
within the Euro3D environment.

*/
/* ================================================================== */

#include        <math.h>
#include        <IFU_io.h>


/* ================================================================== */
/*!
\page SpectrumHandling2 SpectrumHandling2

Brief description.

\param [\c -name] name of the input spectrum
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================== */

int  main(int argc, char **argv)
{
char      **argval;              /* input arguments of the program  */
char      **arglabel;            /* corresponding labels            */

IMAGE1D   spectrum;              /* spectrum structure              */
char      name[81];              /* name of the spectrum            */
```

```
long      i, n;                 /* work variable                    */
float     value;                /* work variables                   */
char      msg_txt[161];         /* message text                     */
```

The program has a single input argument (the name of the input spectrum).

```
/* ----------------------------------------------------------------- */
/* Initialization of the Euro3D session                              */
/* ----------------------------------------------------------------- */
set_version("1.0");
set_arglist("-name none");
init_session(argv, argc, &arglabel, &argval);

/* ----------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)     */
/* ----------------------------------------------------------------- */
get_argval(0, "%s", name);
```

Then, we open the spectrum in read-only mode as we do not need to modify its contents. Other possible modes are "Output" and "IO".

```
/* ----------------------------------------------------------------- */
/* Opening the spectrum (in read-only mode)                          */
/* ----------------------------------------------------------------- */
open_spec(&spectrum, name, "Input");
```

Then we compute the average of the spectrum. The spectrum values are read using the `RD_spec()`.

```
/* ----------------------------------------------------------------- */
/* Computing the average using only pixels with quality flag        */
/* of 0 (OK)                                                         */
/* ----------------------------------------------------------------- */
value = (float) 0;
n = (long) 0;

for (i = 0 ; i < spectrum.npts ; i++) {
if (RD_qspec(&spectrum,i) == (long) 0) {
value += RD_spec(&spectrum, i);
n++;
}
}

if (n != (long) 0) {
value /= n;
sprintf(msg_txt, "Average value of the spectrum: %12.6e", value);
print_msg(msg_txt);
}
else {
print_msg("No valid point in the spectrum.");
}
```

Last, we close the spectrum and exit the session.

```
    /* ---------------------------------------------------------------- */
    /* Closing the spectrum                                             */
    /* ---------------------------------------------------------------- */
    close_spec(&spectrum);

    /* ---------------------------------------------------------------- */
    /* Session end                                                      */
    /* ---------------------------------------------------------------- */
    exit_session(0);
}
```

The program can be executed by typing (here we assume that the program `SpectrumHandling1` described in the previous section has been executed in the same directory and that the spectrum `spectrum1.fits` already exists):

```
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/SpectrumHandling2 -name spectrum1
```

The messages on the screen will be the following:

```
Program: SpectrumHandling2  Version 1.0-6.0       Obs. de Lyon
Average value of the spectrum: 2.802474e-01
End of SpectrumHandling2
```

If the input spectrum does not exist (e.g. if we try to open the spectrum `spectrum2.fits`), the program will automatically exit with an error message:

```
Program: SpectrumHandling2  Version 1.0-6.0       Obs. de Lyon
ERROR: FATAL error from routine open_spec: spec ./spectrum2.fits
ERROR: Error opening file

SpectrumHandling2 aborted - status code : -10
```

It is possible to avoid this automated behavior using the `set_error_level()` routine (see Sect. 10) and by analizing the return value of the library routines (this is not done in the example programs).

4.5 Example program #3: manipulating spectra in memory

Not yet available.

4.6 FAQ

5 Manipulating 2D images (frames)

5.1 The IMAGE2D structure

In the following, we list the most important fields of the IMAGE2D structure (i.e. those a regular user may have to access while developing programs under the Euro3D environment). It is declared in the `items.h` include file of the `E3D_io_LCL-1.0a` library package.

```
typedef struct
{
char   name[lg_name+1];     /* name of the image             */
...
char   history[lg_hist+1];  /* history of the image          */
...
char   ident[lg_ident+1];   /* identifier of the image       */
char   cunit[lg_unit+1];    /* units of the axes             */
int    nx;                  /* size along the x-axis         */
int    ny;                  /* size along the y-axis         */
double startx;              /* x-axis start coordinate       */
double starty;              /* y-axis start coordinate       */
double endx;                /* x-axis end coordinate         */
double endy;                /* y-axis end coordinate         */
double stepx;               /* step size along the x-axis    */
double stepy;               /* step size along the y-axis    */
double min;                 /* min & max of intensity        */
double max;
...
} IMAGE2D;
```

The `lg_name`, `lg_hist`, `lg_ident` and `lg_unit` lengths are actually aliases defined in one of the standard Euro3D include files (`gendef.h`) and are available through the mandatory `E3D_io.h` include file.

> **Warning:** The `min` and `max` fields are only updated if you are *opening* an existing image or if you make a call to the `image_minmax()` routine provided with the libraries.

5.2 Key variables and routines

Routines to manipulate frames (2D images)

`create_frame()`	create a new 2D image
`open_frame()`	open an existing 2D image
`close_frame()`	close a 2D image
`RD_frame()`	read the value of a given pixel
`RD_qframe()`	read the quality value of a given pixel
`WR_frame()`	write a value into a given pixel
`WR_qframe()`	write the quality of a given pixel

> **Warning:** The first pixel of a 2D image corresponds to indices (0,0) and the last one to indices (frame.nx - 1 , frame.ny - 1).

5.3 Example program #1: creating and filling a 2D image (frame)

In this first example, we will describe how to create a new 2D image, fill it and then close it. The example program `FrameHandling1.c` starts with the usual headers and includes the mandatory `E3D_io.h` include file.

```
/* ================================================================= */
/*!
\file       FrameHandling1.c
\copyright  (c) 2003 - Euro3D
\date       15/03/2003
\author     Pierre Ferruit
\version    1.0 15/03/2003 PF Creation
\keywords   2D Image  Handling, Cookbooks
\brief      2D image handling example file #1

Example file provided with the documentation of the
Euro3D libraries (v1.0). This file provide an
example of how to create and fill a 2D image
within the Euro3D environment.

*/
/* ================================================================= */

#include        <math.h>
#include        <IFU_io.h>


/* ================================================================= */
/*!
\page FrameHandling1 FrameHandling1

Brief description.

\param [\c -name] name of the output 2D image
\param [\c -npix] number of pixels of the 2D image
along the x- and y-axes (defaulted to 100,100)
\param [\c -start] start values of the 2D image
for the x- and y-axes (defaulted to 0.,0.)
\param [\c -step] step size along the x- and y-axes
(defaulted to 1.,1.)
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================= */

int  main(int argc, char **argv)
{
```

Among the variables we define a IMAGE2D-type one which will be used throughout the program.

```
char      **argval;              /* input arguments of the program  */
```

```
char      **arglabel;           /* corresponding labels          */

IMAGE2D   image;                /* 2D image structure            */
char      name[81];             /* name of the spectrum          */
int       npix[2];              /* number of pixels              */
double    start[2];             /* start coordinates             */
double    step[2];              /* step sizes                    */
char      units[lg_unit+1];     /* unit string                   */
char      ident[lg_ident+1];    /* identifier string             */

long      i, j;                 /* work variable                 */
float     x,y,z;                /* work variables                */
```

We initialize the Euro3D session, including the list of input parameters for our program. Once this is done, we read the contents of the input arguments and store them in the corresponding variables.

```
/* ----------------------------------------------------------------- */
/* Initialization of the Euro3D session                              */
/* ----------------------------------------------------------------- */
set_version("1.0");
set_arglist("-name none -npix 100,100 -start 0.,0. -step 1.,1.");
init_session(argv, argc, &arglabel, &argval);

/* ----------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)     */
/* ----------------------------------------------------------------- */
get_argval(0, "%s", name);
sscanf(argval[1], "%d,%d", &(npix[0]), &(npix[1]));
sscanf(argval[2], "%lf,%lf", &(start[0]), &(start[1]));
sscanf(argval[3], "%lf,%lf", &(step[0]), &(step[1]));
```

Now we create the frame using the `create_frame()` routine. On input, we provide: the pointer to the `IMAGE2D` structure (`&frame`); the parameters of the frame (number of points, start coordinates and step sizes); the data type (SHORT, LONG, INT, FLOAT or DOUBLE); the identifier of the frame; the units of the axes (???).

```
/* ----------------------------------------------------------------- */
/* Creating the 2D image using the open_frame() routine              */
/* ----------------------------------------------------------------- */

sprintf(ident, "Test frame");
sprintf(units, "Units x-axis      Units y-axis    Units frame");

create_frame(&image, name, npix, start, step, FLOAT, ident, units);
```

Once the frame created, we fill it with an arbitrary function (namely a 2D Gaussian profile). The nested loops go from 0 (index of the first pixels) to `frame.nx - 1` or `frame.ny - 1`. Contrary to the case of 1D images, there is no pre-defined functions to compute the coordinates of a given pixel. They are therefore computed explicitely at each step. To write the pixel value and quality, we use the `WR_frame()` and `WR_qframe()` routines, respectively.

```
/* ----------------------------------------------------------------- */
/* Filling the frame with z=f(x,y)= exp(-(x**2 + y**2)/10.)          */
```

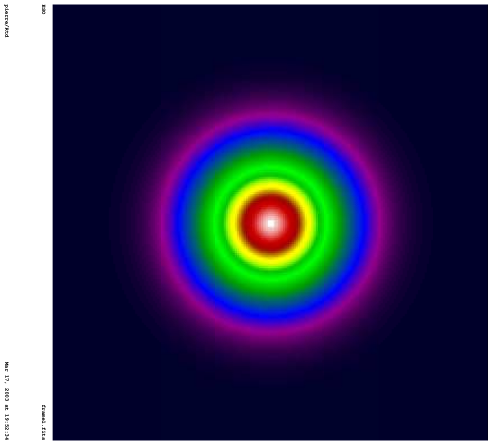

Figure 2: False color display of the frame frame1 produced by the FrameHandling1 program.

```
/* and set the pixel quality to 0 (OK)                               */
/* ----------------------------------------------------------------- */
for (i = 0 ; i < image.nx ; i++) {
for (j = 0 ; j < image.ny ; j++) {
x = image.startx + i * image.stepx;
y = image.starty + j * image.stepy;
z = exp(- (x * x + y * y) / 10.);
WR_frame(&image, i, j, z);
WR_qframe(&image, i, j, (long) 0);
}
}
```

Last we close the frame (everything will be written to disk) and exit the Euro3D session.

```
/* ----------------------------------------------------------------- */
/* Closing the 2D image                                              */
/* ----------------------------------------------------------------- */
close_frame(&image);

/* ----------------------------------------------------------------- */
/* Session end                                                       */
/* ----------------------------------------------------------------- */
exit_session(0);
}
```

The program can be executed by typing:

`$(E3D_PATH)/E3D_CookBook-1.0/user/bin/FrameHandling1 -name frame1 -npix 200,200 -start -10,-10 -step 0.1,0.1`

The messages on the screen will be minimalist as we do not print any message in the program:

```
Program: FrameHandling1  Version 1.0-6.0       Obs. de Lyon
End of FrameHandling1
```

The image produced by the program is displayed in Fig. 5.3. If the same command is executed a second time, the user will be prompted wether to overwrite the frame. To avoid that, it is possible to specify the `-noask` option on input.

By default the frame is created in the format specified in the `E3D_DEFAULT_FMT` environment variable. To change that, it is possible to specify the `-outputformat` on input.

5.4 Example program #2: opening and reading a 2D image (frame)

In this second example, we show how to open an existing 2D image and read its data.

```
/* ================================================================= */
/*!
\file       FrameHandling2.c
\copyright  (c) 2003 - Euro3D
\date       15/03/2003
\author     Pierre Ferruit
\version    1.0 15/03/2003 PF Creation
\keywords   Frame Handling, Cookbooks
\brief      Frame handling example file #2

Example file provided with the documentation of the
Euro3D libraries (v1.0). This file provides an
example of how to open and read a 2D image
within the Euro3D environment.

*/
/* ================================================================= */

#include        <math.h>
#include        <IFU_io.h>


/* ================================================================= */
/*!
\page FrameHandling2 FrameHandling2

Brief description.

\param [\c -name] name of the input image
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================= */

int  main(int argc, char **argv)
{
char      **argval;             /* input arguments of the program  */
char      **arglabel;           /* corresponding labels            */

IMAGE2D   image;                /* 2D image structure              */
char      name[81];             /* name of the image               */

long      i, j, n;              /* work variables                  */
float     value;                /* work variables                  */
```

```
char      msg_txt[161];          /* message text                        */
```

The program has a single input argument (the name of the input frame).

```
/* ---------------------------------------------------------------- */
/* Initialization of the Euro3D session                             */
/* ---------------------------------------------------------------- */
set_version("1.0");
set_arglist("-name none");
init_session(argv, argc, &arglabel, &argval);

/* ---------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)     */
/* ---------------------------------------------------------------- */
get_argval(0, "%s", name);
```

Then, we open the frame in read-only mode as we do not need to modify its contents. Other possible modes are "Output" and "IO".

```
/* ---------------------------------------------------------------- */
/* Opening the frame (in read-only mode)                            */
/* ---------------------------------------------------------------- */
open_frame(&image, name, "Input");
```

Then we compute the average of the 2D image. The pixel values and qualities are read using the `RD_frame()` and `RD_qframe()` routines, respectively.

