

User Manual for libRASCH-0.7.4

Raphael Schneider

User Manual for libRASCH-0.7.4

by Raphael Schneider

libRASCH User Manual Version 0.1 Edition

Published 2004

Copyright © 2004 Raphael Schneider

Table of Contents

Forward	vi
1. Introduction.....	1
1.1. Overview of the next chapters.....	1
2. Concepts and Terminology	2
2.1. General Structure of libRASCH	2
2.2. Terminology	2
3. A Tutorial.....	5
3.1. Initialize libRASCH and Basic Usage	5
3.1.1. C Version	5
3.1.2. Perl Version	7
3.1.3. Python Version.....	8
3.1.4. Matlab/Octave Version	9
3.2. Access Measurements	10
3.3. Access Recordings	12
3.4. Get Sample Data	14
3.5. Access Evaluations.....	16
3.6. Get Events	19
3.7. Usage of process-plugins	20
A. Examples for all supported lanuages/systems	24
A.1. Init libRASCH.....	24
A.1.1. C Version	24
A.1.2. Perl Version.....	25
A.1.3. Python Version.....	26
A.1.4. Matlab/Octave Version	27
A.2. Open measurement.....	28
A.2.1. C Version	28
A.2.2. Perl Version.....	30
A.2.3. Python Version.....	31
A.2.4. Matlab/Octave Version	32
A.3. Handle recordings	32
A.3.1. C Version	33
A.3.2. Perl Version.....	34
A.3.3. Python Version.....	36
A.3.4. Matlab/Octave Version	37
A.4. Access raw data	38
A.4.1. C Version	38
A.4.2. Perl Version.....	39
A.4.3. Python Version.....	40
A.4.4. Matlab/Octave Version	41
A.5. Access evaluation	41
A.5.1. C Version	42
A.5.2. Perl Version.....	44
A.5.3. Python Version.....	45
A.5.4. Matlab/Octave Version	46
A.6. Access events	49

A.6.1. C Version	49
A.6.2. Perl Version.....	50
A.6.3. Python Version.....	51
A.6.4. Matlab/Octave Version	52
A.7. Use process-plugin	53
A.7.1. C Version	53
A.7.2. Perl Version.....	56
A.7.3. Python Version.....	57
A.7.4. Matlab/Octave Version	58

List of Figures

2-1. Structure of libRASCH.	2
3-1. Screenshot after performing the commands in Octave as shown in the Octave session above (and on the left side in 15	
3-2. Screenshot after performing the commands in Octave as shown in the Octave session above (and on the left side in 20	

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

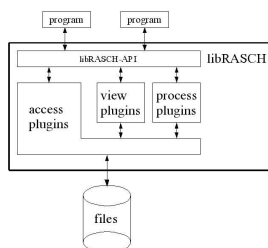
Chapter 2. Concepts and Terminology

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-leads, the measurement consists of one top recording with two sub-recordings. The first sub-recording contains 4 channels (3 eeg 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 'occurrence of a qrs-complex') and/or zero or more continuous events (like 'time interval with noise').

Event

An event describes the occurrence of something in a recording (e.g. a heartbeat in an eeg). 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 exists 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 measurements¹. Most of the times, the direct usage of access-plugins with the libRASCH-API is not needed.

Process-Plugin

Process plugins perform a specific task on the measurement (e.g. the HRT-plugin calculates the Heart-Rate Turbulence parameters for an eeg 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 appropriate 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-property 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.

- ③ `ra_lib_get_error()` returns the last error occurred in libRASCH. To check if the initialization was successful, check that no error occurred. If an error occurred, a short description of the error can be retrieved with the function `ra_lib_get_error()`.
 - ④ `ra_value_malloc()` returns a 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_*`.
 - ⑤ `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.)
 - ⑥ `ra_value_free()` frees the memory associated with the value object allocated above.
 - ⑦ `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`).
 - ⑧ `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.
 - ⑨ `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 (): 30
measurements found in ./database:
 1: ./database/JesusOlivian2003-12-EMG2.edf
 2: ./database/100s.hef
```

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";
}

# find all measurements in a directory
my $meas = $ra->find_meas($ARGV[0]); ❹
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 array including (1) the value, (2) the name and (3) a short description of the wanted information.
- ❹ `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, name, desc] = ra.get_info(info='num_plugins') ❸
if (value):
    print name, "("+desc+":)", value

# find all measurements in a directory
meas = ra.get_all_meas(sys.argv[1]) ❹
print "measurements found in " + sys.argv[1] + ":\n"
cnt = 1
for item in meas:
    print "%d: %s" % (cnt, item)
    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 an array including (1) the value, (2) the name and (3) a short description of the wanted information.
- ❹ `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.

