User Manual for libRASCH-0.8.29

Raphael Schneider

User Manual for libRASCH-0.8.29

by Raphael Schneider

libRASCH User Manual Version 0.1 Edition
Published 2006

Table of Contents

List of Figures

Forword

Nothing yet.

Chapter 1. Introduction

When analyzing biological signals, access to the raw data is mandatory. Because of different demands, both, the industry and the research community created a high number of different data formats for signal storage and signal distribution. To analyze data stored in a new data format, either a conversion program has to be written or the access functionality for the new data format has to be added to the analyzing program(s).

One way to solve this problem is the use of a standard file format, which is powerful enough to handle all needs for storing and distributing signals. For physiological signals, the 'File Exchange Format for Vital Signs' (FEF) tries to accomplish this task. If this format (or a similar one) is accepted by the research community and if the industry provides the possibility to export the signal data in this format, the data access will be facilitated.

Our approach, moreover, is different. We assume that there will be always different data formats (some 'standard' formats and a lot of proprietary formats). Therefore libRASCH, a programming library, was developed, which hides the differences of the data formats behind a common application programming interface (API).

Programs using libRASCH, no longer need to be adapted to each new data format. The implementation of a new data format needs only to be done once for libRASCH. Than all libRASCH based programs can handle the new format.

Additionally, libRASCH provides the infrastructure to perform processing algorithms in a standardized way and provide support to display the signal data on the computer screen.

1.1. Overview of the next chapters

concept/terminology: describes how a measurement is seen in libRASCH and describes the words used in libRASCH

installation: more or less the INSTALL file (use one source for both files)

tutorial: short examples describing the primary functions of libRASCH

Chapter 2. Concepts and Terminology

This section gives a short overview about the concepts and terminology used in the manual.

2.1. General Structure of libRASCH

Figure 2-1 shows the operational area of libRASCH. Programs from User-space will call libRASCH to access measurements stored in files, to display signals on the screen and to process the signal. In the following "User-space" means all programs, which use the external interface of libRASCH (the API of libRASCH). Also programs like Matlab or Octave, for which an interface is available, are user-space programs (and the scripts used in this programs). "Library-space" describes the internal interfaces used in libRASCH. And "Filesystem" describes the real files used to store the measurement data on disk.

Figure 2-1. Structure of libRASCH.

2.2. Terminology

Measurement

A measurement is the topmost object in libRASCH. Measurements consists of one or more sessions, information about the measurement object (e.g. name, forename and birthday if the measurement object is a person) and zero or more evaluations.

Session

A Session is a recording for a specific time interval without any interruptions during this time interval. In a measurement can be more than one session, but the layout of the recording (see below) must not be changed.

Recording

A recording contains the measured data (e.g. ecg-leads V1-V6). A recording has one or more channels or two or more sub-recordings. Sub-recordings are used if more than one recording device is used. For example when one ADC-system records 3 ecg-leads and one bloodpressure channel and another system records 12 eeg-leeds, the measurement consists of one top recording with two sub-recordings. The first sub-recording contains 4 channels (3 ecg and 1 bloodpressure channel) and the second sub-recording contains 12 channels (12 eeg channels).

Evaluation

The results of an analysis (e.g. detection of qrs-complexes in ecg's) are stored in an evaluation. An evaluation contains zero or more discrete events (like 'occurence of a qrs-complex') and/or zero or more continous events (like 'time interval with noise').

Event

An event describes the occurence of something in a recording (e.g. a heartbeat in an ecg). An event has one or more event-properties.

Event-Property

An event-property is a specific property of the event (e.g. the position of the event, the type of the event). A specific event-property is allowed only once in an evaluation, for example there can be not more than one ’qrs-position’ property.

Event-Set

An event-set describes a group of event-properties. For example the event-set ’heartbeat’ contains all properties which belongs to a heart beat (like position of qrs-complex, RR interval, type of qrs-complex, systolic bloodpressure).

Plugins

libRASCH makes heavy use of plugins. Plugins are small "programs" which are loaded when libRASCH is initialized. In the plugins the real work is done, the library-code coordinates that the correct plugin is used and does some other administrative tasks. In libRASCH exits three principal types of plugins:

- signal plugins
- view plugins
- process plugins

Access-Plugin

Access plugins handle the access to measurement files. They hide the differences of the various types of formats and offer an consistent interface to the measurements[1]. Most of the times, the direct usage of access-plugins with the libRASCH-API is not needed.

Process-Plugin

Process plugins perform a specifc task on the measurement (e.g. the HRT-plugin calculates the Heart-Rate Turbulence parameters for an ecg or the detect plugin performs a simple beat detection in ecg’s). The usage of these plugins

View-Plugin

View plugins allow to display the measurements on the screen. If you develop a program using a graphical user interface (GUI) all you need to display signals is to call the apropriate plugin. At the moment the following GUI’s are supported:

- Qt from Trolltech (for Linux)
- MFC from Microsoft (for Windows)

Notes

1. The signal plugins can be compared with device drivers in operating systems.

Chapter 3. A Tutorial

The handling of evaluations will be changed in the next version of libRASCH. Because we are working already on it, the sections about evaluations, event-sets and event-propertiey may not work with the current version (0.7.0). If you have any questions about the changes, please send them to the mailing list.

3.1. Initialize libRASCH and Basic Usage

This section will show you how to start using libRASCH. We will initialize the library and get some information about it.

So let's start with the first example. We will perform the following steps:

- init libRASCH
- get number of plugins
- find all measurements in a directory

3.1.1. C Version

```
//
#include <stdio.h>
#include <ra.h> ❶

int main(int argc, char *argv[])
{
    ra_handle ra;
    value_handle vh;
    ra_find_handle f;
    struct ra_find_struct fs;

    /* initialize libRASCH */
    ra = ra_lib_init(); ❷

    /* check if init was successful */
    if ((ra == NULL)
        || (ra_lib_get_error(ra, NULL, 0) != RA_ERR_NONE)) ❸
    {
        if (!ra)
            printf("error initializing libRASCH\n");
        else
        {
            char err_t[200];
            long err_num;
```

```
            err_num = ra_lib_get_error(ra, err_t, 200);
            printf("while initializing libRASCH, error #%d "
                   "occured\n  %s\n", err_num, err_t);

            ra_lib_close(ra);
        }
        return -1;
    }

    /* get some infos */
    vh = ra_value_malloc(); ❹
    if (ra_info_get(ra, RA_INFO_NUM_PLUGINS_L, vh) == 0) ❺
    {
        printf("%s (%s): %d\n", ra_value_get_name(vh),
               ra_value_get_desc(vh), ra_value_get_long(vh));
    }
    ra_value_free(vh); ❻

    /* find all measurements in a directory */
    f = ra_meas_find_first(ra, argv[1], &fs); ❼
    if (f)
    {
        int cnt = 1;

        printf("measurements found in %s:\n", argv[1]);
        do
        {
            printf("  %2d: %s\n", cnt, fs.name);
            cnt++;
        }
        while (ra_meas_find_next(f, &fs)); ❽

        ra_meas_close_find(f); ❾
    }

    /* close libRASCH */
    ra_lib_close(ra); (10)

    return 0;
}                               /* main() */

//
```

❶ To use libRASCH, the header-file ra.h must be included. In this example the include-directory of libRASCH is in the INCLUDE path of the C compiler. In ra.h you will find all function prototypes of the API of libRASCH. ra.h includes the header-file ra_defines.h, there you will find all define's and structure's needed for the libRASCH API.

❷ ra_lib_init() initialize libRASCH. The function returns an ra_handle.

❸ ra_lib_get_error() returns the last error occured in libRASCH. To check if the initialization was successful, check that no error occured. If an error occured, a short description of the error can be retrieved with the function ra_lib_get_error().

❹ ra_value_malloc() returns an value object. This object will be used in libRASCH to handle data. To set/get the data and to get informations about the stored data, API functions in libRASCH are available. The functions start with 'ra_value_*'.

❺ ra_info_get() returns information about libRASCH and all objects handled by libRASCH. RA_INFO_NUM_PLUGINS_L asks libRASCH for the the number of loaded plugins. The last character of the info-id ('L') indicates that the returned value will be a long-value. Therefore the number of plugins will be returned by using the function ra_value_get_long(). (See the descriptions of the ra_value_* functions in the reference manual what else can be done with an value object.)

❻ ra_value_free() frees the memory associated with the value object allocated above.

❼ ra_meas_find_first() returns a valid handle (not NULL) if at least one supported measurement (this means that at least one measurement in the directory can be handled with one of the loaded access-plugins). The information about the found measurement will be set in the ra_find_struct (for definition of structure see ra_defines.h).

❽ ra_meas_find_next() returns true (!= 0) if another measurement is available. Again, the info about the measurement will be in the ra_find_struct. This function iterates over all found measurements.

❾ ra_meas_close_find() frees all memory allocated during ra_meas_find_first() and ra_meas_find_next().

(10) ra_lib_close() unloads all plugins and frees all allocated memory.

Running the example in the examples directory with the command **init_lib ./database** produced the following output.

```
#plugins (): 37
measurements found in ./database:
   1: ./database/100s.hea
   2: ./database/JesusOlivan2003-12-EMG2.edf
   3: ./database/100s.dat
```

3.1.2. Perl Version

The Perl script shown below produces exactly the same output as the C version, therefore the output is not shown.

```
#
use strict;
use RASCH; ❶
```

```
# initialize libRASCH
my $ra = new RASCH or die "error initializing libRASCH\n"; ❷
my ($err_num, $err_text) = $ra->get_error ();
if ($err_num != 1)
{
    print "while initializing libRASCH, error # $err_num " .
        "occured:\n $err_text\n";
    exit -1;
}

# get some infos
my $value = $ra->get_info(info =>'num_plugins'); ❸
if ($value->is_ok())
{
    print $value->name() . " (" . $value->desc() . "): " . $value->value() . "\n"; ❹
}

# find all measurements in a directory
my $meas = $ra->find_meas($ARGV[0]); ❺
print "measurements found in $ARGV[0]:\n";
my $cnt = 1;
for (@$meas)
{
    print "$cnt: " . $_->filename() . "\n";
    $cnt++;
}

# ra_close() will be called when $ra is being destroyed

exit 0;
#
```

❶ After installing the Perl support for libRASCH, the package RASCH.pm is available.

❷ Create a new RASCH object.

❸ The function get_info() returns an RAvalue object.

❹ The function value() returns the value stored in a RAvalue object. The functions name() and desc() returns the name and a short description of the info, respectively.

❺ get_all_meas() returns an array with all the measurements found.

3.1.3. Python Version

The Python script shown below produces exactly the same output as the C and Perl version, therefore the output is not shown.

```
#
import sys
from RASCH import * ❶

# initialize libRASCH
ra = RASCH() ❷
if not ra:
    print "can't initialize libRASCH"
    sys.exit()
[err_num, err_text] = ra.get_error()
if err_num != 1:
    print "while initializing libRASCH, error #%d occured:\n " \
          "%s\n" % err_num, err_text
    sys.exit()

# get some infos
value = ra.get_info(info='num_plugins')  ❸
if (value.is_ok()):
    print value.name(), "("+value.desc()+"):", value.value() ❹

# find all measurements in a directory
meas = ra.find_meas(sys.argv[1]) ❺
print "measurements found in " + sys.argv[1] + ":\n"
cnt = 1
for item in meas:
    print "%d: %s" % (cnt, item.filename())
    cnt = cnt + 1

#
```

❶ After installing the Python support for libRASCH, the module 'RASCH' is available.

❷ Create a new RASCH object.

❸ The function get_info() returns a RAvalue object.

❹ The function value() returns the value stored in a RAvalue object. The functions name() and desc() returns the name and a short description of the info, respectively.

❺ get_all_meas() returns an array with all the measurements found.