```
/* ---------------------------------------------------------------- */
/* Computing the average using only pixels with quality flag        */
/* of 0 (OK)                                                        */
/* ---------------------------------------------------------------- */
value = (float) 0;
n = (long) 0;

for (i = 0 ; i < image.nx ; i++) {
for (j = 0 ; j < image.ny ; j++) {
if (RD_qframe(&image,i, j) == (long) 0) {
value += RD_frame(&image, i, j);
n++;
}
}
}

if (n != (long) 0) {
value /= n;
sprintf(msg_txt, "Average value of the image: %12.6e", value);
print_msg(msg_txt);
}
else {
print_msg("No valid point in the image.");
}
```

Last, we close the frame and exit the session.

```
/* ---------------------------------------------------------------- */
/* Closing the image                                                */
/* ---------------------------------------------------------------- */
close_frame(&image);

/* ---------------------------------------------------------------- */
/* Session end                                                      */
/* ---------------------------------------------------------------- */
exit_session(0);
}
```

The program can be executed by typing (here we assume that the program `FrameHandling1` described in the previous section has been executed in the same directory and that the frame `frame1.fits` already exists):

```
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/FrameHandling2 -name frame1
```

The messages on the screen will be the following:

```
Program: FrameHandling2  Version 1.0-6.0       Obs. de Lyon
Average value of the image: 7.853643e-02
End of FrameHandling2
```

If the input frame does not exist (e.g. if we try to open the frame `frame2.fits`), the program will automatically exit with an error message:

```
Program: FrameHandling2  Version 1.0-6.0       Obs. de Lyon
ERROR: FATAL error from routine open_frame: frame ./frame2.fits
ERROR: Error opening file

FrameHandling2 aborted - status code : -10
```

It is possible to avoid this automated behavior using the `set_error_level()` routine (see Sect. 10) and by analizing the return value of the library routines (this is not done in these example programs).

5.5 FAQ

6 Manipulating 3D images (cubes)

6.1 The IMAGE3D structure

In the following, we list the most important fields of the IMAGE3D structure (i.e. those a regular user may have to access while developing programs under the Euro3D environment). It is declared in the `items.h` include file of the `E3D_io_LCL-1.0a` library package.

```
typedef struct
{
char   name[lg_name+1];     /* name of the cube              */
...
char   history[lg_hist+1];  /* history of the cube           */
...
char   ident[lg_ident+1];   /* identifier of the cube        */
char   cunit[lg_unit+1];    /* units of the axes and values  */
int    nx;                  /* size along the x-axis         */
int    ny;                  /* size along the y-axis         */
int    nz;                  /* size along the z-axis         */
double startx;              /* x-axis start coordinate       */
double starty;              /* y-axis start coordinate       */
double startz;              /* z-axis start coordinate       */
double endx;                /* x-axis end coordinate         */
double endy;                /* y-axis end coordinate         */
double endz;                /* z-axis end coordinate         */
double stepx;               /* step size along the x-axis    */
double stepy;               /* step size along the y-axis    */
double stepz;               /* step size along the z-axis    */
double min;                 /* min & max of intensity        */
double max;
...
} IMAGE2D;
```

The `lg_name`, `lg_hist`, `lg_ident` and `lg_unit` lengths are actually aliases defined in one of the standard Euro3D include files (`gendef.h`) and are available through the mandatory `E3D_io.h` include file.

Warning: The `min` and `max` fields are only updated if you are *opening* an existing cube or if you make a call to the `cube_minmax()` routine provided with the libraries.

6.2 Key variables and routines

Routines to manipulate cubes (3D image)

`create_cube()`	create a new 3D image
`open_cube()`	open an existing 3D image
`close_cube()`	close a 3D image
`RD_cube()`	read the value of a given pixel
`RD_qcube()`	read the quality value of a given pixel
`WR_cube()`	write a value into a given pixel
`WR_qcube()`	write the quality of a given pixel

Warning: The first pixel of a 3D image corresponds to indices (0,0,0) and the last one to indices (cube.nx - 1 , cube.ny - 1, cube.nz - 1).

6.3 Example program #1: creating and filling a 3D image (cube)

In this first example, we will describe how to create a new 3D image, fill it and then close it. The example program `CubeHandling1.c` starts with the usual headers and includes the mandatory `E3D_io.h` include file.

```
/* ================================================================= */
/*!
\file       CubeHandling1.c
\copyright  (c) 2003 - Euro3D
\date       16/03/2003
\author     Pierre Ferruit
\version    1.0 16/03/2003 PF Creation
\keywords   3D Cube  Handling, Cookbooks
\brief      3D cube handling example file #1

Example file provided with the documentation of the
Euro3D libraries (v1.0). This file provide an
example of how to create and fill a 3D cube
within the Euro3D environment.
CAUTION : this is NOT a Euro3D data cube but a
standard 3D cube

*/
/* ================================================================= */

#include        <math.h>
#include        <IFU_io.h>


/* ================================================================= */
/*!
\page CubeHandling1 CubeHandling1

Brief description.

\param [\c -name] name of the output 3D cube
\param [\c -npix] number of pixels of the 3D cube
along the x-, y- and z-axes (defaulted to 100,100,100)
\param [\c -start] start values of the 3D cube
for the x-, y-  and z-axes (defaulted to 0.,0.,0.)
\param [\c -step] step size along the x-, y-  and z-axes
(defaulted to 1.,1.,1.)
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================= */
```

Among the variables we define a IMAGE3D-type one which will be used throughout the program.

```
int  main(int argc, char **argv)
{
char      **argval;             /* input arguments of the program  */
char      **arglabel;           /* corresponding labels            */

IMAGE3D   cube;                 /* 3D cube structure               */
char      name[81];             /* name of the spectrum            */
int       npix[3];              /* number of pixels                */
double    start[3];             /* start coordinates               */
double    step[3];              /* step sizes                      */
char      units[lg_unit+1];     /* unit string                     */
char      ident[lg_ident+1];    /* identifier string               */

long      i, j, k;              /* work variables                  */
float     x, y, z, value;       /* work variables                  */
```

We initialize the Euro3D session, including the list of input parameters for our program. Once this is done, we read the contents of the input arguments and store them in the corresponding variables.

```
/* ---------------------------------------------------------------- */
/* Initialization of the Euro3D session                             */
/* ---------------------------------------------------------------- */
set_version("1.0");
set_arglist("-name none -npix 100,100,100 -start 0.,0.,0. -step 1.,1.,1.");
init_session(argv, argc, &arglabel, &argval);

/* ---------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)     */
/* ---------------------------------------------------------------- */
get_argval(0, "%s", name);
sscanf(argval[1], "%d,%d,%d", &(npix[0]), &(npix[1]), &(npix[2]));
sscanf(argval[2], "%lf,%lf,%lf", &(start[0]), &(start[1]), &(start[2]));
sscanf(argval[3], "%lf,%lf,%lf", &(step[0]), &(step[1]), &(step[2]));
```

Now we create the cube using the `create_cube()` routine. On input, we provide: the pointer to the `IMAGE3D` structure (`&cube`); the parameters of the cube (number of points, start coordinates and step sizes); the data type (SHORT, LONG, INT, FLOAT or DOUBLE); the identifier of the cube; the units of the axes and values (???).

```
/* ---------------------------------------------------------------- */
/* Creating the 3D cube using the create_cube() routine             */
/* ---------------------------------------------------------------- */

sprintf(ident, "Test cube");
sprintf(units, "Units x-axis      Units y-axis    Units z-axis     Units frame");

create_cube(&cube, name, npix, start, step, FLOAT, ident, units);
```

Once the cube created, we fill it with an arbitrary function (namely a $f(x, y, z) = \sqrt{(}x^2 + y^2 + z^2))$. The nested loops go from 0 (index of the first pixels) to `cube.nx - 1`, `cube.ny - 1` or `cube.nz - 1`. Contrary to the case of 1D cubes, there is no pre-defined functions to compute the coordinates of a given pixel. They are therefore computed explicitely at each step. To write the pixel value and quality, we use the `WR_cube()` and `WR_qcube()` routines, respectively.

```
/* ---------------------------------------------------------------- */
/* Filling the cube with value=f(x,y,z)= sqrt(x^2 + y^2 + z^2)      */
/* and set the pixel quality to 0 (OK)                              */
/* ---------------------------------------------------------------- */
for (i = 0 ; i < cube.nx ; i++) {
for (j = 0 ; j < cube.ny ; j++) {
for (k = 0 ; k < cube.nz ; k++) {
x = cube.startx + i * cube.stepx;
y = cube.starty + j * cube.stepy;
z = cube.startz + k * cube.stepz;
value = sqrt(x * x + y * y + z * z);
WR_cube(&cube, i, j, k, value);
WR_qcube(&cube, i, j, k, (long) 0);
}
}
}
```

Last we close the cube (everything will be written to disk) and exit the Euro3D session.

```
/* ---------------------------------------------------------------- */
/* Closing the 3D cube                                             */
/* ---------------------------------------------------------------- */
close_cube(&cube);

/* ---------------------------------------------------------------- */
/* Session end                                                      */
/* ---------------------------------------------------------------- */
exit_session(0);
}
```

The program can be executed by typing:

`$(E3D_PATH)/E3D_CookBook-1.0/user/bin/CubeHandling1 -name cube1 -npix 200,200,200 -start -100,-100,-100 -step 1.,1.,1.`

The messages on the screen will be minimalist as we do not print any message in the program:

```
Program: CubeHandling1  Version 1.0-6.0
End of CubeHandling1
```

If the same command is executed a second time, the user will be prompted wether to overwrite the cube. To avoid that, it is possible to specify the `-noask` option on input.

By default the cube is created in the format specified in the `E3D_DEFAULT_FMT` environment variable. To change that, it is possible to specify the `-outputformat` on input.

6.4 Example program #2: opening and reading a 3D image (cube)

In this second example, we show how to open an existing 3D image and read its data.

```
/* ================================================================ */
```

```
/*!
\file       CubeHandling2.c
\copyright  (c) 2003 - Euro3D
\date       16/03/2003
\author     Pierre Ferruit
\version    1.0 16/03/2003 PF Creation
\keywords   Cube Handling, Cookbooks
\brief      Cube handling example file #2

Example file provided with the documentation of the
Euro3D libraries (v1.0). This file provides an
example of how to open and read a 3D cube
within the Euro3D environment.
CAUTION : this is NOT a Euro3D data cube but a
standard 3D cube

*/
/* ================================================================ */

#include        <math.h>
#include        <IFU_io.h>


/* ================================================================ */
/*!
\page CubeHandling2 CubeHandling2

Brief description.

\param [\c -name] name of the input cube
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================ */

int  main(int argc, char **argv)
{
char      **argval;              /* input arguments of the program  */
char      **arglabel;            /* corresponding labels            */

IMAGE3D   cube;                  /* 3D cube structure              */
char      name[81];              /* name of the cube               */

long      i, j, k, n;            /* work variables                  */
float     value;                 /* work variables                  */
char      msg_txt[161];          /* message text                    */
```

The program has a single input argument (the name of the input 3D image).

```
/* ---------------------------------------------------------------- */
/* Initialization of the Euro3D session                             */
/* ---------------------------------------------------------------- */
```

```
set_version("1.0");
set_arglist("-name none");
init_session(argv, argc, &arglabel, &argval);

/* -------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)   */
/* -------------------------------------------------------------- */
get_argval(0, "%s", name);
```

Then, we open the cube in read-only mode as we do not need to modify its contents. Other possible modes are ”Output” and ”IO”.

```
/* -------------------------------------------------------------- */
/* Opening the cube (in read-only mode)                           */
/* -------------------------------------------------------------- */
open_cube(&cube, name, "Input");
```

Then we compute the average of the 3D image. The pixel values and qualities are read using the `RD_cube()` and `RD_qcube()` routines, respectively.

```
/* -------------------------------------------------------------- */
/* Computing the average using only pixels with quality flag      */
/* of 0 (OK)                                                      */
/* -------------------------------------------------------------- */
value = (float) 0;
n = (long) 0;

for (i = 0 ; i < cube.nx ; i++) {
for (j = 0 ; j < cube.ny ; j++) {
for (k = 0 ; k < cube.nz ; k++) {
if (RD_qcube(&cube,i, j, k) == (long) 0) {
value += RD_cube(&cube, i, j, k);
n++;
}
}
}
}

if (n != (long) 0) {
value /= n;
sprintf(msg_txt, "Average value of the cube: %12.6e", value);
print_msg(msg_txt);
}
else {
print_msg("No valid point in the cube.");
}
```

Last, we close the cube and exit the session.

```
/* -------------------------------------------------------------- */
/* Closing the cube                                              */
/* -------------------------------------------------------------- */
close_cube(&cube);
```

```
/* ---------------------------------------------------------------- */
/* Session end                                                      */
/* ---------------------------------------------------------------- */
exit_session(0);
}
```

The program can be executed by typing (here we assume that the program `CubeHandling1` described in the previous section has been executed in the same directory and that the cube `cube1.fits` already exists):

`$(E3D_PATH)/E3D_CookBook-1.0/user/bin/CubeHandling2 -name cube1`

The messages on the screen will be the following:

```
Program: CubeHandling2  Version 1.0-6.0
Average value of the cube: 9.639745e+01
End of CubeHandling2
```

If the input cube does not exist (e.g. if we try to open the cube `cube2.fits`), the program will automatically exit with an error message:

```
Program: CubeHandling2  Version 1.0-6.0
ERROR: FATAL error from routine open_cube: cube cube2.fits
ERROR: Error opening file

CubeHandling2 aborted - status code : -10
```

It is possible to avoid this automated behavior using the `set_error_level()` routine (see Sect. 10) and by analizing the return value of the library routines (this is not done in these example programs).

6.5 FAQ

7 Manipulating tables

Warning: The current version of the Euro3D libraries (v1.0) does allow to manipulate vector columns (i.e. columns which elements are arrays of values). These functionalities, already available in the underlying `cfitsio` layer, will probably be present in the next version of the Euro3D libraries.

7.1 The TABLE structure

In the following, we list the most important fields of the TABLE structure (i.e. those a regular user may have to access while developing programs under the Euro3D environment). It is declared in the `items.h` include file of the `E3D_io_LCL-1.0a` library package.