```

rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init ❶
ra = 145016336
octave:2> [err_num, err_text]=ra_lib_get_error(ra)

```

```

err_num = 1
err_text = libRASCH: no error
octave:3> [value, name, desc]=ra_lib_get_info(ra, 'num_plugins') ❷
value = 25
name = #plugins
desc =
octave:4> meas=ra_meas_find(ra, './database') ❸
meas =
{
  [1,1] = ./database/JesusOlivan2003-12-EMG2.edf
  [2,1] = ./database/100s.heaf
}
octave:5>

```

- ❶ Initialize libRASCH. If function call is successful, 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], 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 samplerate 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 -- recorded at $rec_date" .
    " -- #devices=$num_dev #channels=$num_ch\n";

# print name for every device ❷
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;
#

```

- ❶ The function `get_first_session_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 this is done by setting the argument `'dev'` to the device number, in C you do this in setting the variable `'dev'` in the `ra_info` struct.
- ❸ When asking for information about a channel you need to set the channel number. In Perl this is done by setting the argument `'ch'` to the channel number, in C you do this in setting the variable `'ch'` in the `ra_info` struct.

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.

```

rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init
ra = 145017272
octave:2> meas=ra_meas_open(ra, '/home/rasch/ecgs/AT011030_rdi.ART', 0)
meas = 145244336
octave:3> rec=ra_rec_get_first(meas, 0)
rec = 145421120
octave:4> num_ch=ra_rec_get_info(rec, 'rec_num_channel')
num_ch = 5
octave:5> ch_all=[];
octave:6> for i=0:(num_ch-1)
> ch=ra_raw_get(rec, i, 0, 10000); ❶
> ch_all=[ch_all ch'];
> endfor
octave:7> whos ch_all

*** local user variables:

prot  type                rows  cols  name
====  ====                ====  ====  ====
rwd  matrix                10000   5  ch_all

octave:8> samplerate=ra_ch_get_info(rec, 0, 'ch_samplerate')
samplerate = 1600
octave:9> x=0:9999;
octave:10> x = x / samplerate;
octave:11> figure(0); plot(x, ch_all(:,1));
octave:12> figure(1); plot(x, ch_all(:,2));
octave:13> figure(2); plot(x, ch_all(:,3));
octave:14> figure(3); plot(x, ch_all(:,4));
octave:15> figure(4); plot(x, ch_all(:,5));
octave:16>

```

- 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.


```

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

int main(int argc, char *argv[])
{
    ra_handle ra;
    value_handle vh;
    meas_handle meas;
    eval_handle eval;
    long l, num;

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

    /* get default evaluation */
    eval = ra_eval_get_def(meas); ❶
    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-properties */
    num = 0;
    if (ra_info_get(eval, RA_INFO_EVAL_PROP_NUM_L, vh) == 0)
        num = ra_value_get_long(vh);
    for (l = 0; l < num; l++) ❷
    {
        prop_handle evprop;