3.1.4. Matlab/Octave Version

Here an Octave session using the libRASCH support is shown. The same tasks are performed as in the previous examples. Most of the functions have in Matlab and Octave the same behaviour. Differences are listed in the function reference section for Matlab and Octave.

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
```

```
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WA❶RRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".

Additional information about Octave is available at              ❷
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit                    ❸
http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).

For information about changes from previous versions, type
'news'.

+ ra=ra_lib_init
ra =  162635120
+ [err_num, err_text]=ra_lib_get_error(ra)
err_num =  1
err_text = libRASCH (ra.c:109): no error
+ [value, name, desc]=ra_lib_info_get(ra, 'num_plugins')
value =  37
name = #plugins
desc =
+ meas=ra_meas_find(ra, './database')
meas =

{
  [1,1] = ./database/100s.hea
  [2,1] = ./database/JesusOlivan2003-12-EMG2.edf
  [3,1] = ./database/100s.dat
}

+ ra_lib_close(ra);
```

❶ Initialize libRASCH. If function call is successfull, a value not 0 will be returned.

❷ The function returns an array including (1) the value, (2) the name and (3) a short description of the wanted information.

❸ The function returns a cell-array with all the measurements found.

3.2. Access Measurements

The example will show how to open a measurement and how to get some information from a measurement. We will perform the following steps:

- open a measurement
- get the number of sessions and the maximum samplerate
- print all available infos about the measurement object (in this example it is a patient)

```
//
#include <stdio.h>
#include <ra.h>

int main(int argc, char *argv[])
{
    ra_handle ra;
    value_handle vh;
    meas_handle meas;

    /* initialize libRASCH */
    ra = ra_lib_init();
    if ((ra == NULL)
        || (ra_lib_get_error(ra, NULL, 0) != RA_ERR_NONE))
    {
        if (!ra)
            printf("error initializing libRASCH\n");
        else
        {
            char err_t[200];
            long err_num;

            err_num = ra_lib_get_error(ra, err_t, 200);
            printf("while initializing libRASCH, error #%d "
                   "occured\n  %s\n", err_num, err_t);

            ra_lib_close(ra);
        }
        return -1;
    }

    /* open measurement */
    meas = ra_meas_open(ra, argv[1], NULL, 0); ❶
    if (meas == NULL)
    {
        char err_t[200];
        long err_num;

        err_num = ra_lib_get_error(ra, err_t, 200);
        printf("can't open measurement %s\nerror #%d: %s\n",
```

```
            argv[1], err_num, err_t);
        ra_lib_close(ra);
        return -1;
    }

    /* get some infos */
    vh = ra_value_malloc();
    if (ra_info_get(meas, RA_INFO_NUM_SESSIONS_L, vh) == 0)
        printf("%s (%s): %d\n", ra_value_get_name(vh),
            ra_value_get_desc(vh), ra_value_get_long(vh));
    if (ra_info_get(meas, RA_INFO_MAX_SAMPLERATE_D, vh) == 0)
        printf("%s (%s): %f\n", ra_value_get_name(vh),
            ra_value_get_desc(vh), ra_value_get_double(vh));

    /* get all measurement-object infos */
    if (ra_info_get(meas, RA_INFO_NUM_OBJ_INFOS_L, vh) == 0) ❷
    {
        long l;
        long n = ra_value_get_long(vh);

        for (l = 0; l < n; l++)
        {
            ra_info_get_by_idx(meas, RA_INFO_OBJECT, l, vh); ❸
            printf("%s (%s): ", ra_value_get_name(vh),
                   ra_value_get_desc(vh));
            switch (ra_value_get_type(vh)) ❹
            {
            case RA_VALUE_TYPE_LONG:
                printf("%d\n", ra_value_get_long(vh));
                break;
            case RA_VALUE_TYPE_DOUBLE:
                printf("%f\n", ra_value_get_double(vh));
                break;
            case RA_VALUE_TYPE_CHAR:
                printf("%s\n", ra_value_get_string(vh));
                break;
            default:
                printf("not supported type\n");
                break;
            }
        }
    }

    /* close */
    ra_value_free(vh);
    ra_meas_close(meas);
    ra_lib_close(ra);

    return 0;
}                               /* main() */

//
```

❶ ra_open_meas() opens a measurement. The user do not need to specify the format in which the measurement was saved, libRASCH selects the access plugin which can handle the format. If no access plugin can handle the measurement, the function fails. The third parameter (here it is set to '0') controls if a fast-open should be done (set to '1'). When a fast-open is selected, some "time-consuming" initialization is skipped and only the object informations and some basic recording informations (e.g. recording date) are available. [1]

❷ With this call, the number of available measurement objects is returned. In the following loop, we will get all available informations about the measurement object.

❸ With the function ra_info_get_by_idx() we select the information we want by a number. Because we do not know which number is which information, this function is only useful to display all available data in a list.

❹ Because we do not know which information is returned, we do not know the type of the information. Therefore we have to get the type of the information so we use the correct way to display the information.

The output of the above example for the measurement '100s' is shown here:

```
#sessions (): 1
max. samplerate (maximum samplrate used in measurement):
360.000000
ID (Patient-ID): 100s
```

3.3. Access Recordings

The example will show how to get the root recording and some information about it. We will perform the following steps:

- get root recording
- get some general informations about the recording
- get some infos about all used recording devices
- get some infos about all recorded channels

When more than one recording device was used, the root recording provides access to the channels as if they were recorded with one device.

```
#
use strict;
use RASCH;
```

```
# initialize libRASCH
my $ra = new RASCH or die "error initializing libRASCH\n";

# open measurement
my $meas = $ra->open_meas($ARGV[0], '', 0) or
    die "can't open measurement $ARGV[0]\n";

# get root recording
my $rec = $meas->get_first_rec(0) or ❶
    die "can't get root recording\n";

# get some infos about recording
my $v = $rec->get_info(info => 'rec_num_devices');
my $num_dev = $v->value();
$v = $rec->get_info(info => 'rec_num_channel');
my $num_ch = $v->value();
$v = $rec->get_info(info => 'rec_name');
my $rec_name = $v->value();
$v = $rec->get_info(info => 'rec_date');
my $rec_date = $v->value;
print "measurement $rec_name\nrecorded at $rec_date\n" .
    "#devices=$num_dev\n#channels=$num_ch\n\n";

# print name for every device
print "infos about the recording devices used:\n";
for (my $i = 0; $i < $num_dev; $i++)
{
    $v = $rec->get_info(dev => $i, info => 'dev_hw_name'); ❷
    my $name = $v->value();
    print "  device #$i: $name\n";
}
print "\n";

# print name for every channel
print "infos about the channels:\n";
for (my $i = 0; $i < $num_ch; $i++)
{
    $v = $rec->get_info(ch => $i, info => 'ch_name'); ❸
    my $name = $v->value();
    $v = $rec->get_info(ch => $i, info => 'ch_unit');
    my $unit = $v->value();
    print "  channel #$i: $name [$unit]\n";
}
print "\n";

exit 0;
#
```

❶ The function get_first_rec(s_num) returns the root-recording of the session s_num. The sessions start with number 0.

❷ When asking for information about recording devices you need to set the device number. In Perl and Python this is done by setting the argument 'dev' to the device number. In C the device number is set

in the value_handle (using ra_value_set_number()) used to receive the information when calling ra_info_get().

❸ When asking for information about a channel you need to set the channel number. In Perl and Python this is done by setting the argument 'ch' to the device number. In C the device number is set in the value_handle (using ra_value_set_number()) used to receive the information when calling ra_info_get()

The output of the above example for the measurement '100s' is shown here:

```
measurement 100s
recorded at 00.00.0
#devices=
#channels=2

infos about the recording devices used:

infos about the channels:
  channel #0: MLII []
  channel #1: V5 []
```

3.4. Get Sample Data

The example will show how to get the root recording and some information about it. We will perform the following steps:

- get root recording
- get some general informations about the recording
- get some infos about all used recording devices
- get some infos about all recorded channels

When more than one recording device was used, the root recording provides access to the channels as if they were recorded with one device.

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".
```

```
Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit
http://www.octave.org/help-wanted.h❶tml

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).

For information about changes from previous versions, type
'news'.

+ ra=ra_lib_init
ra =  185012408
+ meas=ra_meas_open(ra, './database/100s.hea', '', 0)
meas =  185820352
+ rec=ra_rec_get_first(meas, 0)
rec =  184808528
+ num_ch=ra_rec_info_get(rec, 'rec_num_channel')
num_ch =  2
+ ch_all=[];
+ for i=0:(num_ch-1)
+   ch=ra_raw_get_unit(rec, i, 0, 10000);
+   ch_all=[ch_all ch'];
+ endfor
+ whos ch_all

*** local user variables:

  Prot Name        Size                     Bytes  Class
  ==== ====        ====                     =====  =====
   rwd ch_all  10000x2                    160000  double

Total is 20000 elements using 160000 bytes

+ samplerate=ra_ch_info_get(rec, 0, 'ch_samplerate')
samplerate =  360
+ x=0:9999;
+ x = x / samplerate;
+
+ figure();
+ for i=1:num_ch
+   subplot(num_ch,1,i)
+   plot(x,ch_all(:,i))
+ endfor
+
+ ra_meas_close(meas);
+ ra_lib_close(ra);
```

❶ Returns 10,000 samples for channels 0..4, starting with sample 0 in each channel.

Figure 3-1 shows a screenshot of Octave after performing the above steps. Each plot window on the right side shows a recording channel. The x-axis is in seconds.

Figure 3-1. Screenshot after performing the commands in Octave as shown in the Octave session above (and on the left side in the screenshot).

3.5. Access Evaluations

The example will show how to get the evaluation belonging to a measurement and to get some information about it. We will perform the following steps:

- get default evaluation
- get some general infos about the evaluation
- get a list of event-properties

libRASCH supports different "types" of evaluations, these are 'original evaluation', 'default evaluation' and '"old" evaluations. The 'original evaluation' is the evaluation performed with the recording system (e.g. commercial Holter systems). In a measurement there can be only one 'original evaluation', but there can be none.

The 'default evaluation' is the evaluation which should be used as default. If a measurement has only a 'original evaluation', than this will also be the 'default evaluation'.

Old evaluations are like a history of evaluations. When a new evaluation is added to a measurement, than this will be the new 'default evaluation' and the previous 'default evaluation' will become a 'old evaluation'. This enables the user to go back to a previous evaluation, if the current evalution went wrong.

```
//
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ra.h>

int main(int argc, char *argv[])
{
    ra_handle ra;
    value_handle vh;
    meas_handle meas;
```

```
eval_handle eval;
long l, m, num_class, num_prop;
class_handle *clh = NULL;
prop_handle *ph = NULL;

/* initialize libRASCH */
ra = ra_lib_init();
if (ra == NULL)
{
    printf("error initializing libRASCH\n");
    return -1;
}

/* open measurement */
meas = ra_meas_open(ra, argv[1], NULL, 0);
if (meas == NULL)
{
    printf("can't open measurement %s\n", argv[1]);
    return -1;                ❶
}

/* get default evaluation */
eval = ra_eval_get_default(meas);
if (eval == NULL)
{
    printf("no evaluation in measurement %s\n", argv[1]);
    return -1;
}

/* get some infos about evaluation */
vh = ra_value_malloc();
if (ra_info_get(eval, RA_INFO_EVAL_NAME_C, vh) == 0)
    printf("evaluation %s ", ra_value_get_string(vh));
if (ra_info_get(eval, RA_INFO_EVAL_ADD_TS_C, vh) == 0)
    printf("was added at %s ", ra_value_get_string(vh));
if (ra_info_get(eval, RA_INFO_EVAL_PROG_C, vh) == 0)
    printf("using the program %s", ra_value_get_string(vh));
printf("\n\n");

/* list event-class's */  ❷
num_class = 0;
if (ra_class_get(eval, NULL, vh) == 0)
{
    num_class = ra_value_get_num_elem(vh);
    clh = malloc(sizeof(class_handle) * num_class);
    memcpy(clh, (void *)ra_value_get_voidp_array(vh),
           sizeof(class_handle) * num_class);
}

for (l = 0; l < num_class; l++)
{
    if (ra_info_get(clh[l], RA_INFO_CLASS_NAME_C, vh) == 0)
        printf("event-class %s", ra_value_get_string(vh));
```

```
        if (ra_info_get(clh[l], RA_INFO_CLASS_EV_NUM_L, vh) == 0)
            printf(" with %d events", ra_value_get_long(vh));
        printf(":\n");

        /* list event-properties */
        num_prop = 0;
        if (ra_prop_get_all(clh[l], vh) == 0)
        {
            num_prop = ra_value_get_num_elem(vh);
            ph = malloc(sizeof(prop_handle) * num_prop);
            memcpy(ph, (void *)ra_value_get_voidp_array(vh),
                   sizeof(prop_handle) * num_prop);
        }

        for (m = 0; m < num_prop; m++)
        {
            if (ra_info_get(ph[m], RA_INFO_PROP_ASCII_ID_C, vh)
                == 0)
                printf("  %s\n", ra_value_get_string(vh));
            if (ra_info_get(ph[m], RA_INFO_PROP_DESC_C, vh) == 0)
                printf("    %s", ra_value_get_string(vh));
            printf("\n");
        }

        free(ph);
    }
    free(clh);

    /* close */
    ra_value_free(vh);
    ra_meas_close(meas);
    ra_lib_close(ra);

    return 0;
}                               /* main() */

//
```

❶ Returns the 'default evaluation'.