```
typedef struct
{
char   path[lg_name+1];     /* path to the table            */
char   name[lg_name+1];     /* name of the table            */
...
char   history[lg_hist+1];  /* history of the cube          */
...
int   col;                  /* number of columns            */
int   row;                  /* selected number of row       */
int   allrow;               /* total number of row          */
int   select_flag;          /* selection flag               */
...
} TABLE;
```

The `lg_name` and `lg_hist` lengths are actually aliases defined in one of the standard Euro3D include files (`gendef.h`) and are available through the mandatory `E3D_io.h` include file.

It is possible to apply a selection in a table (see example#3) and the number of row (field `row` of the structure) will corresponded to the number of selected rows (and not to the total number of rows in the table) if the selection flag is activated using the `handle_select_flag()` routine.

7.2 Key variables and routines

Routines to manipulate tables

`create_table()`	create a new table
`open_table()`	open an existing table
`close_table()`	close a table
`handle_select_flag()`	enable or disable the selection on a table

Routines to manipulate columns

`create_col()`	create a column in a table
`get_col_ref()`	get the identifier of a column (number used to access the column)

Routines to read and write values in tables

`RD_tbl()`	read the table value for a given column and a given row
`WR_tbl()`	write a value in a table at a given column and a given row

Warning: The first row of a table corresponds to index 0 and the last one to index table.row - 1.

7.3 Example program #1: creating a table with two columns and filling them

In this first example, we will describe how to create a new table with two columns, fill them with the contents of a spectrum and then close table. The example program makes use of routines like `open_spec()` which are described in Sect. 4).

The program, `TableHandling1.c`, starts with the usual headers and includes the mandatory `E3D_io.h` include file.

```
/* ================================================================= */
/*!
\file       TableHandling1.c
\copyright  (c) 2003 - Euro3D
\date       17/03/2003
\author     Pierre Ferruit
\version    1.0 17/03/2003 PF Creation
\keywords   Table Handling, Cookbooks
\brief      Table handling example file #1

Example file provided with the documentation of the
Euro3D libraries (v6.0). This file provide an
example of how to create and fill a table
within the Euro3D environment.

*/
/* ================================================================= */

#include        <IFU_io.h>


/* ================================================================= */
/*!
\page TableHandling1 TableHandling1

Brief description.

\param [\c -is] name of the input spectrum
\param [\c -ot] name of the output table
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================= */

int  main(int argc, char **argv)
{
```

We have defined a TABLE variable as well as two `int` variables, which will receive the identifiers of the two columns. As we will open a spectrum to read its values, there is also a IMAGE1D structure.

```
char      **argval;              /* input arguments of the program  */
char      **arglabel;            /* corresponding labels            */

TABLE     table;                 /* table structure                 */
```

```
char      tname[lg_name+1];     /* name of the table             */
int       colID_x, colID_y;     /* column identifiers            */

IMAGE1D   spectrum;             /* spectrum structure            */
char      sname[lg_name+1];     /* name of the spectrum          */

long      i;                    /* work variable                 */
float     x,y;                  /* work variables                */
```

We initialize the Euro3D session, including the list of input parameters for our program. Once this is done, we read the contents of the input arguments and store them in the corresponding variables.

```
/* --------------------------------------------------------------- */
/* Initialization of the Euro3D session                            */
/* --------------------------------------------------------------- */
set_version("1.0");
set_arglist("-is none -ot none");
init_session(argv, argc, &arglabel, &argval);

/* --------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)    */
/* --------------------------------------------------------------- */
get_argval(0, "%s", sname);
get_argval(1, "%s", tname);
```

Now we open the spectrum from which the data will be extracted (when we will execute the program, we will use the spectrum produced by the `SpectrumHandling1` program described in Sect. 4).

```
/* --------------------------------------------------------------- */
/* Opening the input spectrum in read-only                         */
/* --------------------------------------------------------------- */
open_spec(&spectrum, sname, "Input");
```

The table is then created using the `create_table()` routine. On input, we provide the expected numbers of rows and columns. They are actually not necessary and can even be wrong. However, the low level I/O routines will optimize their work based on the values you provide. Therefore we strongly advise to provide theses values when available. The `'I'` character code stands for `Inquiry` and means that, if the table already exists, the user will be prompted before overwriting. It is also possible to use: `'N'` (for `New`) in which case the program will exit if the table already exists; `'W'` (for `Warning`) in which case a warning message will be printed before the table is overwritten; `'Q'` (for `Quiet` in which case the table is overwritten silently.

> **Warning:** In the `create_table()` routine, the creation code is a single character and must therefore be between *simple quotes* ('N', 'I', 'W' or 'Q').

Once the table created, we create the two columns `x` and `y` using the `create_col()` routine. In this example, the type of data is `FLOAT` (possible types are `CHAR`, `SHORT`, `INT`, `LONG`, `FLOAT` or `DOUBLE`). As for the creation of the table, we can decide of the behavior of the routine if a column with the same name already exists in the table. Possible values are: `'N'` (for `New`) in which case the program exits if the column already exists; `'R'` (for `Rewrite` in which case ??; `'O'` (for `Overwrite`) in which case the column is overwritten.

> **Warning:** As in the `create_table()` routine, the creation code of the `create_col()` routine is a single character and must therefore be between *simple quotes* ('N', 'R', 'O').

The `create_col()` routine returns the column ID which will be used to read or write data from or in these columns.

```
/* ---------------------------------------------------------------- */
/* Create the output table and its columns                          */
/* Number of rows : number of points in the input spectrum          */
/* Number of columns : 2 (xand y-axes)                              */
/* ---------------------------------------------------------------- */
create_table(&table, tname, spectrum.npts, (int) 2, 'I', "test table");
colID_x = create_col(&table, "x", FLOAT, 'N', "E12.6", "nm");
colID_y = create_col(&table, "y", FLOAT, 'N', "E12.6", "flux");
```

Now, we fill the columns with the content of the input spectrum using the `WR_tbl()` routine. Each time, we specify the row number and the column ID returned by the `create_col()` routine.

> **Warning:** When using the `WR_tbl()` routine, the value to be written in the table must be stored in a variable, the address of which will be given to the routine. The type of the variable must be consistent with that of the column. The compiler will not be able to detect a discrepancy at this level.

The index of the rows start at zero. In the first column (labelled `x`) we put the coordinates of the pixels, while the pixel value is written in the second column (labelled `y`).

> **Warning:** When creating a new table, the `table.row` field of the `TABLE` structure is not updated and should not be used.

```
/* ---------------------------------------------------------------- */
/* Copying the complete spectrum into the table                     */
/* ---------------------------------------------------------------- */
for (i = 0 ; i < spectrum.npts ; i++) {
x = coord_spec(&spectrum, i);
y = RD_spec(&spectrum, i);
WR_tbl(&table, i, colID_x, &x);
WR_tbl(&table, i, colID_y, &y);
}
```

Last we close the spectrum and the table before ending the session.

```
/* ---------------------------------------------------------------- */
/* Closing the spectrum and the table                               */
/* ---------------------------------------------------------------- */
close_spec(&spectrum);
close_table(&table);

/* ---------------------------------------------------------------- */
/* Session end                                                      */
/* ---------------------------------------------------------------- */
exit_session(0);
}
```

The program needs a spectrum on input. We can use the file `spectrum1.fits` created in Sect. 4. We need to type:

```
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/SpectrumHandling1 -name spectrum1 -npix 200 -start
```

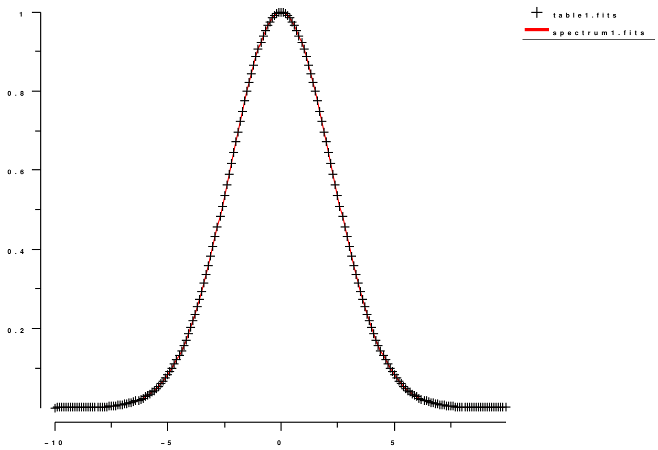

Figure 3: Plot of the contents of the table1.fits table (black symbols), with the spectrum spectrum1.fits overplotted (red solid line).

```
-10 -step 0.1 -noask
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/TableHandling1 -is spectrum1 -ot table1
```

The messages on the screen will be minimalist as we do not print any message in the course of the program:

```
   Program: TableHandling1  Version 1.0-6.0      Obs. de Lyon
   End of TableHandling1
```

A plot of the contents of the output table is displayed in Fig. 7.3. If the same command is executed a second time, the user will be prompted wether to overwrite the cube. To avoid that, it is possible to specify the `-noask` option on input.

By default the table is created in the format specified in the `E3D_DEFAULT_FMT` environment variable. To change that, it is possible to specify the `-outputformat` on input.

7.4 Example program #2: opening and reading a table

In this second example, we show how to open an existing table and to read data in it in order to compute the average of a column.

```
/* ================================================================ */
/*!
   \file        TableHandling2.c
   \copyright   (c) 2003 - Euro3D
   \date        17/03/2003
   \author      Pierre Ferruit
   \version     1.0 17/03/2003 PF Creation
   \keywords    Table Handling, Cookbooks
   \brief       Table handling example file #2

   Example file provided with the documentation of the
   Euro3D libraries (v6.0). This file provide an
   example of how to open and read a table
   within the Euro3D environment.

*/
/* ================================================================ */
```

```
#include        <IFU_io.h>


/* ================================================================= */
/*!
   \page TableHandling2 TableHandling2

   Brief description.

   \param [\c -is] name of the input table
   \param [\c -ic] name of the input column
   \return [\c int] Returns 0 if everything went smoothly,
   -1 otherwise.
*/
/* ================================================================= */

int  main(int argc, char **argv)
{
char      **argval;             /* input arguments of the program  */
char      **arglabel;           /* corresponding labels            */

TABLE     table;                /* table structure                 */
char      name[lg_name+1];      /* name of the table               */
char      colname[lg_name+1];   /* name of the column              */
int       colID;                /* column identifier               */

long      i;                    /* work variable                   */
float     total, value;         /* work variables                  */
char      msg_txt[161];         /* messages                        */
char      selection[161];       /* selection string                */
```

The program has a single input argument (the name of the input table).

```
   /* --------------------------------------------------------------- */
   /* Initialization of the Euro3D session                            */
   /* --------------------------------------------------------------- */
   set_version("1.0");
   set_arglist("-it none -ic none");
   init_session(argv, argc, &arglabel, &argval);

   /* --------------------------------------------------------------- */
   /* Analysis of the input arguments (no check of their validity)    */
   /* --------------------------------------------------------------- */
   get_argval(0, "%s", name);
   get_argval(1, "%s", colname);
```

Then, we open the table in read-only mode as we do not need to modify its contents. Other possible modes are "Output" and "IO".

Warning: The argument of the `open_table()` used to control the opening behavior of the routine is a *string* and not a single character as it is the case for the `create_table()` and `create_col()` routines. Therefore, it must be enclosed between *double quotes* ("I" or "Input", "O" or "Output", "IO").

As in the `TableHandling1` program, before opening the table we enable the selection (if the input table has an active selection, the program will only see the selected rows, see next section). Once the table is opened, we obtain the reference ID of the column we want to access and store it in the `colID` variable.

```
/* ---------------------------------------------------------------- */
/* Enabling selection on the table                                  */
/* Opening the input table (read-only) and get the identifier       */
/* of the input column                                              */
/* ---------------------------------------------------------------- */
handle_select_flag(&table, 'W', selection);
open_table(&table, name, "I");
colID = get_col_ref(&table, colname);
```

We make a loop on the (selected) rows to compute the average of the column. We use the `RD_tbl()` routine to access the table data. The loop runs from 0 to `table.row`. The latter contains the number of selected rows at the moment of the opening of the table.

```
/* ---------------------------------------------------------------- */
/* Computing the average value of the column                        */
/* ---------------------------------------------------------------- */
total = (float) 0.;

for (i = 0 ; i < table.row ; i++) {
RD_tbl(&table, i, colID, &value);
total += value;
}

if (table.row != (int) 0) {
total /= table.row;
sprintf(msg_txt,"Average value of column %s of table %s : %f",
colname, name, total);
print_msg(msg_txt);
}
else {
print_msg("Input table has no row.");
}
```

Last, we close the table and exit the session.