```

```

evset_handle evset;

evprop = ra_prop_get_by_num(eval, 1);
evset = ra_evset_get_by_prop(evprop);

if ((evprop == NULL) || (evset == NULL))
    continue;

if (ra_info_get(evprop, RA_INFO_EVPROP_NAME_C, vh) == 0)
    printf("event-property %s ",
           ra_value_get_string(vh));
if (ra_info_get(evprop, RA_INFO_EVPROP_NAME_C, vh) == 0)
    printf("(%s) ", ra_value_get_string(vh));
if (ra_info_get(evset, RA_INFO_EVSET_NAME_C, vh) == 0)
    printf("belongs to event-set %s ",
           ra_value_get_string(vh));
if (ra_info_get(evset, RA_INFO_EVSET_EV_NUM_L, vh) == 0)
    printf("and contains %d events",
           ra_value_get_long(vh));
printf("\n");
}

/* 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 original was added at 27.05.2005 09:28:53 using
the program handle_eval

event-property qrs-pos (qrs-pos) belongs to event-set heartbeat
and contains 74 events
event-property qrs-class (qrs-class) belongs to event-set
heartbeat and contains 74 events
event-property qrs-temporal (qrs-temporal) belongs to event-set
heartbeat and contains 74 events
event-property rri (rri) belongs to event-set heartbeat and
contains 74 events
event-property rri-class (rri-class) belongs to event-set

```

```

heartbeat and contains 74 events
event-property rri-refvalue (rri-refvalue) belongs to event-set
heartbeat and contains 74 events
event-property rri-num-refvalue (rri-num-refvalue) belongs to
event-set heartbeat and contains 74 events

```

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 position of the QRS-complexes
- plot the tachogram of the measurement

```

rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init
ra = 145018424
octave:2> meas=ra_meas_open(ra, '/home/rasch/ecgs/AT011030_rdi.ART', 0)
meas = 145245184
octave:3> eva=ra_eval_get_def(meas)
eva = 145612904
octave:4> prop_rri=ra_prop_get_by_name(eva, 'rri')
prop_rri = 145667632
octave:5> rri=ra_ev_get_values(prop_rri, 0, -1); ❶
octave:6> whos rri

*** local user variables:

prot  type                rows  cols  name
====  ====                ====  ====  ====
rwd   matrix                1    1807  rri

octave:7> prop_qrs=ra_prop_get_by_name(eva, 'qrs-pos')
prop_qrs = 145613912
octave:8> qrs_pos=ra_ev_get_values(prop_qrs, 0, -1);
octave:9> whos qrs_pos

*** local user variables:

prot  type                rows  cols  name
====  ====                ====  ====  ====
rwd   matrix                1    1807  qrs_pos

octave:10> samplerate=ra_meas_get_info(meas, 'max_samplerate')
samplerate = 1600

```

```

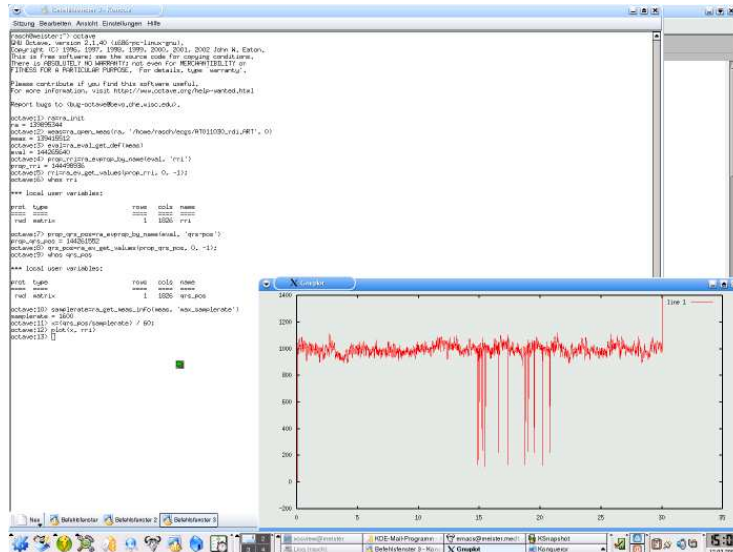
octave:11> x=(qrs_pos/samplerate) / 60;
octave:12> plot(x, rri)
octave:13>

```

①

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

    /* get default evaluation */
    eval = ra_eval_get_def(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); ❶
    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(pl); ❷
    pi->mh = meas;
    pi->rh = ra_rec_get_first(meas, 0);
    pi->eh = eval;
    if (ra_proc_do(pi) == 0) ❸
    {
        long num_results, l;
        value_handle vh;