❷ In the function ra_info_get() we got the number of available event properties. Now we return some informations about each event property.

The output of the above example for ECG '100s' is shown here:

```
evaluation RASCHlab was added at 15.11.2009  11:48:00 using
the program raschlab_qt

event-class heartbeat with 75 events:
  qrs-pos
    position of fiducial point of QRS-complex in sampleunits
```

```
qrs-annot
  annotation of QRS complex
ecg-noise

ecg-morph-flags
  flags for the morphology values
ecg-p-type

ecg-qrs-type

ecg-t-type

ecg-p-start
  offset of p-wave begin from qrs-pos in sampleunits
ecg-p-end
  offset of p-wave end from qrs-pos in sampleunits
ecg-p-width

ecg-p-peak-1
  offset of p-wave peak from qrs-pos in sampleunits
ecg-p-peak-2
  offset of 2nd p-wave peak (if biphasic) from qrs-pos
  in sampleunits
ecg-qrs-start
  offset of start of qrs complex from qrs-pos in sampleunits
ecg-qrs-end
  offset of end of qrs complex from qrs-pos in sampleunits
ecg-qrs-width

ecg-q-peak
  offset of q-wave from qrs-pos in sampleunits
ecg-r-peak
  offset of r-wave from qrs-pos in sampleunits
ecg-s-peak
  offset of s-wave from qrs-pos in sampleunits
ecg-rs-peak
  offset of r'-wave from qrs-pos in sampleunits
ecg-t-start
  offset of t-wave begin from qrs-pos in sampleunits
ecg-t-end
  offset of t-wave end from qrs-pos in sampleunits
ecg-t-width

ecg-t-peak-1
  offset of t-wave peak from qrs-pos in sampleunits
ecg-t-peak-2
  offset of 2nd t-wave peak (if biphasic) from qrs-pos
  in sampleunits
ecg-pq
  PQ interval
ecg-qt
  QT interval
ecg-qtc
```

```
    QTc interval
  qrs-temporal
    temporal setting of beat
  ecg-flags
    ECG releated flags
  rri
    RR interval
  rri-annot
    annoation of RR interval
  rri-refvalue
    reference rri representing the current heart-rate
  rri-num-refvalue
    number of rri's used for calculation of reference value
```

3.6. Get Events

The example will show how to get values associated with an event. We will perform the following steps:

- get RR intervals and the postion of the QRS-complexes
- plot the tachogram of the measurement

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".

Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software usefu❶l.
For more information, visit
http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).

For information about changes from previous versions, type
'news'.

+ ra=ra_lib_init
ra =  184868040
+ meas=ra_meas_open(ra, './database/100s.hea', '', 0)
```

```
meas =  185403952
+ eva=ra_eval_get_default(meas)
eva =  185920976
+
+ samplerate=ra_meas_info_get(meas, 'max_samplerate')
samplerate =  360
+
+ cl = ra_class_get(eva, 'heartbeat')
cl =  185834776
+ num_hb = length(cl)
num_hb =  1
+ for i = 1:num_hb
+   ev = ra_class_get_events(cl(i), -1, -1, 0, 1);
+   num_ev = length(ev)
+
+   prop_rri = ra_prop_get(cl(i), 'rri')
+   prop_qrs_pos = ra_prop_get(cl(i), 'qrs-pos')
+
+   rri = [];
+   qrs_pos = [];
+   for j = 1:num_ev
+     r = ra_prop_get_value(prop_rri, ev(j), -1);
+     p = ra_prop_get_value(prop_qrs_pos, ev(j), -1);
+
+     rri = [rri r];
+     qrs_pos = [qrs_pos p];
+   endfor
+
+   %% To get all property values within one function call
+   %% use ra_prop_get_value_all():
+   % [ev_ids, chs, rri] = ra_prop_get_value_all(prop_rri);
+   % [ev_ids, chs, qrs_pos] =
ra_prop_get_value_all(prop_qrs_pos);
+   %
+   %% !!! This function returns also the event-id's and channel
+   %% !!! numbers the value belongs to. The order of returned
values
+   %% !!! is not guaranteed to be in chronological order.
+
+   whos rri
+   whos qrs_pos
+
+   figure();
+   x=(qrs_pos/samplerate) / 60;
+   plot(x, rri);
+ endfor
num_ev =  75
prop_rri =  186329584
prop_qrs_pos =  186289912

*** local user variables:

  Prot Name        Size                     Bytes  Class
```

```
  ==== ====        ====                    =====  =====
   rwd rri         1x75                      600  double

Total is 75 elements using 600 bytes


*** local user variables:

  Prot Name         Size                    Bytes  Class
  ==== ====         ====                    =====  =====
   rwd qrs_pos      1x75                      600  double

Total is 75 elements using 600 bytes

+
+ ra_meas_close(meas);
+ ra_lib_close(ra);
```

❶

Figure 3-2 shows a screenshot of Octave after performing the above steps.

Figure 3-2. Screenshot after performing the commands in Octave as shown in the Octave session above (and on the left side in the screenshot).

3.7. Usage of process-plugins

The example will show how to calculate Heart-Rate-Variability (HRV) using the process-plugin 'hrv'. We will perform the following steps:

- get 'hrv' plugin
- calculate HRV using the plugin
- print the calculation results

```
//
#include <stdio.h>
#include <string.h>
#include <ra.h>

int main(int argc, char *argv[])
{
    ra_handle ra;
    struct ra_info *inf;
    meas_handle meas;
```

```
eval_handle eval;
plugin_handle pl;
struct proc_info *pi;

/* initialize libRASCH */
ra = ra_lib_init();
if (ra == NULL)
{
    printf("error initializing libRASCH\n");
    return -1;
}

/* open measurement */
meas = ra_meas_open(ra, argv[1], NULL, 0);
if (meas == NULL)
{
    printf("can't open measurement %s\n", argv[1]);
    return -1;
}

/* get default evaluation */
eval = ra_eval_get_default(meas);
if (eval == NULL)
{
    printf("no evaluation in measurement %s\n", argv[1]);
    return -1;
}

/* get plugin-handle for hrv-plugin */    ❶
pl = ra_plugin_get_by_name(ra, "hrv", 0);
if (pl == NULL)
{
    printf("can't find plugin 'hrv'\n");
    return -1;
}

/* calculate hrv-values using the hrv-plug❷in */
pi = (struct proc_info *)ra_proc_get(meas, pl, NULL);
if (ra_proc_do(pi) == 0)
{
    long num_res_sets, nu❸m_results, m, l;
    value_handle vh;

    /* get number of result-sets */
    vh = ra_value_malloc();
    if (ra_info_get(pi, RA_INFO_PROC_NUM_RES_SETS_L, vh) !=
        0)
    {
        printf("no result-sets\n");
        return -1;
    }
    num_res_sets = ra_value_get_long(vh);
```

```
    /* get number of results */
    if (ra_info_get(pi, RA_INFO_PROC_NUM_RES_L, vh) != 0)
    {
        printf("no results\n");
        return -1;
    }
    num_results = ra_value_get_long(❹vh);

    for (m = 0; m < num_res_sets; m++)
    {
        printf("result-set #%d:\n", m + 1);

        for (l = 0; l < num_results; l++)
        {
            char out[200], t[100];

            /* set number of result in which we are interested */
            ra_value_set_number(vh, l);

            /* test if result is a default value (some
               non-default results are arrays which we skip in
               this example) */
            ra_info_get(pl, RA_INFO_PL_RES_DEFAULT_L, vh);
            if (ra_value_get_long(vh) == 0)
                continue;

            out[0] = '\0';                    ❺
            if (ra_info_get(pl, RA_INFO_PL_RES_NAME_C, vh) ==
                0)
            {
                strcpy(t, ra_value_get_string(vh));
                strcat(out, t);
            }
            if (ra_info_get(pl, RA_INFO_PL_RES_DESC_C, vh) ==
                0)
            {
                sprintf(t, " (%s)", ra_value_get_string(vh));
                strcat(out, t);
            }
            if (ra_proc_get_result(pi, l, m, vh) == 0)
            {
                sprintf(t, ": %lf", ra_value_get_double(vh));
                strcat(out, t);
            }

            printf("  %s\n", out);
        }
    }

    ra_value_free(vh);
}

/* close */
```

```
    ra_proc_free(pi);
    ra_meas_close(meas);
    ra_lib_close(ra);

    return 0;
}                                /* main() */

//
```

❶ Returns the plugin-handle for the plugin 'hrv'.

❷ Initialize the proc_info structure (defined in ra_defines.h). In this structure must be the information set, which will be needed for the processing (the next 3 lines).

Not all process-plugins need the information of the recording and/or the evaluation. Please check the plugin-reference section which information is needed. In case of doubt set all variables.

❸ This function-call starts the processing. If the processing is successfull, the function returns 0.

❹ To select the result, in which you are interessted, the variable 'res_num' in the ra_info structure must be set. The value of the result as well as information about the result will be returned using a ra_info structure.

❺ Information about a result will be obtained using ra_get_info_id(). The value of the result will be obtained using ra_get_result().

The output of the above example for the ECG '100s' is shown here:

```
result-set #1:
  SDNN (standard deviation of normal-to-normal intervals):
  30.238227
  HRVI (HRV-Index): 7.200000
  SDANN (standard deviation of averaged normal-to-normal
  intervals): nan
  rmssd (root mean of squared sucsessive differences): 32.494403
  pNN50 (): 7.142857
  TP (total power): 476.707920
  ULF (ultra low frequency power): 0.000000
  VLF (very low frequency power of short-term recordings):
  31.580847
  LF (low frequency power): 157.664788
  LF_NORM (normalised low frequency power): 35.420175
  HF (high freuqency power): 287.462286
  HF_NORM (normalised high frequency power): 64.579825
  LF_HF_RATIO (LF/HF ratio): 0.548471
  POWER_LAW (power law behavior): 0.000000
  SD1 (SD1 of the Poincare Plot): 15.845239
  SD2 (SD2 of the Poincare Plot): 17.873049
  DFA (overall DFA Alpha): nan
  DFA_OFFSET (offset of the overall DFA Alpha slope): nan
```