```
/* ---------------------------------------------------------------- */
/* Closing the spectrum and the table                               */
/* ---------------------------------------------------------------- */
close_table(&table);

/* ---------------------------------------------------------------- */
/* Session end                                                      */
/* ---------------------------------------------------------------- */
exit_session(0);
}
```

The program can be executed by typing (here we assume that the program `TableHandling1` described in the previous section has been executed in the same directory and that the table `table1.fits` already exists):

`$(E3D_PATH)/E3D_CookBook-1.0/user/bin/TableHandling2 -it table1 -ic y`

The messages on the screen will be the following:

```
Program: TableHandling2  Version 1.0-6.0       Obs. de Lyon
Average value of column y of table table1 : 0.280247
End of TableHandling2
```

If the input column does not exist (e.g. if we try to open the column `a`), the program will automatically exit with an error message:

```
ERROR: FATAL error from routine get_col_ref: table table1.fits column a
ERROR: Column does not exist

TableHandling2 aborted - status code : -60
```

It is possible to avoid this automated behavior using the `set_error_level()` routine (see Sect. 10) and by analizing the return value of the library routines (this is not done in these example programs).

7.5 Example program #3: playing with selections

Not yet available.

7.6 FAQ

None yet.

8 Manipulating EURO3D data cubes

This section gives several examples of programs that manipulate datacubes in the Euro3D format using the Euro3D I/O libraries.

Warning: The structure of the Euro3D format is complex, each *spectrum* beeing associated with one or more *spaxels*, which, in turn, point to *groups*. We strongly advise the reader to carefully read the Euro3D format definition document [1] before trying to understand these example programs of this section as it should make things much easier.

8.1 The SPAXEL structure

In the course of defining the Euro3D data format, we have defined the term `spaxel` which stands for SPAtial piXture ELement. The structure presented below describes such a spatial element. It is attached to a given spectrum identified by its unique Id (field `specId`) and corresponds to a given group Id (field `group`, see next subsection for a description of the `GROUP` structure). The spaxel has an Id (`spaxelId`) and a position `(xpos,ypos)`.

```
typedef struct
{
int specId;                   /* spectrum Id                              */
char spaxelId[lg_label+1];    /* spaxel Id                                */
int group;                    /* Id of the associated group               */
float xpos;                   /* x-axis position of the spaxel, in ??     */
float ypos;                   /* y-axis position of the spaxel, in ??     */
} SPAXEL;
```

Warning: A given spectrum (with a *unique* `specId` identifier) can correspond to more than one *spaxel* [1]. This association of *spaxels* is called a *compound*.

8.2 The GROUP structure

Each Euro3D datacube contains a list of `groups`, which describe the geometry of the `spaxels` and provide all the pieces of information needed for atmospheric dispersion correction (ADC, if applicable). There is a dedicated group structure `GROUP`.

```
typedef struct
{
/* ________ Shape and orientation related fields  __________________________ */
int groupId;                   /* group identifier                           */
char shape;                    /* shape identifier 'S', 'C', 'H' or 'R'      */
float size1;                   /* first spaxel size parameter, in ??         */
float size2;                   /* second spaxel size parameter, in ??        */
float angle;                   /* orientation of the spaxel, in ??           */
/* ________ ADC related fields ____________________________________________ */
float poswav;                  /* reference wavelength for the ADC           */
float airmass;                 /* airmass of the observation                 */
float parang;                  /* paralactic angle of the observation        */
float pressure;                /* atmospheric pressure in ??                 */
float temperature;             /* temperature during in ??                   */
```

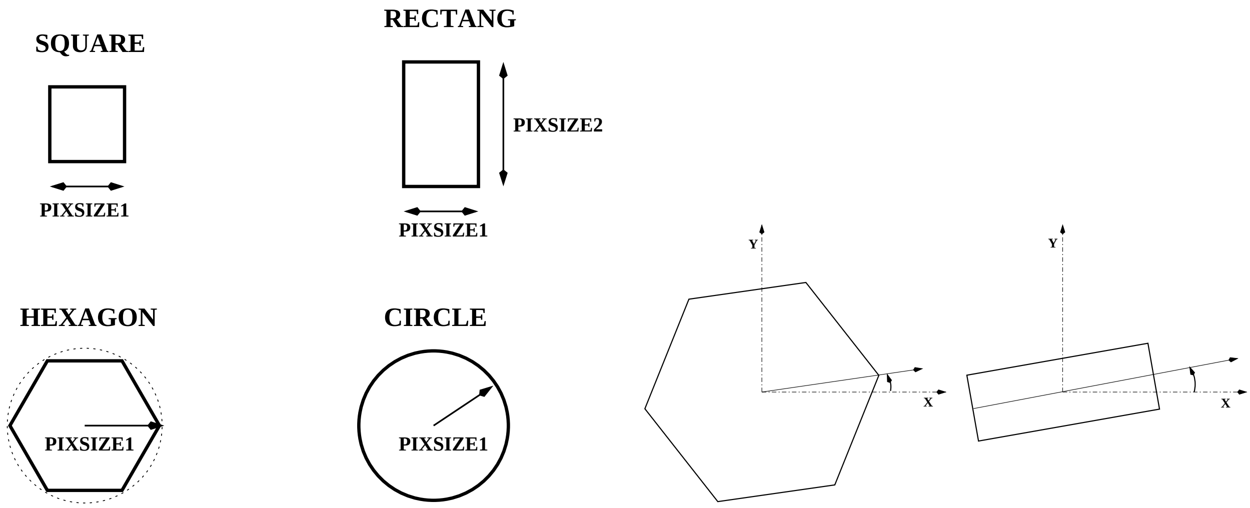

Figure 4: Left: drawing of the different spaxel shapes available with their parameters (PIXSIZE1 and PIXSIZE2 correspond to fields `size1` and `size2` of the `SPAXEL` structure, respectively. Right: drawing showing of the orientation of the spaxel is defined using the `angle` parameter.

```
float rel_humidity;           /* relative humidity in ??                    */
} GROUP;
```

Currently, four different shape identifiers are allowed: `'C'` for circular spaxels, in which case `size1` corresponds to the *radius* of the circular aperture and `size2` and `angle` are unused; `'S'` for square spaxels, in which case `size1` corresponds to the length of a side of the square aperture (in ??), `angle` corresponds to the orientation of the square aperture, while `size2` is unused; `'R'` for rectangular spaxels, in which case all three parameters `size1` (length of the first side), `size2` (length of the second side) and `angle` (angle made by the first side with the reference x-axis) are used (see Fig. 8.2); `'H'` for hexagonal spaxels, in which case `size1` corresponds to the radius of the circle *including* the hexagon (see Fig. 8.2, left panel), `size2` is unused and `angle` is the angle made by the hexagon with the reference x-axis (see Fig. 8.2 right pannel). As specified in [1] size must be specified in ?? and angles in ??.

8.3 The E3Dspec_desc and E3D_file structures

In the following, we list the most important fields of the `E3D_file` structure (i.e. those a regular user may have to access while developing programs under the Euro3D environment). It is declared in the `3D_items.h` include file of the `E3D_io_LCL-1.0a` library package. It contains arrays of `E3Dspec_desc` and `GROUP` structures. The `GROUP` structure is describde in previous subsection, while the `E3Dspec_desc` is presented later in this section.

```
typedef struct
{
char   name[lg_name+1];       /* name of the E3D file                         */
...
char   history[lg_hist+1];    /* contents of the history descriptor           */
...
char   ident[lg_ident+1];     /* contents of the identifier descriptor        */
char   cunit[lg_unit+1];      /* units of the spectra                         */
...
double start;                 /* reference start value (descriptor value)     */
```

```
double step;                  /* reference step size (descriptor value)     */
...
int    nbspec;                /* number of spectra in the datacube          */
...
E3Dspec_desc *signal;         /* basic description of the signal spectra    */
E3Dspec_desc *noise;          /* basic description of the noise spectra     */
...
int    ngroups;               /* number of groups                           */
GROUP  *groups;               /* description of the groups                  */
} E3D_file;
```

The `start` and `step` fields contain the reference start value and step size for the spectra of the Euro3D datacube. In the Euro3D file, they are stored in the descriptors `??START??` and `??STEP??`.

Warning: Although we define a *reference* start value for the Euro3D data cube, this does not mean that *all* the spectra must have the same start value. The Euro3D format only enforces the fact that all the spectra must be "aligned", i.e. they must have a start value satisfying the following relation:

$$start_{spectrum} = start_{reference} + k \times step_{reference}$$

where k is an integer and can be negative.

The number of spectra included in the Euro3D datacube is stored in the `nbspec` field of the structure and the basic parameters of the spectra are available in the arrays of `E3Dspec_desc` structures `*signal` (for the mandatory signal spectra) and `*noise` for the optional statistical error spectra.

The number of groups is stored in the `ngroups` field of the structure and their characteristics available in the array of `GROUP` structures (of size `ngroups`).

The `E3Dspec_desc` structure used to store the basic information about individual spectra has the following fields which may be of interest for the developers:

```
typedef struct
{
int      specId;              /* spectrum identifier                                 */
double   start;               /* start wavelength of the spectrum                    */
double   end;                 /* end wavelength fo the spectrum                      */
int      npix;                /* number of spectral pixels                           */
...
} E3Dspec_desc;
```

8.4 The SLICE structure

A Euro3D data cube can be accessed using two different ways. The first one is access it spectrum after spectrum (most common way, see e.g. example program 1 in this section). The second one is to access it using "monochromatic slices". The libraries provide routine for these two approaches and for the slices it was necessary to create a dedicated structure called `SLICES` (for the spectra, we use the regular `SPECTRUM` or `IMAGE1D` structure).

```
typedef struct
{
int    index;                 /* index of spectral pixel corresponding to the slice  */
int    npts;                  /* number of points in the slice                       */
int    *specId;               /* indices of the corresponding spectra                */
```

```
    ...
} SLICE;
```

A slice is simply a collection of "monochromatic" values (actually pairs as both the pixel value and the associated quality flag are stored) , each one corresponding to a single spectrum. The wavelength of a slice can be computed using the relation $\lambda_{reference} = start_{reference} + index \times step_{reference}$ where $start_{reference}$ and $step_{reference}$ are stored in the `start` and `step` fields of the `E3D_file` structure.

The identifiers of the spectra corresponding to each point of the slice are stored in the `specId` array of size `npts`.

Warning: The `SLICE` structure does not contain any `SPAXEL` or `GROUP` information and when creating a Euro3D datacube using slices they have to be added using the appropriate routines (see example program #4).

8.5 Key variables and routines

Routines to manipulate the Euro3D file

`create_E3D_file()`	create a new Euro3D file
`open_E3D_file()`	open an existing Euro3D file
`close_E3D_file()`	close a Euro3D file

Accessing data using SPECTRA

`init_new_E3D_spec()`	initialize a spectrum for the Euro3D data cube
`get_E3D_spec()`	get a spectrum from the Euro3D datacube
`put_E3D_spec()`	put a spectrum into the Euro3Ddatacube
`delete_E3D_spec()`	delete a spectrum and its associated spaxels from a Euro3D datacube (do not remove any of the associated group(s) as they can be shared by different spectra)

Manipulating SPAXELS

`get_E3D_spaxels()`	retrieve the spaxels associated to a given spectrum from a Euro3D datacube
`put_E3D_spaxels()`	put the spaxel information associated to a given spectrum into a Euro3D datacube

Manipulating GROUPS

`get_E3D_groups()`	retrieve the groups associated to a given spectrum from a Euro3D datacube
`put_E3D_groups()`	put the list of groups into a Euro3D datacube. Already existing groups (i.e. groups with the same group identifiers) will be overwritten.
`delete_E3D_group()`	delete a group from a Euro3D datacube

Manipulating SPECTRA + SPAXELS + GROUPS simultaneously

`get_E3D_row()`	retrieve a spectrum and its associated spaxel(s) and group(s) from a Euro3D datacube. Equivalent to successive calls to get_E3D_spec(), get_E3D_spaxels() and get_E3D_groups().
`put_E3D_row()`	put a spectrum and its associated spaxel(s) and group(s) into a Euro3D datacube. Equivalent to successive calls to put_E3D_spec(), put_E3D_spaxels() and put_E3D_groups().

Accessing data using SLICES

`init_new_E3D_slice()`	initialize a slice for a given Euro3D datacube (slice parameters will be obtained from this datacube)
`get_E3D_slice()`	retrieve a "monochromatic" slice from a Euro3D datacube
`put_E3D_slice()`	put a "monochromatic" slice into a Euro3D datacube
`alloc_slice_mem()`	initialize a slice (mainly array allocation) for a given set of parameters provided by the user
`free_slice_mem()`	free the memory allocated for a slice
`RD_slice()`	read data from a slice
`RD_qslice()`	read quality data from a slice
`WR_slice()`	write data into a slice
`WR_qslice()`	write quality data into a slice

Miscellaenous

`exist_spec_ID()`	checks if a given spectrum ID is present in a Euro3D datacube
`get_spectra_ID()`	returns an array containing the list of spectrum IDs present in a Euro3D datacube

8.6 Example program #1: creating and filling a Euro3D datacube with spectra (identical spectra with simple spaxels, single group)

In this first example, we show how to create a very simple datacube with a single type of spaxels (and therefore a single group entry).

Number and position of the spectra	400 spectra distributed over a 20×20 regular grid running from -10 to $+10$
Spaxels	all the spectra are associated to a single spaxel (no compound).
Characteristics of the spectra	all the spectra have the same start value (-1000) and number of pixels (200), corresponding to a common step size of 10.
Contents of the spectra	the spectra are filled with $$f(r,\lambda) = \frac{1}{1+r^2} \times \exp\left(-\frac{\lambda^2}{10}\right)$$ where $r = \sqrt{x^2+y^2}$ is the radius at the location of the spectrum.
Groups	single group corresponding to $0\farcs3$ hexagonal spaxels; ADC-related fields of the structure filled with arbitrary values.

The example program, `E3DHandling1.c`, starts with the usual headers and includes the mandatory `E3D_io.h` include file. On entry, it is possible to change the number of spectra (by changing the size and/or sampling of the grid over which they are distributed) and their characteristics (start, step and number of pixels).