        /* get number of results */
        vh = ra_value_malloc();
        if (ra_info_get(pl, RA_INFO_PL_NUM_RESULTS_L, vh) != 0)

```

```

    {
        printf("no results\n");
        return -1;
    }
    num_results = ra_value_get_long(vh);

    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); ❹

        /* 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, vh) == 0) ❺
        {
            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 successful, the function returns 0.
- ❹ To select the result, in which you are interested, 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:

```
SDNN (standard deviation of normal-to-normal intervals):
30.876196
HRVI (HRV-Index): 4.800000
SDANN (standard deviation of averaged normal-to-normal
intervals): nan
rmssd (root mean of squared successive differences): 33.839537
pNN50 (): 7.142857
TP (total power): 482.603096
ULF (ultra low frequency power): 0.000000
VLF (very low frequency power of short-term recordings):
32.889150
LF (low frequency power): 157.213726
LF_NORM (normalised low frequency power): 34.958606
HF (high frequency power): 292.500220
HF_NORM (normalised high frequency power): 65.041394
LF_HF_RATIO (LF/HF ratio): 0.537482
POWER_LAW (power law behavior): 0.000000
```

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 languages/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 (): 30
measurements found in ./database:
1: ./database/JesusOlivan2003-12-EMG2.edf
2: ./database/100s.heg
```

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 (): 30
measurements found in ./database:
1: ./database/JesusOlivan2003-12-EMG2.edf
2: ./database/100s.heg
```

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, name, desc] = ra.get_info(info='num_plugins')
if (value):
    print name, "("+desc+"):", value

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

#

#plugins (): 30
measurements found in ./database:

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

A.1.4. Matlab/Octave Version

```
rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init
ra = 145016336
octave:2> [err_num, err_text]=ra_lib_get_error(ra)
err_num = 1
err_text = libRASCH: no error
octave:3> [value, name, desc]=ra_lib_get_info(ra, 'num_plugins')
value = 25
name = #plugins
```

```
desc =
octave:4> meas=ra_meas_find(ra, './database')
meas =
{
  [1,1] = ./database/JesusOlivan2003-12-EMG2.edf
  [2,1] = ./database/100s.hef
}
octave:5>
```

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], 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 samplerate 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 samprate 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, name, desc] = meas.get_info(info='num_sessions')
if value:
    print name, ("+"desc+"): ", value
[value, name, desc] = meas.get_info(info='max_samplerate')
if value:
    print name, ("+"desc+"): ", value

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

#

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

A.2.4. Matlab/Octave Version

```
rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init
ra = 145016232
octave:2> meas=ra_meas_open(ra, '/home/rasch/ecgs/AT011030_rdi.ART', 0)
meas = 145244032
octave:3> [v, n, d]=ra_meas_get_info(meas, 'num_sessions')
v = 1
n = #sessions
d =
octave:4> [v, n, d]=ra_meas_get_info(meas, 'max_samplerate')
v = 1600
n = max. samplerate
d = maximum samplerate used in measurement
octave:5> num=ra_meas_get_info(meas, 'num_obj_infos')
num = 7
octave:6> for i=0:(num-1)
> [v,n,d]=ra_info_get_by_idx(meas, 'meas', i)
> endfor
v = A.
n = Name
d = name of the person measured
v = T.
n = Forename
d = forename of the person measured
v = 16.01.1955
n = Birthday
d = birthday of the person measured
v = male
n = gender
d = gender of the person measured
v = 46
n = Age
d = age of the person measured
v = 196
n = Height
d = height of the person measured in cm
v = 99
n = Weight
d = weight of the person measured in kg
octave:7>
```

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], 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 -- ", ra_value_get_string(vh));
    if (ra_info_get(rec, RA_INFO_REC_GEN_DATE_C, vh) == 0)
```

```

    printf("recorded at %s -- ", ra_value_get_string(vh));
printf("#devices=%d #channels=%d\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 -- recorded at $rec_date" .
    " -- #devices=$num_dev #channels=$num_ch\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_session_rec(0)
if not rec:
    print "can't get root recording"
    sys.exit()