```
DFA1 (DFA Alpha1): 0.242035
DFA1_OFFSET (offset of the DFA Alpha-1 slope): 1.404195
DFA2 (DFA Alpha2): nan
DFA2_OFFSET (offset of the DFA Alpha-2 slope): nan
DFA_USER (DFA Alpha of user-defined range)
DFA_USER_OFFSET (offset of the user-defined DFA Alpha slope)
```

Notes

1. Some access plugins do not differ between fast and normal open. But it is recommended, when writing access plugins, to skip time consuming tasks when fast-open was selected. This decreases the time needed to list all measurements in a directory.

Appendix A. Examples for all supported lanuages/systems

This appendix shows the examples presented in Chapter 3 for all supported languages/systems.

A.1. Init libRASCH

A.1.1. C Version

```
//
#include <stdio.h>
#include <ra.h>

int main(int argc, char *argv[])
{
    ra_handle ra;
    value_handle vh;
    ra_find_handle f;
    struct ra_find_struct fs;

    /* initialize libRASCH */
    ra = ra_lib_init();

    /* check if init was successful */
    if ((ra == NULL)
        || (ra_lib_get_error(ra, NULL, 0) != RA_ERR_NONE))
    {
        if (!ra)
            printf("error initializing libRASCH\n");
        else
        {
            char err_t[200];
            long err_num;

            err_num = ra_lib_get_error(ra, err_t, 200);
            printf("while initializing libRASCH, error #%d "
                   "occured\n  %s\n", err_num, err_t);

            ra_lib_close(ra);
        }
        return -1;
    }

    /* get some infos */
    vh = ra_value_malloc();
```

```
    if (ra_info_get(ra, RA_INFO_NUM_PLUGINS_L, vh) == 0)
    {
        printf("%s (%s): %d\n", ra_value_get_name(vh),
               ra_value_get_desc(vh), ra_value_get_long(vh));
    }
    ra_value_free(vh);

    /* find all measurements in a directory */
    f = ra_meas_find_first(ra, argv[1], &fs);
    if (f)
    {
        int cnt = 1;

        printf("measurements found in %s:\n", argv[1]);
        do
        {
            printf("  %2d: %s\n", cnt, fs.name);
            cnt++;
        }
        while (ra_meas_find_next(f, &fs));

        ra_meas_close_find(f);
    }

    /* close libRASCH */
    ra_lib_close(ra);

    return 0;
}                               /* main() */

//
```

```
#plugins (): 37
measurements found in ./database:
   1: ./database/100s.hea
   2: ./database/JesusOlivan2003-12-EMG2.edf
   3: ./database/100s.dat
```

A.1.2. Perl Version

```
#
use strict;
use RASCH;
```

```
# initialize libRASCH
my $ra = new RASCH or die "error initializing libRASCH\n";
my ($err_num, $err_text) = $ra->get_error ();
if ($err_num != 1)
{
    print "while initializing libRASCH, error # $err_num " .
        "occured:\n $err_text\n";
    exit -1;
}

# get some infos
my $value = $ra->get_info(info =>'num_plugins');
if ($value->is_ok())
{
    print $value->name() . " (" . $value->desc() . "): " . $value->value() . "\n";
}

# find all measurements in a directory
my $meas = $ra->find_meas($ARGV[0]);
print "measurements found in $ARGV[0]:\n";
my $cnt = 1;
for (@$meas)
{
    print "$cnt: " . $_->filename() . "\n";
    $cnt++;
}

# ra_close() will be called when $ra is being destroyed

exit 0;
#
```

```
#plugins (): 37
measurements found in ./database:
1: ./database/100s.dat
2: ./database/100s.hea
3: ./database/JesusOlivan2003-12-EMG2.edf
```

A.1.3. Python Version

```
#
import sys
from RASCH import *
```

```
# initialize libRASCH
ra = RASCH()
if not ra:
    print "can't initialize libRASCH"
    sys.exit()
[err_num, err_text] = ra.get_error()
if err_num != 1:
    print "while initializing libRASCH, error #%d occured:\n " \
          "%s\n" % err_num, err_text
    sys.exit()

# get some infos
value = ra.get_info(info='num_plugins')
if (value.is_ok()):
    print value.name(), "("+value.desc()+"):", value.value()

# find all measurements in a directory
meas = ra.find_meas(sys.argv[1])
print "measurements found in " + sys.argv[1] + ":\n"
cnt = 1
for item in meas:
    print "%d: %s" % (cnt, item.filename())
    cnt = cnt + 1

#
```

```
#plugins (): 37
measurements found in ./database:

1: ./database/100s.dat
2: ./database/100s.hea
3: ./database/JesusOlivan2003-12-EMG2.edf
```

A.1.4. Matlab/Octave Version

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".
```

```
Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit
http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).

For information about changes from previous versions, type
'news'.

+ ra=ra_lib_init
ra =  162635120
+ [err_num, err_text]=ra_lib_get_error(ra)
err_num =  1
err_text = libRASCH (ra.c:109): no error
+ [value, name, desc]=ra_lib_info_get(ra, 'num_plugins')
value =  37
name = #plugins
desc =
+ meas=ra_meas_find(ra, './database')
meas =

{
  [1,1] = ./database/100s.hea
  [2,1] = ./database/JesusOlivan2003-12-EMG2.edf
  [3,1] = ./database/100s.dat
}

+ ra_lib_close(ra);
```

A.2. Open measurement

A.2.1. C Version

```
//
#include <stdio.h>
#include <ra.h>

int main(int argc, char *argv[])
{
    ra_handle ra;
```

```
value_handle vh;
meas_handle meas;

/* initialize libRASCH */
ra = ra_lib_init();
if ((ra == NULL)
    || (ra_lib_get_error(ra, NULL, 0) != RA_ERR_NONE))
{
    if (!ra)
        printf("error initializing libRASCH\n");
    else
    {
        char err_t[200];
        long err_num;

        err_num = ra_lib_get_error(ra, err_t, 200);
        printf("while initializing libRASCH, error #%d "
               "occured\n  %s\n", err_num, err_t);

        ra_lib_close(ra);
    }
    return -1;
}

/* open measurement */
meas = ra_meas_open(ra, argv[1], NULL, 0);
if (meas == NULL)
{
    char err_t[200];
    long err_num;

    err_num = ra_lib_get_error(ra, err_t, 200);
    printf("can't open measurement %s\nerror #%d: %s\n",
           argv[1], err_num, err_t);
    ra_lib_close(ra);
    return -1;
}

/* get some infos */
vh = ra_value_malloc();
if (ra_info_get(meas, RA_INFO_NUM_SESSIONS_L, vh) == 0)
    printf("%s (%s): %d\n", ra_value_get_name(vh),
           ra_value_get_desc(vh), ra_value_get_long(vh));
if (ra_info_get(meas, RA_INFO_MAX_SAMPLERATE_D, vh) == 0)
    printf("%s (%s): %f\n", ra_value_get_name(vh),
           ra_value_get_desc(vh), ra_value_get_double(vh));

/* get all measurement-object infos */
if (ra_info_get(meas, RA_INFO_NUM_OBJ_INFOS_L, vh) == 0)
{
    long l;
    long n = ra_value_get_long(vh);
```

```
        for (l = 0; l < n; l++)
        {
            ra_info_get_by_idx(meas, RA_INFO_OBJECT, l, vh);
            printf("%s (%s): ", ra_value_get_name(vh),
                   ra_value_get_desc(vh));
            switch (ra_value_get_type(vh))
            {
            case RA_VALUE_TYPE_LONG:
                printf("%d\n", ra_value_get_long(vh));
                break;
            case RA_VALUE_TYPE_DOUBLE:
                printf("%f\n", ra_value_get_double(vh));
                break;
            case RA_VALUE_TYPE_CHAR:
                printf("%s\n", ra_value_get_string(vh));
                break;
            default:
                printf("not supported type\n");
                break;
            }
        }
    }

    /* close */
    ra_value_free(vh);
    ra_meas_close(meas);
    ra_lib_close(ra);

    return 0;
}                               /* main() */

//
```

```
#sessions (): 1
max. samplerate (maximum samplrate used in measurement):
360.000000
ID (Patient-ID): 100s
```

A.2.2. Perl Version

```
#
use strict;
use RASCH;
```

```
# initialize libRASCH
my $ra = new RASCH or die "error initializing libRASCH\n";

# open measurement
my $meas = $ra->open_meas($ARGV[0], ”, 0) or
    die "can’t open measurement $ARGV[0]\n";

# get some infos
my $v = $meas->get_info(info => ’num_sessions’);
print $v->name() . ’ (’ . $v->desc() . ’): ’ . $v->value() ."\n" if (defined($v));
$v = $meas->get_info(info => ’max_samplerate’);
print $v->name() . ’ (’ . $v->desc() . ’): ’ . $v->value() ."\n" if (defined($v));

# get all measurement-object infos
$v = $meas->get_info(info => ’num_obj_infos’);
my $num = $v->value();
for (my $i = 0; $i < $num; $i++)
{
    $v = $meas->get_info_idx(index => $i);
    print $v->name() . ’ (’ . $v->desc() . ’): ’ . $v->value() ."\n" if (defined($v));
}

exit 0;
#
```

```
#sessions (): 1
max. samplerate (maximum samplrate used in measurement): 360
```

A.2.3. Python Version

```
#
import sys
from RASCH import *

# initialize libRASCH
ra = RASCH()
if not ra:
    print "can’t initialize libRASCH"
    sys.exit()

# open measurement
meas = ra.open_meas(sys.argv[1], ”, 0)
if not meas:
    print "can’t open measurement", sys.argv[1]
```

```
    sys.exit()

# get some infos
value = meas.get_info(info='num_sessions')
if value.is_ok():
    print value.name(), "("+value.desc()+"):", value.value()
value = meas.get_info(info='max_samplerate')
if value.is_ok():
    print value.name(), "("+value.desc()+"):", value.value()

# get all measurement-object infos
num = meas.get_info(info='num_obj_infos')
for i in range(num.value()):
    value = meas.get_info_idx(index=i)
    if value.is_ok():
        print value.name(), "("+value.desc()+"):", value.value()

#
```

```
#sessions (): 1
max. samplerate (maximum samplrate used in measurement): 360.0
ID (Patient-ID): 100s
```

A.2.4. Matlab/Octave Version

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".

Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit
http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).
```

```
For information about changes from previous versions, type
'news'.