```
/* ================================================================= */
/*!
\file       E3DHandling1.c
\copyright  (c) 2003 - Euro3D
\date       17/03/2003
\author     Pierre Ferruit
\version    1.0 17/03/2003 PF Creation
\keywords   E3D Cube  Handling, Cookbooks
\brief      E3D cube handling example file #1

Example file provided with the documentation of the
Euro3D libraries (v1.0). This file provides an
example of how to :
create and fill a E3D cube with identical spectra (same
start and length) and a single group entry (single spaxel
type).

*/
/* ================================================================= */

#include        <math.h>
#include        <IFU_io.h>
```

```
/* ================================================================ */
/*!
\page E3DHandling1 E3DHandling1

Brief description.

\param [\c -name] name of the output E3D cube
\param [\c -xaxis] npix,start,step
(defaulted to 20,-10,1)
\param [\c -yaxis] npix,start,step
(defaulted to 20,-10,1)
\param [\c -wave]  npix,start,step
(defaulted to 200,-1000,10)
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================ */

int  main(int argc, char **argv)
{
char      **argval;             /* input arguments of the program  */
char      **arglabel;           /* corresponding labels            */

E3D_file  cube;                 /* E3D file structure              */
char      name[lg_name+1];      /* name fo the E3D file            */
char      units[lg_unit+1];     /* unit string                     */
char      ident[lg_ident+1];    /* identifier string               */

long      npixx, npixy;         /* number of spaxels               */
double    startx, starty;       /* start values (x- and y-axes)    */
double    stepx, stepy;         /* sept sizes (x- and y-axes)      */

long      npixw;                /* number of spectral pixels       */
double    startw;               /* wavelength start value          */
double    stepw;                /* wavelength step size            */

SPAXEL    spaxel;               /* spaxel description structure    */
GROUP     group;                /* group description structure     */

SPECTRUM  signal;               /* signal spectrum structure       */
SPECTRUM  noise;                /* noise spectrum structure        */
int       specID;               /* spectrum identifier             */

long      i, j, k;              /* work variables                  */
float     x, y, w, norm;        /* work variables                  */
float     r, value, noisevalue; /* work variables                  */

/* ---------------------------------------------------------------- */
/* Initialization of the Euro3D session                             */
/* ---------------------------------------------------------------- */
set_version("1.0");
```

```
set_arglist("-name none -xaxis 20,-10,1 -yaxis 20,-10,1 -wave 200,-1000,10");
init_session(argv, argc, &arglabel, &argval);

/* ----------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)     */
/* ----------------------------------------------------------------- */
get_argval(0, "%s", name);
sscanf(argval[1], "%ld,%lf,%lf", &npixx, &startx, &stepx);
sscanf(argval[2], "%ld,%lf,%lf", &npixy, &starty, &stepy);
sscanf(argval[3], "%ld,%lf,%lf", &npixw, &startw, &stepw);
```

Then we need to initialize the `E3D_file` structure using the routine `create_E3D_file()` routine. On input, we provide the structure (`cube`), the name of the file (`name`), the characteristics of the spectra (`npixw`, the number of spectral pixels; `startw` the start value; `stepw` the step size), the type of the data (`FLOAT`, available values are CHAR, SHORT, INT, LONG, FLOAT and DOUBLE), the identifier string (`ident`) and the units of the two axes (`units`).

Warning: If on input to `create_E3D_file()`, the number of spectral pixels is ≥ 0 (as it is the case in our example), the input values of the start and step variable will be used as *references* for all the spectra of the datacube. If the number of spectral pixels is < 0 on input, then the input start and step values are ignored and *the first spectrum created will provide the reference values*.

```
/* ----------------------------------------------------------------- */
/* Initializing the Euro3D file using the create_E3D_file()        */
/* routine                                                           */
/* ----------------------------------------------------------------- */

sprintf(ident, "Test E3D file");
sprintf(units, "Units x-axis      Units y-axis");

create_E3D_file(&cube, name, npixw, startw, stepw, FLOAT, ident, units);
```

In this example, we have a single group common to all the spectra (e.g. as it would be the case for a single exposure of a given instrument). We have define a `GROUP` structure that we fill by hand with fairly arbitrary values. The spaxels are assumed hexagonal (shape code `'H'`) with a size of 0.3 and an orientation angle of 0. The ADC-related fields are filled with completely unrealistic values...

```
/* ----------------------------------------------------------------- */
/* Filling the GROUP information (arbitrary values, single group)   */
/* ----------------------------------------------------------------- */
group.groupId = (int) 0;
group.shape = 'H';              /* hexagonal spaxels               */
group.size1 = (float) 0.3;      /* in arcseconds                   */
group.size2 = (float) 0.;       /* unused                          */
group.angle = (float) 0.;       /* in degrees                      */

group.poswav = (float) 500.;    /* in nm                           */
group.airmass = (float) 1.345;  /* unitless                        */
group.parang = (float) 0.;      /* in degrees                      */
group.pressure = (float) 0.;    /* in Pa                           */
group.temperature = (float) 0.; /* in Celsius degrees              */
```

For this simple example, all the spectra will correspond to a single spaxel pointing toward the same group defined above. The `group` field of the `SPAXEL` structure is therefore initialized to the `groupId` of 0.

```
/* ----------------------------------------------------------------- */
/* Initializing the SPAXEL structure (just filling the group ID      */
/* with value 0)                                                     */
/* ----------------------------------------------------------------- */
spaxel.group = (int) 0;
```

Then comes the main loop in which the spectra are going to be created, filled and put into the E3D file. There are two nested loops running over the grid of positions for the spectra.

```
/* ----------------------------------------------------------------- */
/* Loop to fill the E3D file with the freshly created spectra        */
/* ----------------------------------------------------------------- */
for (i = 0 ; i < npixx ; i++) {
for (j = 0 ; j < npixy ; j++) {
```

At each position corresponds a spectrum and hence a single spectrumId (in this case we have choosen to index the ID on the position in the spatial grid...).

```
/* ........................................................... */
/* Spectrum ID                                                 */
/* ........................................................... */
specID = j + i * npixy;
```

We fill the `SPAXEL` structure with the relevant values (unique numerical spectrum ID, spaxel identifier string [does not have to be unique] and positions).

```
/* ........................................................... */
/* Filling the spaxel structure (single spaxel, no compoud)    */
/* The group field has already been filled with 0 (same        */
/* group for all the spectra in this example)                  */
/* ........................................................... */
x = startx + i * stepx;
y = starty + j * stepy;
spaxel.specId = specID;
spaxel.spaxelId = 'S';
spaxel.xpos = x;
spaxel.ypos = y;
```

We now initialize the two spectrum structure in which we will put the spectrum itself and its corresponding statistical error spectrum. We use a dedicated routine `init_new_E3D_spec()`. On input we provide the addresses of the Euro3D file (`&cube`) and spectrum (`&signal` or `&noise`) structures. As we have put -1 for the number of pixels, the routine will use the reference values of the start, step and number of pixels (i.e. those specified when creating the Euro3D file using the `create_E3D_file()`).

```
/* ........................................................... */
/* Initializing the spectra, filling them and putting them     */
/* into the cube                                               */
/* ........................................................... */
init_new_E3D_spec(&cube, &signal, (int) -1, (int) -1);
```

```
init_new_E3D_spec(&cube, &noise, (int) -1, (int) -1);
```

Now we fill the signal and noise spectra using the standard `WR_spec()` and `WR_qspec()` routines described in Sect. 4. The quality of all the spectral pixels is set to 0 (OK) in this example.

```
r = sqrt(x * x + y * y);
norm = 1. / (1. + r * r);
for (k = 0 ; k < npixw ; k++) {
w = startw + k * stepw;
value = norm * exp(- w * w / 100.);
WR_spec(&signal, k, value);
WR_qspec(&signal, k, (long) 0);
noisevalue = sqrt(value);
WR_spec(&noise, k, noisevalue);
WR_qspec(&noise, k, (long) 0);
}
```

The last step is to put the spectrum and its associated spaxel and group into the Euro3D datacube. This is done in a single step using the `put_E3D_row()` routine. In our case there is a single spaxel and a single group associated with the spectrum so the arguments `nbspaxel` and `nbgroup` are set to 1 (for more complex cases, the routine accepts arrays of spaxel or group structures).

```
put_E3D_row(&cube, specID, &signal, &noise, (int) 1, &spaxel, (int) 1, &group);
```

Note that the library contains routines which allow to put spectra (`put_E3D_spec()`), spaxels (`put_E3D_spaxels()`) and groups (`put_E3D_groups()`) independantly into the Euro3D file.

We now close the loops on the grid of spectra.

```
  }
  }
```

Last, we close the Euro3D file and exit the session.

```
/* ----------------------------------------------------------------- */
/* Closing the E3D file                                              */
/* ----------------------------------------------------------------- */
close_E3D_file(&cube);

/* ----------------------------------------------------------------- */
/* Session end                                                       */
/* ----------------------------------------------------------------- */
exit_session(0);
}
```

The program can be executed with its default input arguments by typing the following two lines (csh or tcsh environment is needed for the first command):

```
setenv E3D_DEFAULT_FMT euro3d
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/E3DHandling1 -name e3d1
```

The first line sets Euro3D format as the default input and output format for the Euro3D libraries. It is possible to skip this line by calling the program with the `-inputformat euro3d` and `-outputformat auro3d` options (actually only the output format specification is required in our case):

```
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/E3DHandling1 -name e3d1
```

```
                        -inputformat euro3d -outputformat euro3d
```

The output of the program is minimalist (as expected...):

```
Program: E3DHandling1  Version 1.0-6.0       Obs. de Lyon
End of E3DHandling1
```

If the same command is executed a second time, the user will be prompted wether to overwrite the Euro3D datacube. To avoid that, it is possible to specify the `-noask` option on input.

8.7 Example program #2: extracting a spectrum from a Euro3D datacube and displaying information about its associated spaxels and groups

This second program will extract a spectrum from a Euro3D data cube and list the basic properties of the associated spaxels and groups. Its goal is to show how to open an existing Euro3D datacube and access its data using spectra.

The example program, `E3DHandling2.c`, starts with the usual headers and includes the mandatory `E3D_io.h` include file. On entry, the user will specify the name of the input Euro3D datacube, the name of the output spectrum and the identifier of the spectrum to be extracted. By default the output spectrum will contain the signal spectrum but if the `-noise` option is present the statistical error spectrum will be extracted instead.

```
/* ================================================================== */
/*!
\file       E3DHandling2.c
\copyright  (c) 2003 - Euro3D
\date       17/03/2003
\author     Pierre Ferruit
\version    1.0 21/03/2003 PF Creation
\keywords   E3D Cube  Handling, Cookbooks
\brief      E3D cube handling example file #2

Example file provided with the documentation of the
Euro3D libraries (v1.0). This file provides an
example of how to :
open an existing Euro3D datacube and extract a spectrum from
it;
display the information about the spaxels and groups associated
to this spectrum;

*/
/* ================================================================== */

#include        <math.h>
#include        <IFU_io.h>


/* ================================================================== */
/*!
\page E3DHandling2 E3DHandling2

Brief description.
```

```
\param [\c -ic] name of the input E3D cube
\param [\c -os] name of the output spectrum
\param [\c -id] ID of the spectrum to be extracted (integer)
\param [\c -noise] flag argument specifying that it is the
statistical error spectrum which must be extracted
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================ */

int  main(int argc, char **argv)
{
char      **argval;              /* input arguments of the program  */
char      **arglabel;            /* corresponding labels            */

E3D_file  cube;                  /* E3D file structure              */
char      name[lg_name+1];       /* name fo the E3D file            */
SPECTRUM  cubspec;               /* extracted spectrum              */

SPECTRUM  spectrum;              /* ouput spectrum structure        */
char      specname[lg_name+1];   /* name of the output spectrum     */
int       specID;                /* spectrum identifier             */

int       NbSpaxels;             /* number of spaxels               */
SPAXEL    *Spaxels;              /* spaxel array                    */

int       NbGroups;              /* number of groups                */
GROUP     *Groups;               /* group array                     */

long      NoiseFlag;             /* =0 signal  =1 noise extracted   */

long      k;                     /* work variables                  */
char      err_txt[161];          /* error message string            */
char      msg_txt[161];          /* message string                  */

/* ---------------------------------------------------------------- */
/* Initialization of the Euro3D session                             */
/* ---------------------------------------------------------------- */
set_version("1.0");
set_arglist("-ic none -os none -id none -noise");
init_session(argv, argc, &arglabel, &argval);

/* ---------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)     */
/* ---------------------------------------------------------------- */
get_argval(0, "%s", name);
get_argval(1, "%s", specname);
get_argval(2, "%d", &specID);
NoiseFlag = is_true(argval[3]);
```

The first step in the program is to open the Euro3D datacube using the `open_E3D_file()` routine.

In our case, it is opened in read-only ("Input" mode. Other possible modes are "Output" and "IO".

```
/* ---------------------------------------------------------------- */
/* Opening the Euro3 file using open_E3D_file() in read-only mode  */
/* ---------------------------------------------------------------- */
open_E3D_file(&cube, name, "Input");
```

Before trying to retrieve the spectrum from the datacube, we check if a spectrum with the correct ID is actually present. This step is actually not necessary as the routine get_E3D_spec() would anyway exit with an error message if we try to use it to retrieve an unexisting spectrum. We have included it in the program to give an example of how to use the exist_spec_ID() routine.

```
/* ---------------------------------------------------------------- */
/* Checking if a spectrum with the correct identifier is present    */
/* in the Euro3D datacube                                           */
/* ---------------------------------------------------------------- */
if (exist_spec_ID(&cube, specID) < (int) 0) {
sprintf(err_txt, "Spectrum %d is not present in Euro3D datacube %s.",
specID, name);
print_error(err_txt);
exit_session(-1);
}
```

If the user has requested to extract a statistical error spectrum and as they are optional in the Euro3D format, we also need to check if the statistical error spectra are present in this Euro3D datacube. This is done using the exist_statistical_error() routine.