# get some infos about recording
[num_dev, n, d] = rec.get_info(info='rec_num_devices')
[num_ch, n, d] = rec.get_info(info='rec_num_channel')
[rec_name, n, d] = rec.get_info(info='rec_name')
[rec_date, n, d] = rec.get_info(info='rec_date')
print "measurement", rec_name, " -- recorded at", rec_date, \
      " -- #devices =", num_dev, " #channels =", num_ch
```

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

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

```
rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init
ra = 145016952
octave:2> meas=ra_meas_open(ra, '/home/rasch/ecgs/AT011030_rdi.ART', 0)
meas = 145243840
octave:3> rec=ra_rec_get_first(meas, 0)
rec = 145420840
octave:4> num_ch=ra_rec_get_info(rec, 'rec_num_channel')
num_ch = 5
octave:5> rec_name=ra_rec_get_info(rec, 'rec_name')
rec_name = AT011030_rdi
```

```
octave:6> rec_date=ra_rec_get_info(rec, 'rec_date')
rec_date = 30.10.2001
octave:7> for i=0:(num_ch-1)
> name=ra_ch_get_info(rec, i, 'ch_name')
> unit=ra_ch_get_info(rec, i, 'ch_unit')
> endfor
name = EKGX
unit = uV
name = EKGY
unit = uV
name = EKGZ
unit = uV
name = ABP
unit = mmHg
name = RESP
unit = uV
octave:8>
```

A.4. Access raw data

A.4.1. C Version

```
//
#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], 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

```

rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init
ra = 145017272
octave:2> meas=ra_meas_open(ra, '/home/rasch/ecgs/AT011030_rdi.ART', 0)
meas = 145244336
octave:3> rec=ra_rec_get_first(meas, 0)
rec = 145421120
octave:4> num_ch=ra_rec_get_info(rec, 'rec_num_channel')
num_ch = 5
octave:5> ch_all=[];
octave:6> for i=0:(num_ch-1)
> ch=ra_raw_get(rec, i, 0, 10000);
> ch_all=[ch_all ch'];
> endfor
octave:7> whos ch_all

*** local user variables:

prot  type                rows  cols  name
====  ====                =====
rwd   matrix                10000  5    ch_all

octave:8> samplerate=ra_ch_get_info(rec, 0, 'ch_samplerate')
samplerate = 1600
octave:9> x=0:9999;
octave:10> x = x / samplerate;
octave:11> figure(0); plot(x, ch_all(:,1));
octave:12> figure(1); plot(x, ch_all(:,2));
octave:13> figure(2); plot(x, ch_all(:,3));
octave:14> figure(3); plot(x, ch_all(:,4));
octave:15> figure(4); plot(x, ch_all(:,5));
octave:16>

```

A.5. Access evaluation

A.5.1. C Version

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

int main(int argc, char *argv[])
{
    ra_handle ra;
    value_handle vh;
    meas_handle meas;
    eval_handle eval;
    long l, num;

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

    /* get default evaluation */
    eval = ra_eval_get_def(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-properties */
num = 0;
if (ra_info_get(eval, RA_INFO_EVAL_PROP_NUM_L, vh) == 0)
    num = ra_value_get_long(vh);
for (l = 0; l < num; l++)
{
    prop_handle evprop;
    evset_handle evset;

    evprop = ra_prop_get_by_num(eval, l);
    evset = ra_evset_get_by_prop(evprop);

    if ((evprop == NULL) || (evset == NULL))
        continue;

    if (ra_info_get(evprop, RA_INFO_EVPROP_NAME_C, vh) == 0)
        printf("event-property %s ",
            ra_value_get_string(vh));
    if (ra_info_get(evprop, RA_INFO_EVPROP_NAME_C, vh) == 0)
        printf("(%s) ", ra_value_get_string(vh));
    if (ra_info_get(evset, RA_INFO_EVSET_NAME_C, vh) == 0)
        printf("belongs to event-set %s ",
            ra_value_get_string(vh));
    if (ra_info_get(evset, RA_INFO_EVSET_EV_NUM_L, vh) == 0)
        printf("and contains %d events",
            ra_value_get_long(vh));
    printf("\n");
}