+ ra=ra_lib_init
ra =  177749312
+ meas=ra_meas_open(ra, './database/100s.hea', '', 0)
meas =  178286696
+ [v, n, d]=ra_meas_info_get(meas, 'num_sessions')
v =  1
n = #sessions
d =
+ [v, n, d]=ra_meas_info_get(meas, 'max_samplerate')
v =  360
n = max. samplerate
d = maximum samplrate used in measurement
+ num=ra_meas_info_get(meas, 'num_obj_infos')
num =  1
+ for i=0:(num-1)
+     [v,n,d]=ra_info_get_by_idx(meas, 'meas', i)
+ endfor
v = 100s
n = ID
d = Patient-ID
+ ra_meas_close(meas);
+ ra_lib_close(ra);
```

A.3. Handle recordings

A.3.1. C Version

```
//
#include <stdio.h>
#include <ra.h>

int main(int argc, char *argv[])
{
    ra_handle ra;
    value_handle vh;
    meas_handle meas;
    rec_handle rec;
    long l, num_dev, num_ch;

    /* initialize libRASCH */
    ra = ra_lib_init();
```

```
if (ra == NULL)
{
    printf("error initializing libRASCH\n");
    return -1;
}

/* open measurement */
meas = ra_meas_open(ra, argv[1], NULL, 0);
if (meas == NULL)
{
    printf("can't open measurement %s\n", argv[1]);
    return -1;
}

/* get root recording */
rec = ra_rec_get_first(meas, 0);
if (rec == NULL)
{
    printf("can't get recording-handle\n");
    return -1;
}

/* get some infos about recording */
num_dev = num_ch = 0;
vh = ra_value_malloc();
if (ra_info_get(rec, RA_INFO_REC_GEN_NUM_DEVICES_L, vh) == 0)
    num_dev = ra_value_get_long(vh);
if (ra_info_get(rec, RA_INFO_REC_GEN_NUM_CHANNEL_L, vh) == 0)
    num_ch = ra_value_get_long(vh);
if (ra_info_get(rec, RA_INFO_REC_GEN_NAME_C, vh) == 0)
    printf("measurement %s\n", ra_value_get_string(vh));
if (ra_info_get(rec, RA_INFO_REC_GEN_DATE_C, vh) == 0)
    printf("recorded at %s\n", ra_value_get_string(vh));
printf("#devices=%d\n#channels=%d\n\n", num_dev, num_ch);

/* print name for every device */
printf("infos about the recording devices used:\n");
for (l = 0; l < num_dev; l++)
{
    /* set number of device from which the info is wanted */
    ra_value_set_number(vh, l);
    if (ra_info_get(rec, RA_INFO_REC_DEV_HW_NAME_C, vh) == 0)
        printf("  device #%d: %s\n", l,
               ra_value_get_string(vh));
}
printf("\n");

/* print name and unit of every channel */
printf("infos about the channels:\n");
for (l = 0; l < num_ch; l++)
{
    /* set number of channel */
    ra_value_set_number(vh, l);
```

```
        if (ra_info_get(rec, RA_INFO_REC_CH_NAME_C, vh) == 0)
            printf("  ch #%d: %s", l, ra_value_get_string(vh));
        if (ra_info_get(rec, RA_INFO_REC_CH_UNIT_C, vh) == 0)
            printf(" [%s]", ra_value_get_string(vh));
        printf("\n");
    }
    printf("\n");

    /* close */
    ra_value_free(vh);
    ra_meas_close(meas);
    ra_lib_close(ra);

    return 0;
}                               /* main() */

//
```

```
measurement 100s
recorded at 00.00.0
#devices=0
#channels=2

infos about the recording devices used:

infos about the channels:
  ch #0: MLII []
  ch #1: V5 []
```

A.3.2. Perl Version

```
#
use strict;
use RASCH;

# initialize libRASCH
my $ra = new RASCH or die "error initializing libRASCH\n";

# open measurement
my $meas = $ra->open_meas($ARGV[0], ”, 0) or
    die "can’t open measurement $ARGV[0]\n";

# get root recording
```

```
my $rec = $meas->get_first_rec(0) or
    die "can't get root recording\n";

# get some infos about recording
my $v = $rec->get_info(info => 'rec_num_devices');
my $num_dev = $v->value();
$v = $rec->get_info(info => 'rec_num_channel');
my $num_ch = $v->value();
$v = $rec->get_info(info => 'rec_name');
my $rec_name = $v->value();
$v = $rec->get_info(info => 'rec_date');
my $rec_date = $v->value;
print "measurement $rec_name\nrecorded at $rec_date\n" .
    "#devices=$num_dev\n#channels=$num_ch\n\n";

# print name for every device
print "infos about the recording devices used:\n";
for (my $i = 0; $i < $num_dev; $i++)
{
    $v = $rec->get_info(dev => $i, info => 'dev_hw_name');
    my $name = $v->value();
    print "  device #$i: $name\n";
}
print "\n";

# print name for every channel
print "infos about the channels:\n";
for (my $i = 0; $i < $num_ch; $i++)
{
    $v = $rec->get_info(ch => $i, info => 'ch_name');
    my $name = $v->value();
    $v = $rec->get_info(ch => $i, info => 'ch_unit');
    my $unit = $v->value();
    print "  channel #$i: $name [$unit]\n";
}
print "\n";

exit 0;
#
```

```
measurement 100s
recorded at 00.00.0
#devices=
#channels=2

infos about the recording devices used:

infos about the channels:
  channel #0: MLII []
```

```
  channel #1: V5 []
```

A.3.3. Python Version

```
#
import sys
from RASCH import *

# initialize libRASCH
ra = RASCH()
if not ra:
    print "can't initialize libRASCH"
    sys.exit()

# open measurement
meas = ra.open_meas(sys.argv[1], '', 0)
if not meas:
    print "can't open measurement", sys.argv[1]
    sys.exit()

# get root recording
rec = meas.get_first_rec(0)
if not rec:
    print "can't get root recording"
    sys.exit()

# get some infos about recording
num_dev = rec.get_info(info='rec_num_devices')
num_ch = rec.get_info(info='rec_num_channel')
rec_name = rec.get_info(info='rec_name')
rec_date = rec.get_info(info='rec_date')
print "measurement", rec_name.value()
print "recorded at", rec_date.value()
print "#devices =", num_dev.value()
print "#channels =", num_ch.value()
print

# print name for every device
print "infos about the recording devices used:"
if num_dev.value() > 0:
    for i in range(num_dev.value()):
        name = rec.get_info(dev=i, info='dev_hw_name')
        print "  device #%d: %s" % (i, name.value())
print ""

# print name for every channel
print "infos about the channels:";
```

```
if num_ch.value() > 0:
    for i in range(num_ch.value()):
        name = rec.get_info(ch=i, info='ch_name')
        unit = rec.get_info(ch=i, info='ch_unit')
        print "  channel #%d: %s [%s]" % (i, name.value(), unit.value())
print "";

#
```

```
measurement 100s
recorded at 00.00.0
#devices = None
#channels = 2

infos about the recording devices used:

infos about the channels:
  channel #0: MLII []
  channel #1: V5 []
```

A.3.4. Matlab/Octave Version

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".

Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit
http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).

For information about changes from previous versions, type
```

```
'news'.

+ ra=ra_lib_init
ra =  183558320
+ meas=ra_meas_open(ra, './database/100s.hea', '', 0)
meas =  184102104
+ rec=ra_rec_get_first(meas, 0)
rec =  184108552
+ num_ch=ra_rec_info_get(rec, 'rec_num_channel')
num_ch =  2
+ rec_name=ra_rec_info_get(rec, 'rec_name')
rec_name = 100s
+ rec_date=ra_rec_info_get(rec, 'rec_date')
rec_date = 00.00.0
+ for i=0:(num_ch-1)
+     name=ra_ch_info_get(rec, i, 'ch_name')
+     unit=ra_ch_info_get(rec, i, 'ch_unit')
+ endfor
name = MLII
unit =
name = V5
unit =
+ ra_meas_close(meas);
+ ra_lib_close(ra);
```

A.4. Access raw data

A.4.1. C Version

```
//
#include <stdio.h>
#include <stdlib.h>
#include <ra.h>

int main(int argc, char *argv[])
{
    ra_handle ra;
    value_handle vh;
    meas_handle meas;
    rec_handle rec;
    long l, num_ch;
    double *buf = NULL;

    /* initialize libRASCH */
    ra = ra_lib_init();
```

```
    if (ra == NULL)
    {
        printf("error initializing libRASCH\n");
        return -1;
    }

    /* open measurement */
    meas = ra_meas_open(ra, argv[1], NULL, 0);
    if (meas == NULL)
    {
        printf("can't open measurement %s\n", argv[1]);
        return -1;
    }

    /* get root recording */
    rec = ra_rec_get_first(meas, 0);
    if (rec == NULL)
    {
        printf("can't get recording-handle\n");
        return -1;
    }

    /* get first 10000 samples for each channel */
    vh = ra_value_malloc();
    if (ra_info_get(rec, RA_INFO_REC_GEN_NUM_CHANNEL_L, vh) == 0)
        num_ch = ra_value_get_long(vh);
    buf = malloc(sizeof(double) * 10000);
    for (l = 0; l < num_ch; l++)
    {
        long m, num_read;

        num_read = ra_raw_get_unit(rec, l, 0, 10000, buf);
        for (m = 0; m < num_read; m++)
            ;                   /* do something with every sample */
    }

    /* clean up */
    free(buf);
    ra_value_free(vh);
    ra_meas_close(meas);
    ra_lib_close(ra);

    return 0;
}                               /* main() */

//
```

A.4.2. Perl Version

```
#
use strict;
use RASCH;

# initialize libRASCH
my $ra = new RASCH or die "error initializing libRASCH\n";

# open measurement
my $meas = $ra->open_meas($ARGV[0], 0) or
    die "can't open measurement $ARGV[0]\n";

# get root recording
my $rec = $meas->get_first_session_rec(0) or
    die "can't get root recording\n";

# get first 10000 samples for each channel
my ($num_ch) = $rec->get_info(info => 'rec_num_channel');
for (my $i = 0; $i < $num_ch; $i++)
{
    my $data_ref = $rec->get_raw($i, 0, 10000);
    for (@$data_ref)
    {
 ; # do something with every sample
    }
}

exit 0;
#
```

A.4.3. Python Version

```
#
import sys
from RASCH import *

# initialize libRASCH
ra = RASCH()
if not ra:
    print "can't initialize libRASCH"
    sys.exit()

# open measurement
meas = ra.open_meas(sys.argv[1], 0)
if not meas:
```

```
    print "can't open measurement", sys.argv[1]
    sys.exit()

# get root recording
rec = meas.get_first_session_rec(0)
if not rec:
    print "can't get root recording"
    sys.exit()

# get first 10000 samples for each channel
[num_ch, n, d] = rec.get_info(info='rec_num_channel')
if num_ch > 0:
    for i in range(num_ch):
        data = rec.get_raw(i, 0, 10000)
        for elem in data:
            elem # do something with every sample

#
```

A.4.4. Matlab/Octave Version

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".

Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit
http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).

For information about changes from previous versions, type
'news'.

+ ra=ra_lib_init
ra =  185012408
+ meas=ra_meas_open(ra, './database/100s.hea', '', 0)
meas =  185820352
```

```
+ rec=ra_rec_get_first(meas, 0)
rec =  184808528
+ num_ch=ra_rec_info_get(rec, 'rec_num_channel')
num_ch =  2
+ ch_all=[];
+ for i=0:(num_ch-1)
+   ch=ra_raw_get_unit(rec, i, 0, 10000);
+   ch_all=[ch_all ch'];
+ endfor
+ whos ch_all

*** local user variables:

  Prot Name        Size                     Bytes  Class
  ==== ====        ====                     =====  =====
   rwd ch_all  10000x2                    160000  double

Total is 20000 elements using 160000 bytes

+ samplerate=ra_ch_info_get(rec, 0, 'ch_samplerate')
samplerate =  360
+ x=0:9999;
+ x = x / samplerate;
+
+ figure();
+ for i=1:num_ch
+   subplot(num_ch,1,i)
+   plot(x,ch_all(:,i))
+ endfor
+
+ ra_meas_close(meas);
+ ra_lib_close(ra);
```

A.5. Access evaluation

A.5.1. C Version