```
/* ---------------------------------------------------------------- */
/* If the user is asking for the statistical error spectrum         */
/* check that it exists                                             */
/* ---------------------------------------------------------------- */
if (NoiseFlag) {
if (exist_statistical_error(&cube) < (int) 0) {
print_error("Statistical error spectra are not present.");
exit_session(-1);
}
}
```

Then we display the number of spaxels associated to this spectrum and their basic characteristics. This information is obtained using the get_E3D_spaxels() routine. Note that, before continuing, we free the *Spaxels array which has been allocated by get_E3D_spaxels().

```
/* ---------------------------------------------------------------- */
/* Display spaxel information related to the spectrum               */
/* ---------------------------------------------------------------- */
NbSpaxels = get_E3D_spaxels(&cube, specID, &Spaxels);

sprintf(msg_txt,"Found %d spaxels associated with spectrum %d.",
NbSpaxels, specID);
print_msg(msg_txt);

for (k = 0 ; k < NbSpaxels ; k++) {
sprintf(msg_txt, "[%4ld] %c %8.4f %8.4f %d", k,
```

```
(Spaxels[k]).spaxelId, (Spaxels[k]).xpos,
(Spaxels[k]).ypos, (Spaxels[k]).group);
print_msg(msg_txt);
}

free(Spaxels);
```

We repeat the same operation but for the groups this time and using the `get_E3D_groups()` routine. We only display the shape information.

```
/* ----------------------------------------------------------------- */
/* Display shape information related to the spectrum                 */
/* ----------------------------------------------------------------- */
NbGroups = get_E3D_groups(&cube, specID, &Groups);

sprintf(msg_txt,"Found %d groups associated with spectrum %d.",
NbGroups, specID);
print_msg(msg_txt);

for (k = 0 ; k < NbGroups ; k++) {
sprintf(msg_txt, "[%4ld] %d %c %8.4f %8.4f %8.4f", k,
(Groups[k]).groupId, (Groups[k]).shape,
(Groups[k]).size1, (Groups[k]).size2,
(Groups[k]).angle);
print_msg(msg_txt);
}

free(Groups);
```

Now, we retrieve the signal or noise spectrum from the Euro3D datacube. It is possible to retrieve them simultaneously by replacing the `NULL` entry by a pointer on a `SPECTRUM` structure. Note that the extracted spectrum is not associated to any file on disk (contrary to what happens if the spectrum is created using `create_spec()` or opened with `open_spec()`; see Sect. 4).

```
/* ----------------------------------------------------------------- */
/* Retrieve the signal or noise spectrum from the Euro3D datacube    */
/* ----------------------------------------------------------------- */
if (NoiseFlag) {
get_E3D_spec(&cube, NULL, &cubspec, specID);
}
else {
get_E3D_spec(&cube, &cubspec, NULL, specID);
}
```

As `cubspec` has no real existence on disk, it is necessary to create a spectrum and to transfer the data into it. Note that we use the fields of the `cubsepc` structure as inputs to the `create_spec()` routine.

```
/* ----------------------------------------------------------------- */
/* Create the output spectrum, transfer the data and close it        */
/* ----------------------------------------------------------------- */
create_spec(&spectrum, specname, cubspec.npts, cubspec.start,
cubspec.step, cubspec.data_type, "extracted spectrum",
```

```
cubspec.cunit);

for (k = 0 ; k < cubspec.npts ; k++) {
WR_spec(&spectrum, k, RD_spec(&cubspec,k));
WR_qspec(&spectrum, k, RD_qspec(&cubspec,k));
}

close_spec(&spectrum);
```

The output spectrum has been created, filled and closed and before exiting the session we close the Euro3D file and free the memory of `cubspec`.

> **Warning:** The `get_E3D_spec()` routine does not associate any real file to the extracted spectrum (for that, it is equivalent to the `create_spec_mem()` routine described in Sect. 4). Therefore it should *not* be closed using the `close_spec()` routine but using the `free_spec_mem()` routine (as done in this example).

```
/* --------------------------------------------------------------- */
/* Closing the E3D file and freeing the memory of the cubspec      */
/* structure                                                       */
/* --------------------------------------------------------------- */
free_spec_mem(&cubspec);
close_E3D_file(&cube);

/* --------------------------------------------------------------- */
/* Session end                                                     */
/* --------------------------------------------------------------- */
exit_session(0);
}
```

The program can be executed by typing (here we assume that the program `E3DHandling1` described in the previous section has been executed in the same directory, i.e. that the Euro3D file `e3d1.fits` already exists):

```
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/E3DHandling2 -ic e3d1 -os spec1 -id 10
```

The messages on the screen will be the following:

```
Program: E3DHandling2  Version 1.0-6.0
Found 1 spaxels associated with spectrum 10.
[   0] S -10.0000   0.0000 0
Found 1 groups associated with spectrum 10.
[   0] 0 H   0.3000   0.0000   0.0000
End of E3DHandling2
```

If we specify an unknown spectrum ID on input (e.g. -1):

```
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/E3DHandling2 -ic e3d1 -os spec1 -id -1
```

The `exist_spec_ID()` routine will return a negative value and the program will exit after displaying the following error message:

```
Program: E3DHandling2  Version 1.0-6.0
ERROR: Spectrum -1 is not present in Euro3D datacube e3d1.
E3DHandling2 aborted - status code : -1
```

If the input Euro3D file does not exist (e.g. if we try to open the file `e3d2.fits`), the program will automatically exit with an error message:

```
Program: E3DHandling2  Version 1.0-6.0
ERROR: FATAL error from routine open_E3D_file: frame e3d2.fits
ERROR: Error opening file

E3DHandling2 aborted - status code : -10
```

It is possible to avoid this automated behavior using the `set_error_level()` routine (see Sect. 10) and by analizing the return value of the library routines (this is not done in these example programs).

Also, if the same command is executed a second time, the user will be prompted wether to overwrite the output spectrum. To avoid that, it is possible to specify the `-noask` option on input.

8.8 Example program #3: deleting a spectrum from a Euro3D file

This third program will delete a spectrum from a Euro3D data cube. Its goal is to show how to open an existing Euro3D datacube, delete a spectrum and check that now group is left "detached".

The example program, `E3DHandling3.c`, starts with the usual headers and includes the mandatory `E3D_io.h` include file. On entry, the user will specify the name of the input Euro3D datacube and the identifier of the spectrum to be deleted.

```
/* ================================================================ */
/*!
\file       E3DHandling3.c
\copyright  (c) 2003 - Euro3D
\date       17/03/2003
\author     Pierre Ferruit
\version    1.0 24/03/2003 PF Creation
\keywords   E3D Cube  Handling, Cookbooks
\brief      E3D cube handling example file #3

Example file provided with the documentation of the
Euro3D libraries (v1.0). This file provides an
example of how to :
open an existing Euro3D datacube and delete a
given spectrum

*/
/* ================================================================ */

#include        <math.h>
#include        <IFU_io.h>

/* ================================================================ */
/*!
\page E3DHandling3 E3DHandling3

Brief description.

\param [\c -ic] name of the input E3D cube
```

```
\param [\c -id] ID of the spectrum to be deleted (integer)
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================ */

int  main(int argc, char **argv)
{
char      **argval;              /* input arguments of the program  */
char      **arglabel;            /* corresponding labels            */

E3D_file  cube;                  /* E3D file structure              */
char      name[lg_name+1];       /* name fo the E3D file            */
int       specID;                /* spectrum identifier             */

int       NbGroups;              /* number of groups                */
GROUP     *Groups;               /* group array                     */
int       CurrentNbGroups;       /* number of groups                */
GROUP     *CurrentGroups;        /* group array                     */

int       *IDs;                  /* work array                      */
short     *ToBeDeletedFlag;      /* work array                      */
long      i,j,k;                 /* work variables                  */

/* ---------------------------------------------------------------- */
/* Initialization of the Euro3D session                             */
/* ---------------------------------------------------------------- */
set_version("1.0");
set_arglist("-ic none -id none");
init_session(argv, argc, &arglabel, &argval);

/* ---------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)     */
/* ---------------------------------------------------------------- */
get_argval(0, "%s", name);
get_argval(1, "%d", &specID);
```

We open the Euro3D datacube in input/output mode (as we are going to delete a spectrum, we need to be able to modify the datacube).

```
/* ---------------------------------------------------------------- */
/* Opening the Euro3 file using open_E3D_file() in I/O mode        */
/* ---------------------------------------------------------------- */
open_E3D_file(&cube, name, "IO");
```

When we delete a spectrum, there is the possibility that we leave an "unsused" group. To be able to check if it is the case we store the list of the groups associated with the spectrum to be deleted. For that, we use the `get_E3D_groups()` routine.

```
/* ---------------------------------------------------------------- */
/* Get the list of the groups associated to the spectrum            */
/* ---------------------------------------------------------------- */
```

```
NbGroups = get_E3D_groups(&cube, specID, &Groups);
```

Now we delete the spectrum using the `delete_E3D_spec()` routine.

Warning: The `delete_E3D_spec()` routine will delete a spectrum *and* its associated spaxels *but* will not delete any group as they can be shared by different spaxels.

```
/* ---------------------------------------------------------------- */
/* Delete the spectrum                                              */
/* ---------------------------------------------------------------- */
delete_E3D_spec(&cube, specID);
```

At this stage, we could close the Euro3D datacube and exit the program and the spectrum would have been deleted. However, we need to make sure that we do not leave any "detached" groups. We first allocate memory for an array (`ToBeDeletedFlag[]`) to flag the groups to be deleted. After the allocation, we check that everything went well and exit otherwise.

```
/* ---------------------------------------------------------------- */
/* Now check that we do not have detached groups                    */
/* ---------------------------------------------------------------- */
ToBeDeletedFlag = (short *) calloc(NbGroups, sizeof(short));
if (ToBeDeletedFlag == NULL) {
print_error("Failed to allocate memory for the ToBeDeletedFlag[]");
print_error("array.");
exit_session(ERR_ALLOC);
}
```

We need the list of the spectrum IDs and we obtain them using the `get_spectra_ID()` routine.

```
IDs = (int *) malloc(cube.nbspec * sizeof(int));
if (IDs == NULL) {
print_error("Failed to allocate memory for the IDs[]");
print_error("array.");
exit_session(ERR_ALLOC);
}

get_spectra_ID(&cube, IDs);
```

Now we make a loop over all the spectra, retrieve the associated groups (using the `get_E3D_groups()` routine) and check if there is any overlap between the two sets of groups. If we find an overlap, the corresponding entry of the `ToBeDeletedFlag[]` array is set to -1 and the group will not be deleted.

Warning: The `get_E3D_groups()` routine allocates memory for the `GROUP` array and it is necessary to free it before the next step of the loop.

Once the loop ended, we free the `IDs` array which was allocated by the `get_spectra_ID()` routine.

```
for (i = 0 ; i < cube.nbspec ; i++) {
/* ............................................................ */
/* Get the groups associated with the current spectrum          */
/* ............................................................ */
CurrentNbGroups = get_E3D_groups(&cube, IDs[i], &CurrentGroups);

/* ............................................................ */
```

```
/* Check if the groups are still used                              */
/* ............................................................... */
for (j = 0 ; j < CurrentNbGroups ; j++) {
for (k = 0 ; k < NbGroups ; k++) {
if ((CurrentGroups[j]).groupId == (Groups[k]).groupId) {
ToBeDeletedFlag[k] = (short) -1;
}
}
}
/* ............................................................... */
/* Free memory                                                     */
/* ............................................................... */
free(CurrentGroups);
}

free(IDs);
```

We delete the "detached" groups (i.e. those with a flag = 0).

```
/* --------------------------------------------------------------- */
/* Delete any detached group                                       */
/* --------------------------------------------------------------- */
for (k = 0 ; k < NbGroups ; k++) {
if (ToBeDeletedFlag[k] == (short) 0) {
delete_E3D_group(&cube, (Groups[k]).groupId);
}
}
```

Everything is done, we free memory, close the Euro3D datacube and exit the Euro3D session.

```
/* --------------------------------------------------------------- */
/* Closing the E3D file                                            */
/* --------------------------------------------------------------- */
free(ToBeDeletedFlag);
free(Groups);
close_E3D_file(&cube);

/* --------------------------------------------------------------- */
/* Session end                                                     */
/* --------------------------------------------------------------- */
exit_session(0);

}
```

The program can be executed by typing (here we assume that the program `E3DHandling1` described in the previous section has been executed in the same directory, i.e. that the Euro3D file `e3d1.fits` already exists):

`$(E3D_PATH)/E3D_CookBook-1.0/user/bin/E3DHandling3 -ic e3d1 -id 10`

The messages on the screen will be the following:

```
Program: E3DHandling3  Version 1.0-6.0
End of E3DHandling3
```

After that, we run the program a second time and as the spectrum has been deleted, we get the following error messages on exit:

```
Program: E3DHandling3  Version 1.0-6.0
ERROR: FATAL error from routine get_E3D_groups: spec 10
ERROR: Image does not exist

E3DHandling3 aborted - status code : -44
```

8.9 Example program #4: creating and filling a E3D datacube with slices

Example program #4 (`E3DHandling4`) creates a very simple Euro3D datacube similar to one created by example program #1 (`E3DHandling1`. This cube will be made of identical spectra (same start, step and number of pixels) each associated to a single spaxel (no compound) and a unique group (e.g. datacube originating from a single observation). Contrary to what we did in example #1, the distribution of the spectra does not follow a regular grid this time. The properties of the output Euro3D datacube are listed in the table below.

Number and position of the spectra	100 spectra distributed over a spiral
Spaxels	all the spectra are associated to a single spaxel (no compound) of position $$r(i) = \log(i+1) \quad \theta(i) = i \times \frac{\pi}{10}$$ where i is the spectrum ID (running from 0 to 99).
Characteristics of the spectra	all the spectra have the same start value (-1000) and number of pixels (200), corresponding to a common step size of 10.
Contents of the spectra	the spectra are filled with $$f(r,\lambda) = \frac{1}{1+r^2} \times \exp\left(-\frac{\lambda^2}{10}\right)$$ where $r = \sqrt{x^2+y^2}$ is the radius at the location of the spectrum.
Groups	single group corresponding to $0''.3$ hexagonal spaxels; ADC-related fields of the structure filled with arbitrary values.

The example program, `E3DHandling4.c`, starts with the usual headers and includes the mandatory `E3D_io.h` include file. On entry, it is possible to specify the name of the output Euro3D datacube (`-name`), the number of spectra (`-nbspec`) and their characteristics (`-wave` ; start, step and number of pixels).