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

return 0;
}                                     /* main() */

//

```

evaluation original was added at 27.05.2005 09:28:53 using
the program handle_eval

```

event-property qrs-pos (qrs-pos) belongs to event-set heartbeat
and contains 74 events
event-property qrs-class (qrs-class) belongs to event-set
heartbeat and contains 74 events
event-property qrs-temporal (qrs-temporal) belongs to event-set
heartbeat and contains 74 events
event-property rri (rri) belongs to event-set heartbeat and

```

```
contains 74 events
event-property rri-class (rri-class) belongs to event-set
heartbeat and contains 74 events
event-property rri-refvalue (rri-refvalue) belongs to event-set
heartbeat and contains 74 events
event-property rri-num-refvalue (rri-num-refvalue) belongs to
event-set heartbeat and contains 74 events
```

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";

# !!!!! eval handling will be changed in the next version !!!!!

# list event-properties
#my $evprops = $eval->get_props_all();
#for (@$evprops)
#{
#    my $prop = $eval->get_prop_by_name($_);
#    my ($name) = $prop->get_info(info => 'eval_prop_name');
#    my ($desc) = $prop->get_info(info => 'eval_prop_desc');
#
#    my $evset = $prop->get_evset();
#    my ($evset_name) = $evset->get_info(info => 'eval_set_name');
#    my ($n_ev) = $evset->get_info(info => 'eval_set_num_events');
```

```
#
#   print "event-property $name ($desc) belongs to event-set" .
# " $evset_name and contains $n_ev events\n";
#}

exit 0;
#
```

evaluation original was added at 27.05.2005 09:28:53 using
the program perl

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_def_eval()
if not eva:
    print "no evaluation in measurement"
    sys.exit()

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

# list event-properties
```

```
evprops = eva.get_evprops()
for elem in evprops:
    prop = eva.get_evprop_by_name(elem)
    [name, n, d] = prop.get_info(info='eval_prop_name')
    [desc, n, d] = prop.get_info(info='eval_prop_desc')

    evset = prop.get_evset()
    [evset_name, n, d] = evset.get_info(info='eval_set_name')
    [num_events, n, d] = evset.get_info(info='eval_set_num_events')

    print "event-property", name, "("+desc+") belongs to event-set", \
          evset_name, "and contains", num_events, "events";

#
```

evaluation original was added at 27.05.2005 09:28:53 using
the program python

```
event-property qrs-pos (position of fiducial point of
QRS-complex in sampleunits) belongs to event-set heartbeat
and contains 74 events
event-property qrs-class (classification of QRS complex)
belongs to event-set heartbeat and contains 74 events
event-property qrs-temporal (temporal setting of beat) belongs
to event-set heartbeat and contains 74 events
event-property rri (RR interval) belongs to event-set heartbeat
and contains 74 events
event-property rri-class (classification of RR interval)
belongs to event-set heartbeat and contains 74 events
event-property rri-refvalue (reference rri representing the
current heart-rate) belongs to event-set heartbeat and contains
74 events
event-property rri-num-refvalue (number of rri's used for
calculation of reference value) belongs to event-set heartbeat
and contains 74 events
```

A.5.4. Matlab/Octave Version

```
rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init
ra = 145016856
```