```
//
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ra.h>

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

```
ra_handle ra;
value_handle vh;
meas_handle meas;
eval_handle eval;
long l, m, num_class, num_prop;
class_handle *clh = NULL;
prop_handle *ph = NULL;

/* initialize libRASCH */
ra = ra_lib_init();
if (ra == NULL)
{
    printf("error initializing libRASCH\n");
    return -1;
}

/* open measurement */
meas = ra_meas_open(ra, argv[1], NULL, 0);
if (meas == NULL)
{
    printf("can't open measurement %s\n", argv[1]);
    return -1;
}

/* get default evaluation */
eval = ra_eval_get_default(meas);
if (eval == NULL)
{
    printf("no evaluation in measurement %s\n", argv[1]);
    return -1;
}

/* get some infos about evaluation */
vh = ra_value_malloc();
if (ra_info_get(eval, RA_INFO_EVAL_NAME_C, vh) == 0)
    printf("evaluation %s ", ra_value_get_string(vh));
if (ra_info_get(eval, RA_INFO_EVAL_ADD_TS_C, vh) == 0)
    printf("was added at %s ", ra_value_get_string(vh));
if (ra_info_get(eval, RA_INFO_EVAL_PROG_C, vh) == 0)
    printf("using the program %s", ra_value_get_string(vh));
printf("\n\n");

/* list event-class's */
num_class = 0;
if (ra_class_get(eval, NULL, vh) == 0)
{
    num_class = ra_value_get_num_elem(vh);
    clh = malloc(sizeof(class_handle) * num_class);
    memcpy(clh, (void *)ra_value_get_voidp_array(vh),
           sizeof(class_handle) * num_class);
}

for (l = 0; l < num_class; l++)
```

```
    {
        if (ra_info_get(clh[l], RA_INFO_CLASS_NAME_C, vh) == 0)
            printf("event-class %s", ra_value_get_string(vh));
        if (ra_info_get(clh[l], RA_INFO_CLASS_EV_NUM_L, vh) == 0)
            printf(" with %d events", ra_value_get_long(vh));
        printf(":\n");

        /* list event-properties */
        num_prop = 0;
        if (ra_prop_get_all(clh[l], vh) == 0)
        {
            num_prop = ra_value_get_num_elem(vh);
            ph = malloc(sizeof(prop_handle) * num_prop);
            memcpy(ph, (void *)ra_value_get_voidp_array(vh),
                   sizeof(prop_handle) * num_prop);
        }

        for (m = 0; m < num_prop; m++)
        {
            if (ra_info_get(ph[m], RA_INFO_PROP_ASCII_ID_C, vh)
                == 0)
                printf("  %s\n", ra_value_get_string(vh));
            if (ra_info_get(ph[m], RA_INFO_PROP_DESC_C, vh) == 0)
                printf("    %s", ra_value_get_string(vh));
            printf("\n");
        }

        free(ph);
    }
    free(clh);

    /* close */
    ra_value_free(vh);
    ra_meas_close(meas);
    ra_lib_close(ra);

    return 0;
}                               /* main() */

//
```

```
evaluation RASCHlab was added at 15.11.2009  11:48:00 using
the program raschlab_qt

event-class heartbeat with 75 events:
  qrs-pos
    position of fiducial point of QRS-complex in sampleunits
  qrs-annot
    annotation of QRS complex
```

```
ecg-noise

ecg-morph-flags
  flags for the morphology values
ecg-p-type

ecg-qrs-type

ecg-t-type

ecg-p-start
  offset of p-wave begin from qrs-pos in sampleunits
ecg-p-end
  offset of p-wave end from qrs-pos in sampleunits
ecg-p-width

ecg-p-peak-1
  offset of p-wave peak from qrs-pos in sampleunits
ecg-p-peak-2
  offset of 2nd p-wave peak (if biphasic) from qrs-pos
  in sampleunits
ecg-qrs-start
  offset of start of qrs complex from qrs-pos in sampleunits
ecg-qrs-end
  offset of end of qrs complex from qrs-pos in sampleunits
ecg-qrs-width

ecg-q-peak
  offset of q-wave from qrs-pos in sampleunits
ecg-r-peak
  offset of r-wave from qrs-pos in sampleunits
ecg-s-peak
  offset of s-wave from qrs-pos in sampleunits
ecg-rs-peak
  offset of r'-wave from qrs-pos in sampleunits
ecg-t-start
  offset of t-wave begin from qrs-pos in sampleunits
ecg-t-end
  offset of t-wave end from qrs-pos in sampleunits
ecg-t-width

ecg-t-peak-1
  offset of t-wave peak from qrs-pos in sampleunits
ecg-t-peak-2
  offset of 2nd t-wave peak (if biphasic) from qrs-pos
  in sampleunits
ecg-pq
  PQ interval
ecg-qt
  QT interval
ecg-qtc
  QTc interval
qrs-temporal
```

```
    temporal setting of beat
  ecg-flags
    ECG releated flags
  rri
    RR interval
  rri-annot
    annoation of RR interval
  rri-refvalue
    reference rri representing the current heart-rate
  rri-num-refvalue
    number of rri's used for calculation of reference value
```

A.5.2. Perl Version

```
#
use strict;
use RASCH;

# initialize libRASCH
my $ra = new RASCH or die "error initializing libRASCH\n";

# open measurement
my $meas = $ra->open_meas($ARGV[0], '', 0) or
    die "can't open measurement $ARGV[0]\n";

# get default evaluation
my $eval = $meas->get_default_eval() or
    die "no evaluation in measurement\n";

# get some general infos
my $v = $eval->get_info(info => 'eval_name');
my $eval_name = $v->value();
$v = $eval->get_info(info => 'eval_add_timestamp');
my $eval_add_ts = $v->value();
$v = $eval->get_info(info => 'eval_program');
my $eval_prg = $v->value();
print "evaluation $eval_name was added at $eval_add_ts" .
    " using the program $eval_prg\n\n";


# list event-class
my $cl_all = $eval->get_class();
for my $cl_curr (@$cl_all)
{
    my $name = $cl_curr->get_info(info => 'class_name');
    my $num_events = $cl_curr->get_info(info => 'class_num_events');

    print "event-class " . $name->value() . " with " . $num_events->value() . " events\n";
```

```
    # list event properties
    my $props = $cl_curr->get_prop_all();
    for my $prop (@$props)
    {
        my $name = $prop->get_info(info => 'prop_name');
        my $desc = $prop->get_info(info => 'prop_desc');

        print "  " . $name->value() . "\n";
 print "    " . $desc->value() . "\n";
    }
}

exit 0;
#
```

```
evaluation RASCHlab was added at 15.11.2009  11:48:00 using
the program raschlab_qt

event-class heartbeat with 75 events
  qrs-pos
    position of fiducial point of QRS-complex in sampleunits
  qrs-annot
    annotation of QRS complex
  ecg-noise

  ecg-morph-flags
    flags for the morphology values
  ecg-p-type

  ecg-qrs-type

  ecg-t-type

  ecg-p-start
    offset of p-wave begin from qrs-pos in sampleunits
  ecg-p-end
    offset of p-wave end from qrs-pos in sampleunits
  ecg-p-width

  ecg-p-peak-1
    offset of p-wave peak from qrs-pos in sampleunits
  ecg-p-peak-2
    offset of 2nd p-wave peak (if biphasic) from qrs-pos
    in sampleunits
  ecg-qrs-start
    offset of start of qrs complex from qrs-pos in sampleunits
  ecg-qrs-end
    offset of end of qrs complex from qrs-pos in sampleunits
```

```
  ecg-qrs-width

  ecg-q-peak
    offset of q-wave from qrs-pos in sampleunits
  ecg-r-peak
    offset of r-wave from qrs-pos in sampleunits
  ecg-s-peak
    offset of s-wave from qrs-pos in sampleunits
  ecg-rs-peak
    offset of r'-wave from qrs-pos in sampleunits
  ecg-t-start
    offset of t-wave begin from qrs-pos in sampleunits
  ecg-t-end
    offset of t-wave end from qrs-pos in sampleunits
  ecg-t-width

  ecg-t-peak-1
    offset of t-wave peak from qrs-pos in sampleunits
  ecg-t-peak-2
    offset of 2nd t-wave peak (if biphasic) from qrs-pos
    in sampleunits
  ecg-pq
    PQ interval
  ecg-qt
    QT interval
  ecg-qtc
    QTc interval
  qrs-temporal
    temporal setting of beat
  ecg-flags
    ECG releated flags
  rri
    RR interval
  rri-annotation
    annoation of RR interval
  rri-refvalue
    reference rri representing the current heart-rate
  rri-num-refvalue
    number of rri's used for calculation of reference value
```

A.5.3. Python Version

```
#
import sys
from RASCH import *

# initialize libRASCH
ra = RASCH()
```

```
if not ra:
    print "can't initialize libRASCH"
    sys.exit()

# open measurement
meas = ra.open_meas(sys.argv[1], '', 0)
if not meas:
    print "can't open measurement", sys.argv[1]
    sys.exit()

# get default evaluation
eva = meas.get_default_eval()
if not eva:
    print "no evaluation in measurement"
    sys.exit()

# get some general infos
eval_name = eva.get_info(info='eval_name')
eval_add_ts = eva.get_info(info='eval_add_timestamp')
eval_prg = eva.get_info(info='eval_program')
print "evaluation", eval_name.value(), "was added at", eval_add_ts.value(), \
      "using the program", eval_prg.value(), "\n"

# list event-class
cl = eva.get_class()
for cl_curr in cl:
    name = cl_curr.get_info(info='class_name')
    num_events = cl_curr.get_info(info='class_num_events')

    print "event-class", name.value(), "with",
    print num_events.value(),
    print "events"

    # list event properties
    props = cl_curr.get_prop_all()
    for elem in props:
        name = elem.get_info(info='prop_name')
        desc = elem.get_info(info='prop_desc')

        print " ", name.value()
        print "   ", desc.value()

#
```

```
evaluation RASCHlab was added at 15.11.2009  11:48:00 using
the program raschlab_qt

event-class heartbeat with 75 events
  qrs-pos
```

```
  position of fiducial point of QRS-complex in sampleunits
qrs-annot
  annotation of QRS complex
ecg-noise

ecg-morph-flags
  flags for the morphology values
ecg-p-type

ecg-qrs-type

ecg-t-type

ecg-p-start
  offset of p-wave begin from qrs-pos in sampleunits
ecg-p-end
  offset of p-wave end from qrs-pos in sampleunits
ecg-p-width

ecg-p-peak-1
  offset of p-wave peak from qrs-pos in sampleunits
ecg-p-peak-2
  offset of 2nd p-wave peak (if biphasic) from qrs-pos
  in sampleunits
ecg-qrs-start
  offset of start of qrs complex from qrs-pos in sampleunits
ecg-qrs-end
  offset of end of qrs complex from qrs-pos in sampleunits
ecg-qrs-width

ecg-q-peak
  offset of q-wave from qrs-pos in sampleunits
ecg-r-peak
  offset of r-wave from qrs-pos in sampleunits
ecg-s-peak
  offset of s-wave from qrs-pos in sampleunits
ecg-rs-peak
  offset of r'-wave from qrs-pos in sampleunits
ecg-t-start
  offset of t-wave begin from qrs-pos in sampleunits
ecg-t-end
  offset of t-wave end from qrs-pos in sampleunits
ecg-t-width

ecg-t-peak-1
  offset of t-wave peak from qrs-pos in sampleunits
ecg-t-peak-2
  offset of 2nd t-wave peak (if biphasic) from qrs-pos
  in sampleunits
ecg-pq
  PQ interval
ecg-qt
  QT interval
```

```
  ecg-qtc
    QTc interval
  qrs-temporal
    temporal setting of beat
  ecg-flags
    ECG releated flags
  rri
    RR interval
  rri-annotation
    annoation of RR interval
  rri-refvalue
    reference rri representing the current heart-rate
  rri-num-refvalue
    number of rri's used for calculation of reference value
```

A.5.4. Matlab/Octave Version

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".

Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit
http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).

For information about changes from previous versions, type
'news'.