```
/* ================================================================== */
/*!
\file        E3DHandling4.c
\copyright   (c) 2003 - Euro3D
\date        21/03/2003
\author      Pierre Ferruit
\version     1.0 17/03/2003 PF Creation
\keywords    E3D Cube  Handling, Cookbooks
```

```
\brief      E3D cube handling example file #4

Example file provided with the documentation of the
Euro3D libraries (v1.0). This file provides an
example of how to :
- create and fill a E3D cube using slices

*/
/* ================================================================= */

#include       <math.h>
#include       <IFU_io.h>


/* ================================================================= */
/*!
\page E3DHandling4 E3DHandling4

Brief description.

\param [\c -name] name of the output E3D cube
\param [\c -nbspec] number of spectra
\param [\c -wave]  npix,start,step
(defaulted to 200,-1000,10)
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================= */

int  main(int argc, char **argv)
{
char      **argval;             /* input arguments of the program  */
char      **arglabel;           /* corresponding labels            */

E3D_file  cube;                 /* E3D file structure              */
char      name[lg_name+1];      /* name fo the E3D file            */
char      units[lg_unit+1];     /* unit string                     */
char      ident[lg_ident+1];    /* identifier string               */

long      npixw;                /* number of spectral pixels       */
double    startw;               /* wavelength start value          */
double    stepw;                /* wavelength step size            */

long      NbSpec;               /* number of spectra               */
SPAXEL    spaxel;               /* spaxel description structure    */
GROUP     group;                /* group description structure     */
SLICE     signalslice;          /* signal slice structure          */
SLICE     noiseslice;           /* noise slice structure           */
long      i, k;                 /* work variables                  */
float     x, y, w, norm;        /* work variables                  */
float     r, theta;             /* work variables                  */
```

```
float     value, noisevalue;     /* work variables                  */
SPAXEL    *currentspaxel;        /* work variable                   */
int       NbSpaxels;             /* work variable                   */

/* -------------------------------------------------------------- */
/* Initialization of the Euro3D session                           */
/* -------------------------------------------------------------- */
set_version("1.0");
set_arglist("-name none -nbspec none -wave 200,-1000,10");
init_session(argv, argc, &arglabel, &argval);

/* -------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)   */
/* -------------------------------------------------------------- */
get_argval(0, "%s", name);
get_argval(1, "%ld", &NbSpec);
sscanf(argval[2], "%ld,%lf,%lf", &npixw, &startw, &stepw);
```

The first step (as in example #1) is to create the Euro3D file using the `create_E3D_file()` routine. We provide on input the address of a `E3D_file` structure (`&cube`), the name of the file (`name`), the characteritics of the spectra (`npixw` number of spectral pixels; `startw` reference start value of the spectra; `stepw` reference step value of the spectra), the datatype (`FLOAT`, could also be SHORT, INT, LONG, DOUBLE) and identifier and unit strings (`ident` and `unit`).

```
/* -------------------------------------------------------------- */
/* Initializing the Euro3D file using the create_E3D_file()      */
/* routine                                                        */
/* -------------------------------------------------------------- */

sprintf(ident, "Test E3D file");
sprintf(units, "Units x-axis      Units y-axis");

create_E3D_file(&cube, name, npixw, startw, stepw, FLOAT, ident, units);
```

At this stage, the Euro3D datacube is empty. We need first to initialize it with a list of spectrum IDs. In this example, this is done using dummy signal and noise slices. These two slices are prepared using the `alloc_slice_mem()` routines which will allocate the memory for the various arrays present in the `SLICE` structure (on input we specify the number of spectra to be accomodated `NbSpec`). Then we fill the `specId` array with identifiers (simply the index of the spectrum in our case). Then we put the dummy slices in the datacube using the `put_E3D_slice()` routine.

Warning: The memory of the arrays of the `signalslice` and `noiseslice`, initially allocated by the create_slice_mem() routine in our example, is *automatically freed* by the `put_E3D_slice()` routine.

```
/* -------------------------------------------------------------- */
/* Initializing the Euro3D file with a dummy slice which will     */
/* be overwritten in the main loop.                               */
/* -------------------------------------------------------------- */

alloc_slice_mem(&signalslice, NbSpec);
alloc_slice_mem(&noiseslice, NbSpec);
```

```
for (i = 0 ; i < NbSpec ; i++) {
(signalslice.specId)[i] = (int) i;
(noiseslice.specId)[i] = (int) i;
}

put_E3D_slice(&cube, &signalslice, &noiseslice, (int) 0);
```

Next step is to provide the spaxel information associated to each spectrum. We make a loop on the spectrum IDs and inject this information in the Euro3D datacube using the `put_E3D_spaxels()` routine. Note that all the spaxels point toward a single group of identifier 0.

```
/* ---------------------------------------------------------------- */
/* Adding the spaxel information                                    */
/* ---------------------------------------------------------------- */
spaxel.group = (int) 0;

for (i = 0 ; i < NbSpec ; i++) {
spaxel.spaxelId = i + 0x30; /* generates corresponding ASCII char */
r = log((float) (i+1));
theta = M_PI * (float) i / 10.;
spaxel.xpos = r * sin(theta);
spaxel.ypos = r * cos(theta);

put_E3D_spaxels(&cube, (int) i, (int) 1, &spaxel);
}
```

We also add the group information using the `put_E3D_groups()` routine. In our case, we have assumed that there was a single group.

```
/* ---------------------------------------------------------------- */
/* Adding the group information (single group, arbitrary values)    */
/* ---------------------------------------------------------------- */
group.groupId = (int) 0;
group.shape = 'H';                /* hexagonal spaxels              */
group.size1 = (float) 0.3;        /* in arcseconds                  */
group.size2 = (float) 0.;         /* unused                         */
group.angle = (float) 0.;         /* in degrees                     */

group.poswav = (float) 500.;      /* in nm                          */
group.airmass = (float) 1.345;    /* unitless                       */
group.parang = (float) 0.;        /* in degrees                     */
group.pressure = (float) 0.;      /* in Pa                          */
group.temperature = (float) 0.;   /* in Celsius degrees             */

put_E3D_groups(&cube, (int) 1, &group);
```

Now, we are ready to fill the Euro3D datacube. There will be `npixw` slices in this datacube and we start the main loop.

```
/* ---------------------------------------------------------------- */
/* Loop to fill the E3D file using slices                           */
/* (loop on the spectral pixels)                                    */
/* ---------------------------------------------------------------- */
```

```
for (k = 0 ; k < npixw ; k++) {
```

At each step, we initialize the `signalslice` and `noiseslice` structures. We use the `init_new_E3D_slice()` routine which will automatically use the parameters of the Euro3D datacube (i.e. the number of spectra `cube.nbspec`). Ilt will allocate the memory needed to accomodate these spectra and will fill the `specId` array of the `SLICE` structures with those of the datacubes.

We set the index of the slices to the index of the spectral pixel.

```
/* ............................................................ */
/* Initializing the slices                                      */
/* ............................................................ */
init_new_E3D_slice(&cube, &signalslice);
init_new_E3D_slice(&cube, &noiseslice);

signalslice.index = (int) k;
noiseslice.index = (int) k;
```

Once the slices inititialized, we fill them by making a loop on the number of points in the slices.

```
/* ............................................................ */
/* Filling the slices                                           */
/* ............................................................ */
for (i = 0 ; i < signalslice.npts ; i++) {
```

We retrieve the position information of a given spectrum using the associated spaxel structure. We could have computed `x` and `y` directly (we did that at the beginning of the program when creating the spaxels...), but wanted to show an example of use of the `get_E3D_spaxels()` routine.

Warning: The `get_E3D_spaxels()` routine will allocate memory for the array of spaxels corresponding to a given spectrum identifier. It is important to think to free this memory once it is not useful anymore.

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/* Getting the position information from the spaxels           */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
NbSpaxels = get_E3D_spaxels(&cube, (signalslice.specId)[i],
&currentspaxel);
x = spaxel.xpos;
y = spaxel.ypos;
free(currentspaxel);
```

Compute the signal and statistical error values of the slice at this given point and write them into the slice data arrays using the dedicated `WR_slice()` and `WR_qslice()` routines.

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/* Computing the datacube signal and noise values at this      */
/* position                                                     */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
r = sqrt(x * x + y * y);
norm = 1. / (1. + r * r);
w = startw + k * stepw;
value = norm * exp(- w * w / 100.);
WR_slice(&signalslice, i, value);
WR_qslice(&signalslice, i, (long) 0);
```

```
noisevalue = sqrt(value);
WR_slice(&noiseslice, i, noisevalue);
WR_qspec(&noiseslice, i, (long) 0);
```

We close the loop on the spectra.

```
}
```

We put the signal and statistical error slices in the Euro3D datacube using the `put_E3D_slice()` routine. The index of the slices is the index of the spectral pixel `k`.

Warning: The `put_E3D_slice()` routine will *automatically free* the memory allocated for the arrays of the input signal and/or statistical error structures.

```
/* ........................................................... */
/* Inserting them into the Euro3D datacube                     */
/* ........................................................... */
put_E3D_slice(&cube, &signalslice, &noiseslice, (int) k);
}
```

The cube is now completely filled. We close the Euro3D datacube and exit the Euro3D session.

```
/* ------------------------------------------------------------- */
/* Closing the E3D file                                          */
/* ------------------------------------------------------------- */
close_E3D_file(&cube);

/* ------------------------------------------------------------- */
/* Session end                                                   */
/* ------------------------------------------------------------- */
exit_session(0);
}
```

We can use this example program `E3DHandling4` program to create a e3d4.fits (`-name` option) Euro3D datacube containing 100 spectra (`-nbspec`) option). We do not specify the `-wave` option and the default values will be used (200 spectral pixels, start value of -1000 and stepsize of 10):

```
$(E3D_PATH)/E3D_CookBook-1.0/user/bin/E3DHandling4 -name e3d4 -nbspec 100
```

The messages on the screen will be the following:

```
Program: E3DHandling4  Version 1.0-6.0
End of E3DHandling4
```

Also, if the same command is executed a second time, the user will be prompted wether to overwrite the output Euro3D datacube. To avoid that, it is possible to specify the `-noask` option on input.

8.10 Example program #5: deleting a slice from a Euro3D file

Not yet available.

8.11 Example program #6: accessing a complete E3D datacube as a 2D image

Example program #6 (`E3DHandling6`) shows how to access the data of a Euro3D cube as a whole using a frame (stacked spectra). On output, the program will give the average of the datacube (good quality data only) with its error bars (if the statistical error spectra are present).

The program starts with the usual headers and includes the mandatory `E3D_io.h` include file. On entry, we specify the name of the input Euro3D datacube (`-name`).

```
/* ================================================================= */
/*!
\file       E3DHandling6.c
\copyright  (c) 2003 - Euro3D
\date       17/03/2003
\author     Pierre Ferruit
\version    1.0 24/03/2003 PF Creation
\keywords   E3D Cube  Handling, Cookbooks
\brief      E3D cube handling example file #6

Example file provided with the documentation of the
Euro3D libraries (v1.0). This file provides an
example of how to :
open an existing Euro3D datacube and access it as a frame
(stacked spectra)
*/
/* ================================================================= */

#include        <math.h>
#include        <IFU_io.h>


/* ================================================================= */
/*!
\page E3DHandling6 E3DHandling6

Brief description.

\param [\c -ic] name of the input E3D cube
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================= */

int  main(int argc, char **argv)
{
char      **argval;              /* input arguments of the program  */
char      **arglabel;            /* corresponding labels            */

E3D_file  cube;                  /* E3D file structure              */
char      name[lg_name+1];       /* name fo the E3D file            */

IMAGE2D   signalframe;           /* signal frame structure          */
```

```
IMAGE2D   noiseframe;            /* statistical error frame struct. */
int       NoiseFlag;             /* flag                            */

double    signalvalue;           /* work variable                   */
long      qsignalvalue;          /* work variable                   */
double    noisevalue;            /* work variables                  */
double    sum, sumnoise;         /* work variables                  */
long      nbsum;                 /* work variable                   */

long      i,j;                   /* work variables                  */
char      msg_txt[161];          /* message string                  */

/* -------------------------------------------------------------- */
/* Initialization of the Euro3D session                           */
/* -------------------------------------------------------------- */
set_version("1.0");
set_arglist("-ic none");
init_session(argv, argc, &arglabel, &argval);

/* -------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)   */
/* -------------------------------------------------------------- */
get_argval(0, "%s", name);
```

We open the Euro3D datacube (read-only) and check if the statistical error spectra are present.