```
octave:2> meas=ra_meas_open(ra, '/home/rasch/ecgs/AT011030_rdi.ART', 0)
meas = 145245344
octave:3> eva=ra_eval_get_def(meas)
eva = 145613352
octave:4> ra_eval_get_info(eva, 'eval_name')
ans = detect
octave:5> ra_eval_get_info(eva, 'eval_add_timestamp')
ans = 02.02.2004 15:14:06
octave:6> ra_eval_get_info(eva, 'eval_program')
ans = raschlab_gt
octave:7> props=ra_prop_get_all(eva);
octave:8> num=length(props)
num = 21
octave:9> for i=1:num
> prop=ra_prop_get_by_name(eva, props{i});
> evset=ra_evset_get_by_prop(prop);
> name=ra_prop_get_info(prop, 'eval_prop_name')
> desc=ra_prop_get_info(prop, 'eval_prop_desc')
> set_name=ra_evset_get_info(evset, 'eval_set_name')
> num_events=ra_evset_get_info(evset, 'eval_set_num_events')
> endfor
name = qrs-pos
desc = position of fiducial point of QRS-complex in sampleunits
set_name = heartbeat
num_events = 1807
name = qrs-ch
desc = bitmask of channels where qrs was detected
set_name = heartbeat
num_events = 1807
name = qrs-class
desc = classification of QRS complex
set_name = heartbeat
num_events = 1807
name = qrs-template
desc = QRS template number
set_name = heartbeat
num_events = 1807
name = qrs-template-corr
desc = correlation of qrs-complex with template
set_name = heartbeat
num_events = 1807
name = heartbeat-flags
desc = Flags for heartbeat events
set_name = heartbeat
num_events = 1807
name = qrs-temporal
desc = temporal setting of beat
set_name = heartbeat
num_events = 1807
name = rri
desc = RR interval
set_name = heartbeat
num_events = 1807
```

```
name = rri-class
desc = classification of RR interval
set_name = heartbeat
num_events = 1807
name = rri-refvalue
desc = reference rri representing the current heart-rate
set_name = heartbeat
num_events = 1807
name = rri-num-refvalue
desc = number of rri's used for calculation of reference value
set_name = heartbeat
num_events = 1807
name = rr-systolic
desc = systolic bloodpressure in mmHg
set_name = heartbeat
num_events = 1807
name = rr-diastolic
desc = diastolic bloodpressure in mmHg
set_name = heartbeat
num_events = 1807
name = rr-mean
desc = mean bloodpressure in mmHg
set_name = heartbeat
num_events = 1807
name = rr-systolic-pos
desc = position of systolic bloodpressure in sampleunits
set_name = heartbeat
num_events = 1807
name = rr-diastolic-pos
desc = position of diastolic bloodpressure in sampleunits
set_name = heartbeat
num_events = 1807
name = rr-flags
desc = flags for the rr-values
set_name = heartbeat
num_events = 1807
name = ibi
desc = inter-beat-interval
set_name = heartbeat
num_events = 1807
name = resp-chest-mean-rri
desc = mean chest measurement of the rr interval
set_name = heartbeat
num_events = 1807
name = resp-chest-mean-ibi
desc = mean chest measurement of the inter beat interval
set_name = heartbeat
num_events = 1807
name = rr-calibration-seq
desc = start- and end-value of event is start- and end-position
      of calibration sequence
set_name = rr-calibration
num_events = 26
```

```
octave:10>
```

A.6. Access events

A.6.1. C Version

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

int main(int argc, char *argv[])
{
    ra_handle ra;
    meas_handle meas;
    eval_handle eval;
    prop_handle prop_rri, prop_qrs_pos;
    evset_handle evset;
    long l, num_events;
    double *rri = NULL;
    double *qrs_pos = 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], 0);
    if (meas == NULL)
    {
        printf("can't open measurement %s\n", argv[1]);
        return -1;
    }

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

```

/* get event-properties for RR-intervals and position of
   QRS-complex */
prop_rri = ra_prop_get_by_name(eval, "rri");
if (prop_rri == NULL)
{
    printf("no event-property 'rri' in evaluation\n");
    return -1;
}
prop_qrs_pos = ra_prop_get_by_name(eval, "qrs-pos");
if (prop_qrs_pos == NULL)
{
    printf("no event-property 'qrs-pos' in evaluation\n");
    return -1;
}

/* get values for all RR-intervals and QRS-complexes */
evset = ra_evset_get_by_prop(prop_rri);
/* number of events are the same for rri and qrs-pos, because
   both belongs to the same event-set ('heartbeat') */
num_events = ra_evset_get_num_events(evset);
rri = malloc(sizeof(double) * num_events);
qrs_pos = malloc(sizeof(double) * num_events);
for (l = 0; l < num_events; l++)
{
    rri[l] = ra_ev_get_value(prop_rri, l);
    qrs_pos[l] = ra_ev_get_value(prop_qrs_pos, l);
}