+ ra=ra_lib_init
ra =  172723872
+ meas=ra_meas_open(ra, './database/100s.hea', '', 0)
meas =  173268312
+ eva=ra_eval_get_default(meas)
eva =  174110536
+ ra_eval_info_get(eva, 'eval_name')
ans = RASCHlab
```

```
+ ra_eval_info_get(eva, 'eval_add_timestamp')
ans = 15.11.2009  11:48:00
+ ra_eval_info_get(eva, 'eval_program')
ans = raschlab_qt
+ cl = ra_class_get(eva, '');
+ num_cl = length(cl)
num_cl =  1
+ for i = 1:num_cl
+   name_cl = ra_class_info_get(cl(i), 'class_name')
+   desc_cl = ra_prop_info_get(cl(i), 'class_desc')
+   num_events = ra_class_info_get(cl(i), 'class_num_events')
+
+   props = ra_prop_get_all(cl(i));
+   num_prop = length(props)
+   for j = 1:num_prop
+     id_ascii = ra_prop_info_get(props(j), 'prop_id_ascii')
+     desc_prop = ra_prop_info_get(props(j), 'prop_desc')
+   endfor
+ endfor
name_cl = heartbeat
desc_cl = infos about a heart beat
num_events =  75
num_prop =  33
id_ascii = qrs-pos
desc_prop = position of fiducial point of QRS-complex in
sampleunits
id_ascii = qrs-annot
desc_prop = annotation of QRS complex
id_ascii = ecg-noise
desc_prop =
id_ascii = ecg-morph-flags
desc_prop = flags for the morphology values
id_ascii = ecg-p-type
desc_prop =
id_ascii = ecg-qrs-type
desc_prop =
id_ascii = ecg-t-type
desc_prop =
id_ascii = ecg-p-start
desc_prop = offset of p-wave begin from qrs-pos in sampleunits
id_ascii = ecg-p-end
desc_prop = offset of p-wave end from qrs-pos in sampleunits
id_ascii = ecg-p-width
desc_prop =
id_ascii = ecg-p-peak-1
desc_prop = offset of p-wave peak from qrs-pos in sampleunits
id_ascii = ecg-p-peak-2
desc_prop = offset of 2nd p-wave peak (if biphasic) from
qrs-pos in sampleunits
id_ascii = ecg-qrs-start
desc_prop = offset of start of qrs complex from qrs-pos in
sampleunits
id_ascii = ecg-qrs-end
```

```
desc_prop = offset of end of qrs complex from qrs-pos in
sampleunits
id_ascii = ecg-qrs-width
desc_prop =
id_ascii = ecg-q-peak
desc_prop = offset of q-wave from qrs-pos in sampleunits
id_ascii = ecg-r-peak
desc_prop = offset of r-wave from qrs-pos in sampleunits
id_ascii = ecg-s-peak
desc_prop = offset of s-wave from qrs-pos in sampleunits
id_ascii = ecg-rs-peak
desc_prop = offset of r'-wave from qrs-pos in sampleunits
id_ascii = ecg-t-start
desc_prop = offset of t-wave begin from qrs-pos in sampleunits
id_ascii = ecg-t-end
desc_prop = offset of t-wave end from qrs-pos in sampleunits
id_ascii = ecg-t-width
desc_prop =
id_ascii = ecg-t-peak-1
desc_prop = offset of t-wave peak from qrs-pos in sampleunits
id_ascii = ecg-t-peak-2
desc_prop = offset of 2nd t-wave peak (if biphasic) from
qrs-pos in sampleunits
id_ascii = ecg-pq
desc_prop = PQ interval
id_ascii = ecg-qt
desc_prop = QT interval
id_ascii = ecg-qtc
desc_prop = QTc interval
id_ascii = qrs-temporal
desc_prop = temporal setting of beat
id_ascii = ecg-flags
desc_prop = ECG releated flags
id_ascii = rri
desc_prop = RR interval
id_ascii = rri-annot
desc_prop = annoation of RR interval
id_ascii = rri-refvalue
desc_prop = reference rri representing the current heart-rate
id_ascii = rri-num-refvalue
desc_prop = number of rri's used for calculation of reference
value
+
+ ra_meas_close(meas);
+ ra_lib_close(ra);
```

A.6. Access events

A.6.1. C Version

```
//
#include <stdio.h>
#include <stdlib.h>
#include <ra.h>

int main(int argc, char *argv[])
{
    ra_handle ra;
    meas_handle meas;
    eval_handle eval;
    class_handle *clh;
    prop_handle prop_rri, prop_qrs_pos;
    long l, m, num_heartbeat, num_events, num_rri;
    const long *ev_ids;
    double *rri = NULL;
    double *qrs_pos = NULL;
    value_handle vh, vh_id, value;

    /* initialize libRASCH */
    ra = ra_lib_init();
    if (ra == NULL)
    {
        printf("error initializing libRASCH\n");
        return -1;
    }

    /* open measurement */
    meas = ra_meas_open(ra, argv[1], NULL, 0);
    if (meas == NULL)
    {
        printf("can't open measurement %s\n", argv[1]);
        return -1;
    }

    /* get default evaluation */
    eval = ra_eval_get_default(meas);
    if (eval == NULL)
    {
        printf("no evaluation in measurement %s\n", argv[1]);
        return -1;
    }

    vh = ra_value_malloc();
    if (ra_class_get(eval, "heartbeat", vh) != 0)
    {
        printf
```

```
        ("no 'heartbeat' event-classes in measurement %s\n",
         argv[1]);
    return -1;
}
num_heartbeat = ra_value_get_num_elem(vh);
clh = (class_handle *) ra_value_get_voidp_array(vh);
vh_id = ra_value_malloc();
value = ra_value_malloc();
for (l = 0; l < num_heartbeat; l++)
{
    if (ra_class_get_events(clh[l], -1, -1, 0, 1, vh_id) !=
        0)
    {
        printf("error getting event-id's\n");
        continue;
    }
    ev_ids = ra_value_get_long_array(vh_id);
    num_events = ra_value_get_num_elem(vh_id);

    /* get event-properties for RR-intervals and position of
       QRS-complex */
    prop_rri = ra_prop_get(clh[l], "rri");
    if (prop_rri == NULL)
    {
        printf("no event-property 'rri' in event-class\n");
        continue;
    }
    prop_qrs_pos = ra_prop_get(clh[l], "qrs-pos");
    if (prop_qrs_pos == NULL)
    {
        printf
            ("no event-property 'qrs-pos' in event-class\n");
        continue;
    }

    rri = malloc(sizeof(double) * num_events);
    qrs_pos = malloc(sizeof(double) * num_events);
    num_rri = 0;
    for (m = 0; m < num_events; m++)
    {
        if (ra_prop_get_value(prop_rri, ev_ids[m], -1, value)
            != 0)
            continue;
        rri[num_rri] = ra_value_get_double(value);

        if (ra_prop_get_value
            (prop_qrs_pos, ev_ids[m], -1, value) != 0)
            continue;
        qrs_pos[num_rri] = ra_value_get_double(value);

        num_rri++;
    }
```

```
        /* now do something with the RR-intervals and QRS-complex
           positions */

        if (rri)
            free(rri);
        if (qrs_pos)
            free(qrs_pos);
    }

    /* clean up */
    ra_value_free(vh);
    ra_value_free(vh_id);
    ra_value_free(value);

    ra_meas_close(meas);
    ra_lib_close(ra);

    return 0;
}                               /* main() */

//
```

A.6.2. Perl Version

```
#
use strict;
use RASCH;

# initialize libRASCH
my $ra = new RASCH or die "error initializing libRASCH\n";

# open measurement
my $meas = $ra->open_meas($ARGV[0], 0) or
    die "can't open measurement $ARGV[0]\n";

# get default evaluation
my $eval = $meas->get_default_eval() or
    die "no evaluation in the measurement\n";

my $cl = $eval->get_class('heartbeat');
# we are only interested in the first 'heartbeat' event-class
# in this example
my $cl_use = $cl->[0];

# get event-properties for RR-intervals and position of QRS-complex
my $prop_rri = $cl_use->get_prop('rri') or
    die "no event-property 'rri' in the evaluation\n";
my $prop_qrs_pos = $cl_use->get_prop('qrs-pos') or
```

```
    die "no event-property 'qrs-pos' in the evaluation\n";

# get values for all RR-intervals and QRS-complexes
my $rri_ref = $prop_rri->get_events();
my $qrs_pos_ref = $prop_qrs_pos->get_events();

# now do something with the RR-intervals and QRS-complex-positions

exit 0;
#
```

A.6.3. Python Version

```
#
import sys
from RASCH import *

# initialize libRASCH
ra = RASCH()
if not ra:
    print "can't initialize libRASCH"
    sys.exit()

# open measurement
meas = ra.open_meas(sys.argv[1], 0)
if not meas:
    print "can't open measurement", sys.argv[1]
    sys.exit()

# get default evaluation
eva = meas.get_def_eval()
if not eva:
    print "no evaluation in measurement"
    sys.exit()

# get event-properties for RR-intervals and position of QRS-complexs
prop_rri = eva.get_evprop_by_name('rri')
if not prop_rri:
    print "no event-property 'rri' in the evaluation"
    sys.exit()

prop_qrs_pos = eva.get_evprop_by_name('qrs-pos')
if not prop_qrs_pos:
    print "no event-property 'qrs-pos' in the evaluation"
    sys.exit()

# get values for all RR-intervals and QRS-complexes
rri = prop_rri.get_events()
```

```
qrs_pos_ref = prop_qrs_pos.get_events()

# now do something with the RR-intervals and QRS-complex-positions

#
```

A.6.4. Matlab/Octave Version

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".

Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit
http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).

For information about changes from previous versions, type
'news'.

+ ra=ra_lib_init
ra =  184868040
+ meas=ra_meas_open(ra, './database/100s.hea', '', 0)
meas =  185403952
+ eva=ra_eval_get_default(meas)
eva =  185920976
+
+ samplerate=ra_meas_info_get(meas, 'max_samplerate')
samplerate =  360
+
+ cl = ra_class_get(eva, 'heartbeat')
cl =  185834776
+ num_hb = length(cl)
num_hb =  1
+ for i = 1:num_hb
+   ev = ra_class_get_events(cl(i), -1, -1, 0, 1);
+   num_ev = length(ev)
```

```
+
+   prop_rri = ra_prop_get(cl(i), 'rri')
+   prop_qrs_pos = ra_prop_get(cl(i), 'qrs-pos')
+
+   rri = [];
+   qrs_pos = [];
+   for j = 1:num_ev
+     r = ra_prop_get_value(prop_rri, ev(j), -1);
+     p = ra_prop_get_value(prop_qrs_pos, ev(j), -1);
+
+     rri = [rri r];
+     qrs_pos = [qrs_pos p];
+   endfor
+
+   %% To get all property values within one function call
+   %% use ra_prop_get_value_all():
+   % [ev_ids, chs, rri] = ra_prop_get_value_all(prop_rri);
+   % [ev_ids, chs, qrs_pos] =
ra_prop_get_value_all(prop_qrs_pos);
+   %
+   %% !!! This function returns also the event-id's and channel
+   %% !!! numbers the value belongs to. The order of returned
values
+   %% !!! is not guaranteed to be in chronological order.
+
+   whos rri
+   whos qrs_pos
+
+   figure();
+   x=(qrs_pos/samplerate) / 60;
+   plot(x, rri);
+ endfor
num_ev =  75
prop_rri =  186329584
prop_qrs_pos =  186289912

*** local user variables:

  Prot Name        Size                    Bytes  Class
  ==== ====        ====                    =====  =====
   rwd rri         1x75                      600  double

Total is 75 elements using 600 bytes


*** local user variables:

  Prot Name         Size                     Bytes  Class
  ==== ====         ====                     =====  =====
   rwd qrs_pos      1x75                       600  double

Total is 75 elements using 600 bytes
```

```
+
+ ra_meas_close(meas);
+ ra_lib_close(ra);
```