```
/* -------------------------------------------------------------- */
/* Opening the Euro3 file using open_E3D_file() in read-only mode */
/* Checking ig the statistical error spectra are present          */
/* -------------------------------------------------------------- */
open_E3D_file(&cube, name, "Input");
NoiseFlag = exist_statistical_error(&cube);
```

We use the `get_E3D_file()` routine to stack all the spectra into a single 2D image (`signalframe`). If the statistical error spectra are present we also stack them into a second frame (`noiseframe`).

```
/* -------------------------------------------------------------- */
/* Putting the contents of the cube into frame(s)                 */
/* -------------------------------------------------------------- */
if (NoiseFlag) {
get_E3D_frame(&cube, &signalframe, &noiseframe);
}
else {
get_E3D_frame(&cube, &signalframe, NULL);
}
```

We can now compute the statistic over the images (see Sect. 5). Briefly, we ignore all data with a non-zero quality flag and if the statistical error data are present we also compute the error on the compute average value.

```
/* -------------------------------------------------------------- */
/* COmputing the statistics over the whole datacube               */
/* -------------------------------------------------------------- */
```

```
sum = (float) 0;
nbsum = (long) 0;

for (i = 0 ; i < signalframe.nx ; i++) {
for (j = 0 ; j < signalframe.ny ; j++) {

/* ...................................................... */
/* reading the signal and statistical error values and      */
/* qualities                                                */
/* ...................................................... */
signalvalue = (double) RD_frame(&signalframe, i, j);
qsignalvalue = (long) RD_qframe(&signalframe, i, j);

if (NoiseFlag) {
noisevalue = (double) RD_frame(&noiseframe, i, j);
}

/* ...................................................... */
/* Take them into account only if the the quality is 0 (OK)     */
/* ...................................................... */
if (qsignalvalue == (long) 0) {
sum += signalvalue;
nbsum++;
if (NoiseFlag) {
sumnoise += noisevalue * noisevalue;
}
}
}
}

sprintf(msg_txt, "Found %ld valid points.", nbsum);
print_msg(msg_txt);

if (NoiseFlag) {
sumnoise = sqrt(sumnoise);
sprintf(msg_txt, "Sum of the cube values: %12.6e +/- %12.6e (1 sigma)",
sum, sumnoise);
print_msg(msg_txt);
}
else {
sprintf(msg_txt, "Sum of the cube values: %12.6e", sum);
print_msg(msg_txt);
}
```

End of the program... We close the Euro3D datacube and exit the Euro3D session.

```
/* -------------------------------------------------------------- */
/* Closing the E3D file                                           */
/* -------------------------------------------------------------- */
close_E3D_file(&cube);

/* -------------------------------------------------------------- */
```

```
/* Session end                                                   */
/* ------------------------------------------------------------- */
exit_session(0);
}
```

We can use this example program `E3DHandling6` program to compute the sum of the values of the Euro3D datacube created using the `E3DHandling1` program (i.e. e3d1.fits):

`$(E3D_PATH)/E3D_CookBook-1.0/user/bin/E3DHandling6 -name e3d1`

The messages on the screen will be the following:

```
Program: E3DHandling6  Version 1.0-6.0
Found 79200 valid points.
Sum of the cube values: 2.687960e+01 +/- 5.184554e+00 (1 sigma)
End of E3DHandling6
```

9 Handling descriptors

9.1 Key variables and routines

Routines to manipulate descriptors

`RD_desc()`	read a descriptor from a file header.
`WR_desc()`	write a descriptor into a file header.
`delete_desc()`	delete a descriptor from the header of a file
`get_descr_type()`	return the data-type of a descriptor.
`get_all_desc()`	return the complete list of the descriptors present in a file header
`CP_non_std_desc()`	copy all non-standard descriptors (standard = standard FITS) from the header of one file to the header of another one.

9.2 Example program #1: Reading and writing descriptor from/into the header of a file)

Not yet available.

10 Manipulating the error control level of the library

In all the example programs detailed in the previous sections, an error occuring in one of the Euror3D library routines (e.g. trying to open a spectrum which does not exist) caused an error message to be displayed and the program to exit. This is because, by default, the error control level of the library is set to `FATAL`. In the following we describe how to change this error control level and how it will affect the behavior of a program.

10.1 Key definitions and routines

Error control routines

`set_control_level()`	change the error control level
`get_tiger_errmsg()`	get the error message corresponding to a given error code

Error control levels

`FATAL`	causes the library routines to display an error message and then exit the program in case of error (default behavior)
`WARNING`	causes the library routines to display a warning message and then return the error code in case of error (no exit)
`NONE`	the library routines return the error code without displaying any message

The various existing error codes are liste in Appendix A.

10.2 Example program

The `ErrorControl.c` example program starts with the classical headers, `E3D_io.h` include file and list of variable declarations.

```
/* ================================================================== */
/*!
\file        ErrorControl.c
\copyright   (c) 2003 - Euro3D
\date        15/03/2003
\author      Pierre Ferruit
\version     1.0 15/03/2003 PF Creation
\keywords    Error Control, Cookbooks
\brief       Error control level example file

Example file provided with the documentation of the
Euro3D libraries (v6.0). This file provide an
example of how the behavior of the library routines
used to handle spectra change with the error control
level. The program tries to open a spectrum with an
error control level set by the user.

*/
/* ================================================================== */

#include        <IFU_io.h>


/* ================================================================== */
```

```
/*!
\page ErrorControl ErrorControl

Brief description.

\param [\c -name] name of the input spectrum
\param [\c -level] error control level
0 = FATAL
1 = WARNING
2 = NONE
\return [\c int] Returns 0 if everything went smoothly,
-1 otherwise.
*/
/* ================================================================ */

int  main(int argc, char **argv)
{
char      **argval;              /* input arguments of the program  */
char      **arglabel;            /* corresponding labels            */

SPECTRUM  spectrum;              /* spectrum structure              */
char      name[81];              /* name of the spectrum            */
short     ErrorCtrlLevel;        /* error control level             */
long      status;                /* status returned by open_spec()  */

char      msg_txt[321];          /* message text                    */
```

The input arguments of the program are the name of the input spectrum (`-name`) and the desired error control level (`-level`).

```
/* ---------------------------------------------------------------- */
/* Initialization of the Euro3D session                             */
/* ---------------------------------------------------------------- */
set_version("1.0");
set_arglist("-name none -level none");
init_session(argv, argc, &arglabel, &argval);

/* ---------------------------------------------------------------- */
/* Analysis of the input arguments (no check of their validity)     */
/* ---------------------------------------------------------------- */
get_argval(0, "%s", name);
get_argval(1, "%ld", &ErrorCtrlLevel);
```

Now, we use the `set_control_level()` routine to set the desired control level. Then we try to open the spectrum. Note that contrary to what was done before, we do store the return value of the `open_spec()` routine. In previous programs, the error control level was left to its default, `FATAL` value. In this case, the library routines were causing the program to exit on error and it was not necessary to care about the return values of the routines.

```
/* ---------------------------------------------------------------- */
/* Adjusting the error control level                                */
/* ---------------------------------------------------------------- */
```

```
set_control_level(ErrorCtrlLevel);

/* ---------------------------------------------------------------- */
/* Opening the spectrum (in read-only mode)                         */
/* ---------------------------------------------------------------- */
status = open_spec(&spectrum, name, "Input");
```

Now we just display the value returned by the `open_spec()` routine and the corrsponding error message (obtained by using the `get_tiger_errmsg()` routine).

```
/* ---------------------------------------------------------------- */
/* Error control stuff                                              */
/* ---------------------------------------------------------------- */
sprintf(msg_txt, "Code returned by the open_spec() routine : %d",
ErrorCtrlLevel);
print_msg(msg_txt);

print_msg("Error message corresponding to this code:");

get_tiger_errmsg(ErrorCtrlLevel, msg_txt);

print_msg(msg_txt);
```

Last, we close the spectrum and exit the Euro3D session.

```
/* ---------------------------------------------------------------- */
/* Closing the spectrum                                             */
/* ---------------------------------------------------------------- */
close_spec(&spectrum);

/* ---------------------------------------------------------------- */
/* Session end                                                      */
/* ---------------------------------------------------------------- */
exit_session(0);
}
```

10.3 Executing the program

Now, we are going to examine the behavior of `ErrorControl` program when the spectrum to open does not exist and for the three different error control levels.

`$(E3D_PATH)/E3D_CookBook-1.0/user/bin/ErrorControl -name toto -level 0`

In this case, the error control level is set to `FATAL` (default case) and the output of the program is the following:

```
Program: ErrorControl  Version 1.0-6.0       Obs. de Lyon
ERROR: FATAL error from routine open_spec: spec ./toto.fits
ERROR: Error opening file

ErrorControl aborted - status code : -10
```

As expected, the routine `open_spec()` displays an error message and causes the program to exit.

`$(E3D_PATH)/E3D_CookBook-1.0/user/bin/ErrorControl -name toto -level 1`

In this case, the error control level is set to `WARNING` and the output of the program is the following:

```
Program: ErrorControl  Version 1.0-6.0       Obs. de Lyon
WARNING: WARNING from routine open_spec: spec ./.fits
WARNING: Error opening file

Code returned by the open_spec() routine : -10
Error message corresponding to this code:
Error opening file

WARNING: WARNING from routine set_subspec: spec .fits
WARNING: Bad section specification for image

ERROR: Memory fault detected, signal 11
```

The first four lines correspond to the warning message delivered by the `open_spec()` routine, while the five next line are the error code and the corresponding error message we asked the program to display. At this stage, the program has not exited depsite the presence of errors.

However when we want to close the spectrum (which has not been opened...) with the routine `close_spec()`, this triggers two WARNING messages and causes the program to exit with a memory fault messaget when the routine tries to free the memory of the spectrum (which has never been allocated...) This outlines the dangers of manipulating the error control level in programs...

`$(E3D_PATH)/E3D_CookBook-1.0/user/bin/ErrorControl -name toto -level 2`

In this last case, the error control level is set to `NONE` and the output of the program is the following:

```
Program: ErrorControl  Version 1.0-6.0       Obs. de Lyon
Code returned by the open_spec() routine : -10
Error message corresponding to this code:
Error opening file

ERROR: Memory fault detected, signal 11
```

The behavior of the program is identical to the one in the case of an error control level of `WARNING`, except that all the warning messages have been skipped.

—oOo—

A Error codes

In the following, we list the error control level codes (used in the `set_control_level()` routine) and the various error codes retunred by the Euror3D library routines.

```
/* ============================================================= */
/* Error control level codes                                     */
/* ============================================================= */
#define FATAL   0
#define WARNING 1
#define NONE    2

/* ============================================================= */
/* Error codes returned by the I/O library routines              */
/* ============================================================= */
#define OK               0  /* Successful return                        */
#define UNKNOWN         -1  /* Unknown                                  */
#define ERR_EOF         -2  /* EOF indicator                            */
#define ERR_BAD_PARAM   -3  /* Input element is bad                     */
#define ERR_NB_PARAM    -4  /* Bad input number of elements             */
#define ERR_BAD_TYPE    -5  /* Invalid data type                        */
#define ERR_BAD_SIZE    -6  /* Field width not wide enough              */
#define ERR_OPEN        -10 /* Error opening existing file              */
#define ERR_CREAT       -11 /* Error opening new file                   */
#define ERR_READ        -12 /* Error reading file                       */
#define ERR_WRIT        -13 /* Error writing to file                    */
#define ERR_CLOSE       -14 /* Error closing existing file              */
#define ERR_ACCESS      -15 /* Invalid file access mode                 */
#define ERR_NAXIS       -20 /* Invalid NAXIS parameter                  */
#define ERR_NOIDENT     -21 /* Identifier not found                     */
#define ERR_OFFSET      -31 /* Error returning offset to data           */
#define ERR_NODATA      -32 /* No data available                        */
#define ERR_BAD_IMA     -41 /* Not an image !                           */
#define ERR_IMA_BOUND   -42 /* Bad section specification for image     */
#define ERR_IMA_EXT     -43 /* Bad extension for image                  */
#define ERR_NOIMA       -44 /* Image does not exist                     */
#define ERR_DEL_IMA     -45 /* Error deleting image                     */
#define ERR_REN_IMA     -46 /* Error renaming image                     */
#define ERR_IMA_HEAD    -47 /* Illegal image header                     */
#define ERR_BAD_TBL     -51 /* Not a table !                            */
#define ERR_TBL_EXT     -53 /* Bad extension for table                  */
#define ERR_NOTBL       -54 /* Table does not exist                     */
#define ERR_DEL_TBL     -55 /* Error deleting table                     */
#define ERR_REN_TBL     -56 /* Error renaming table                     */
#define ERR_TBL_HEAD    -57 /* Illegal table header                     */
#define ERR_BAD_COL     -59 /* Error in column format                   */
#define ERR_NOCOL       -60 /* Column does not exist                    */
#define ERR_COL_NUM     -62 /* Wrong column number                      */
#define ERR_ROW_NUM     -63 /* Wrong row number                         */
#define ERR_NODESC      -80 /* Header parameter not found               */
#define ERR_BAD_HEAD    -81 /* Illegal data type for header parameter */
```

```
#define ERR_DEL_DESC    -82 /* Cannot delete descriptor          */
#define ERR_BAD_DESC    -83 /* Descriptor bad                    */
#define ERR_HEADER_SIZE -85 /* Out of space in header            */
#define ERR_BAD_CAT     -90 /* Not a catalog !                   */
#define ERR_GRAPH_DEV  -100 /* Bad graphics device               */
#define ERR_ALLOC      -110 /* Error allocating dynamic memory   */
#define ERR_FREE       -111 /* Error freeing dynamic memory      */
#define ERR_OVERFLOW   -112 /* Overflow (column/frame)           */
#define ERR_FORMAT     -113 /* Unknown data format               */
#define ERR_NOIMPL     -130 /* Not yet implemented               */
```