A.7. Use process-plugin

A.7.1. C Version

```
//
#include <stdio.h>
#include <string.h>
#include <ra.h>

int main(int argc, char *argv[])
{
    ra_handle ra;
    struct ra_info *inf;
    meas_handle meas;
    eval_handle eval;
    plugin_handle pl;
    struct proc_info *pi;

    /* initialize libRASCH */
    ra = ra_lib_init();
    if (ra == NULL)
    {
        printf("error initializing libRASCH\n");
        return -1;
    }

    /* open measurement */
    meas = ra_meas_open(ra, argv[1], NULL, 0);
    if (meas == NULL)
    {
        printf("can't open measurement %s\n", argv[1]);
        return -1;
    }

    /* get default evaluation */
    eval = ra_eval_get_default(meas);
    if (eval == NULL)
    {
        printf("no evaluation in measurement %s\n", argv[1]);
        return -1;
    }
```

```
/* get plugin-handle for hrv-plugin */
pl = ra_plugin_get_by_name(ra, "hrv", 0);
if (pl == NULL)
{
    printf("can't find plugin 'hrv'\n");
    return -1;
}

/* calculate hrv-values using the hrv-plugin */
pi = (struct proc_info *)ra_proc_get(meas, pl, NULL);
if (ra_proc_do(pi) == 0)
{
    long num_res_sets, num_results, m, l;
    value_handle vh;

    /* get number of result-sets */
    vh = ra_value_malloc();
    if (ra_info_get(pi, RA_INFO_PROC_NUM_RES_SETS_L, vh) !=
        0)
    {
        printf("no result-sets\n");
        return -1;
    }
    num_res_sets = ra_value_get_long(vh);

    /* get number of results */
    if (ra_info_get(pi, RA_INFO_PROC_NUM_RES_L, vh) != 0)
    {
        printf("no results\n");
        return -1;
    }
    num_results = ra_value_get_long(vh);

    for (m = 0; m < num_res_sets; m++)
    {
        printf("result-set #%d:\n", m + 1);

        for (l = 0; l < num_results; l++)
        {
            char out[200], t[100];

            /* set number of result in which we are interested */
            ra_value_set_number(vh, l);

            /* test if result is a default value (some
               non-default results are arrays which we skip in
               this example) */
            ra_info_get(pl, RA_INFO_PL_RES_DEFAULT_L, vh);
            if (ra_value_get_long(vh) == 0)
                continue;

            out[0] = '\0';
```

```
                if (ra_info_get(pl, RA_INFO_PL_RES_NAME_C, vh) ==
                    0)
                {
                    strcpy(t, ra_value_get_string(vh));
                    strcat(out, t);
                }
                if (ra_info_get(pl, RA_INFO_PL_RES_DESC_C, vh) ==
                    0)
                {
                    sprintf(t, " (%s)", ra_value_get_string(vh));
                    strcat(out, t);
                }
                if (ra_proc_get_result(pi, l, m, vh) == 0)
                {
                    sprintf(t, ": %lf", ra_value_get_double(vh));
                    strcat(out, t);
                }

                printf("  %s\n", out);
            }
        }

        ra_value_free(vh);
    }

    /* close */
    ra_proc_free(pi);
    ra_meas_close(meas);
    ra_lib_close(ra);

    return 0;
}                               /* main() */

//
```

```
result-set #1:
  SDNN (standard deviation of normal-to-normal intervals):
  30.238227
  HRVI (HRV-Index): 7.200000
  SDANN (standard deviation of averaged normal-to-normal
  intervals): nan
  rmssd (root mean of squared sucsessive differences): 32.494403
  pNN50 (): 7.142857
  TP (total power): 476.707920
  ULF (ultra low frequency power): 0.000000
  VLF (very low frequency power of short-term recordings):
  31.580847
  LF (low frequency power): 157.664788
  LF_NORM (normalised low frequency power): 35.420175
```

```
  HF (high freuqency power): 287.462286
  HF_NORM (normalised high frequency power): 64.579825
  LF_HF_RATIO (LF/HF ratio): 0.548471
  POWER_LAW (power law behavior): 0.000000
  SD1 (SD1 of the Poincare Plot): 15.845239
  SD2 (SD2 of the Poincare Plot): 17.873049
  DFA (overall DFA Alpha): nan
  DFA_OFFSET (offset of the overall DFA Alpha slope): nan
  DFA1 (DFA Alpha1): 0.242035
  DFA1_OFFSET (offset of the DFA Alpha-1 slope): 1.404195
  DFA2 (DFA Alpha2): nan
  DFA2_OFFSET (offset of the DFA Alpha-2 slope): nan
  DFA_USER (DFA Alpha of user-defined range)
  DFA_USER_OFFSET (offset of the user-defined DFA Alpha slope)
```

A.7.2. Perl Version

```
#
use strict;
use RASCH;

# initialize libRASCH
my $ra = new RASCH or die "error initializing libRASCH\n";

# open measurement and get default evaluation
my $meas = $ra->open_meas($ARGV[0], ”, 0) or
    die "can’t open measurement $ARGV[0]\n";

# get plugin-handle for hrv-plugin
my $pl = $ra->get_plugin_by_name(’hrv’) or
    die "can’t find plugin ’hrv’\n";

# calculate hrv-values using the hrv-plugin
my $proc = $pl->get_process($meas) or
    die "can’t initialize processing\n";
my $results = $proc->process();
# $results is a reference to an array; each array-element contains
# another array with three elements (value, name, description)
for (@$results)
{
    print $_->name() . ’ (’ . $_->desc() . ’) = ’ .
 $_->value() . "\n";
}

exit 0;
#
```

```
SDNN (standard deviation of normal-to-normal intervals) =
30.2382268563552
HRVI (HRV-Index) = 7.2
SDANN (standard deviation of averaged normal-to-normal
intervals) = nan
rmssd (root mean of squared sucsessive differences) =
32.4944032625042
pNN50 () = 7.14285714285714
TP (total power) = 476.707920259995
ULF (ultra low frequency power) = 0
VLF (very low frequency power of short-term recordings) =
31.5808466629042
LF (low frequency power) = 157.664787578416
LF_NORM (normalised low frequency power) = 35.4201748063357
HF (high freuqency power) = 287.462286018674
HF_NORM (normalised high frequency power) = 64.5798251936643
LF_HF_RATIO (LF/HF ratio) = 0.548471209082966
POWER_LAW (power law behavior) = 0
TACHO_INDEX (Event numbers used for HRV calculations) =
ARRAY(0x9f74b90)
USER_BAND (frequency power in a user-selected frequency band)
= 0
SD1 (SD1 of the Poincare Plot) = 15.8452385921394
SD2 (SD2 of the Poincare Plot) = 17.8730488314873
DFA (overall DFA Alpha) = nan
DFA_OFFSET (offset of the overall DFA Alpha slope) = nan
DFA1 (DFA Alpha1) = 0.242034900196407
DFA1_OFFSET (offset of the DFA Alpha-1 slope) = 1.40419467895075
DFA2 (DFA Alpha2) = nan
DFA2_OFFSET (offset of the DFA Alpha-2 slope) = nan
 () =
 () =
DFA_X (x-axis for DFA plot) = ARRAY(0x9fc82f8)
DFA_Y (y-axis for DFA plot) = ARRAY(0x9fc8438)
```

A.7.3. Python Version

```
#
import sys
from RASCH import *

# initialize libRASCH
ra = RASCH()
if not ra:
    print "can't initialize libRASCH"
    sys.exit()

# open measurement and get default evaluation
```

```
meas = ra.open_meas(sys.argv[1], ”, 0)
if not meas:
    print "can’t open measurement", sys.argv[1]
    sys.exit()

# get plugin-handle for hrv-plugin
pl = ra.get_plugin_by_name(’hrv’)
if not pl:
    print "can’t find plugin ’hrv’"
    sys.exit();

# calculate hrv-values using the hrv-plugin
proc = pl.get_process(meas)
if not proc:
    print "can’t initialize processing"
    sys.exit()

results = proc.process()
for item in results:
    if item.is_ok():
        print item.name(), "("+item.desc()+") =", item.value()
#
```

```
SDNN (standard deviation of normal-to-normal intervals) =
30.2382268564
HRVI (HRV-Index) = 7.2
SDANN (standard deviation of averaged normal-to-normal
intervals) = nan
rmssd (root mean of squared sucsessive differences) =
32.4944032625
pNN50 () = 7.14285714286
TP (total power) = 476.70792026
ULF (ultra low frequency power) = 0.0
VLF (very low frequency power of short-term recordings) =
31.5808466629
LF (low frequency power) = 157.664787578
LF_NORM (normalised low frequency power) = 35.4201748063
HF (high freuqency power) = 287.462286019
HF_NORM (normalised high frequency power) = 64.5798251937
LF_HF_RATIO (LF/HF ratio) = 0.548471209083
POWER_LAW (power law behavior) = 0.0
TACHO_INDEX (Event numbers used for HRV calculations) = [2.0,
3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 10.0, 11.0, 12.0, 13.0, 14.0,
15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0,
25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0,
35.0, 36.0, 37.0, 38.0, 39.0, 40.0, 41.0, 42.0, 43.0, 44.0,
45.0, 46.0, 47.0, 48.0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0,
55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 61.0, 62.0, 63.0, 64.0,
65.0, 66.0, 67.0, 68.0, 69.0, 70.0, 71.0, 72.0, 73.0, 74.0]
```

```
USER_BAND (frequency power in a user-selected frequency band)
= 0.0
SD1 (SD1 of the Poincare Plot) = 15.8452385921
SD2 (SD2 of the Poincare Plot) = 17.8730488315
DFA (overall DFA Alpha) = nan
DFA_OFFSET (offset of the overall DFA Alpha slope) = nan
DFA1 (DFA Alpha1) = 0.242034900196
DFA1_OFFSET (offset of the DFA Alpha-1 slope) = 1.40419467895
DFA2 (DFA Alpha2) = nan
DFA2_OFFSET (offset of the DFA Alpha-2 slope) = nan
DFA_X (x-axis for DFA plot) = [0.6020599913279624,
0.69897000433601886, 0.77815125038364363, 0.84509804001425681,
0.90308998699194354, 0.95424250943932487, 1.0,
1.0413926851582251, 1.0791812460476249, 1.1139433523068367,
1.146128035678238, 1.1760912590556813, 1.2041199826559248,
1.2304489213782739]
DFA_Y (y-axis for DFA plot) = [1.4964657214676538,
1.5932815728362522, 1.6208138827475429, 1.6561856924575491,
1.6469306466489699, 1.5934526766952628, 1.6510314905644941,
1.6268009535558938, 1.6141161531657984, 1.6277297067193524,
nan, nan, nan, nan]
```

A.7.4. Matlab/Octave Version

```
GNU Octave, version 3.0.5
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "i486-pc-linux-gnu".

Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit
http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).

For information about changes from previous versions, type
'news'.

ra =  163726552
```

```
meas =  164266056
eva =  164682208
pl =  164481848
proc =  165385144
ans = 0
num =  28
1: SDNN (standard deviation of normal-to-normal intervals)
= 30.238227
2: HRVI (HRV-Index) = 7.200000
3: SDANN (standard deviation of averaged normal-to-normal
intervals) = NaN
4: rmssd (root mean of squared sucsessive differences)
= 32.494403
5: pNN50 () = 7.142857
6: TP (total power) = 476.707920
7: ULF (ultra low frequency power) = 0.000000
8: VLF (very low frequency power of short-term recordings)
= 31.580847
9: LF (low frequency power) = 157.664788
10: LF_NORM (normalised low frequency power) = 35.420175
11: HF (high freuqency power) = 287.462286
12: HF_NORM (normalised high frequency power) = 64.579825
13: LF_HF_RATIO (LF/HF ratio) = 0.548471
14: POWER_LAW (power law behavior) = 0.000000
15: TACHO_INDEX (Event numbers used for HRV calculations) =
[not a scalar value]
16: USER_BAND (frequency power in a user-selected frequency
band) = 0.000000
17: SD1 (SD1 of the Poincare Plot) = 15.845239
18: SD2 (SD2 of the Poincare Plot) = 17.873049
19: DFA (overall DFA Alpha) = NaN
20: DFA_OFFSET (offset of the overall DFA Alpha slope) = NaN
21: DFA1 (DFA Alpha1) = 0.242035
22: DFA1_OFFSET (offset of the DFA Alpha-1 slope) = 1.404195
23: DFA2 (DFA Alpha2) = NaN
24: DFA2_OFFSET (offset of the DFA Alpha-2 slope) = NaN
25: DFA_USER (DFA Alpha of user-defined range) = [not a
scalar value]
```