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

/* clean up */
if (rri)
    free(rri);
if (qrs_pos)
    free(rri);

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_def_eval() or
    die "no evaluation in the measurement\n";

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

```
rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init
ra = 145018424
octave:2> meas=ra_meas_open(ra, '/home/rasch/ecgs/AT011030_rdi.ART', 0)
meas = 145245184
octave:3> eva=ra_eval_get_def(meas)
eva = 145612904
octave:4> prop_rri=ra_prop_get_by_name(eva, 'rri')
prop_rri = 145667632
octave:5> rri=ra_ev_get_values(prop_rri, 0, -1);
octave:6> whos rri

*** local user variables:
```

```

prot  type                rows  cols  name
====  ====                ====  ====  ====
rwd  matrix                1    1807  rri

octave:7> prop_qrs=ra_prop_get_by_name(eva, 'qrs-pos')
prop_qrs = 145613912
octave:8> qrs_pos=ra_ev_get_values(prop_qrs, 0, -1);
octave:9> whos qrs_pos

*** local user variables:

prot  type                rows  cols  name
====  ====                ====  ====  ====
rwd  matrix                1    1807  qrs_pos

octave:10> samplerate=ra_meas_get_info(meas, 'max_samplerate')
samplerate = 1600
octave:11> x=(qrs_pos/samplerate) / 60;
octave:12> plot(x, rri)
octave:13>

```

A.7. Use process-plugin

A.7.1. C Version

```

//
#include <stdio.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], 0);
if (meas == NULL)
{
    printf("can't open measurement %s\n", argv[1]);
    return -1;
}

/* get default evaluation */
eval = ra_eval_get_def(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(pl);
pi->mh = meas;
pi->rh = ra_rec_get_first(meas, 0);
pi->eh = eval;
if (ra_proc_do(pi) == 0)
{
    long num_results, l;
    value_handle vh;

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

    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, 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() */

//

```

```

SDNN (standard deviation of normal-to-normal intervals):
30.876196
HRVI (HRV-Index): 4.800000
SDANN (standard deviation of averaged normal-to-normal
intervals): nan
rmssd (root mean of squared successive differences): 33.839537
pNN50 (): 7.142857
TP (total power): 482.603096
ULF (ultra low frequency power): 0.000000
VLF (very low frequency power of short-term recordings):
32.889150

```

```
LF (low frequency power): 157.213726
LF_NORM (normalised low frequency power): 34.958606
HF (high frequency power): 292.500220
HF_NORM (normalised high frequency power): 65.041394
LF_HF_RATIO (LF/HF ratio): 0.537482
POWER_LAW (power law behavior): 0.000000
```

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";
my $eval = $meas->get_default_eval() or
    die "no evaluation in the measurement\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->init_process(eva => $eval) 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) =
30.8761964330589
HRVI (HRV-Index) = 4.8
```

```
SDANN (standard deviation of averaged normal-to-normal
intervals) = nan
rmsd (root mean of squared successive differences) =
33.8395373153104
pNN50 () = 7.14285714285714
TP (total power) = 482.603095556721
ULF (ultra low frequency power) = 0
VLF (very low frequency power of short-term recordings) =
32.8891498386073
LF (low frequency power) = 157.21372565195
LF_NORM (normalised low frequency power) = 34.9586058312928
HF (high frequency power) = 292.500220066165
HF_NORM (normalised high frequency power) = 65.0413941687072
LF_HF_RATIO (LF/HF ratio) = 0.537482418359847
POWER_LAW (power law behavior) = 0
TACHO_INDEX (Event numbers used for HRV calculations) =
ARRAY(0x815add8)
USER_BAND (frequency power in a user-selected frequency band) =
```

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()
eva = meas.get_def_eval()
if not eva:
    print "no evaluation in measurement"
    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.init_process(eva=eva)
if not proc:
    print "can't initialize processing"
    sys.exit()

results = proc.process()
for item in results:
    print item[1], "("+item[2]+") =", item[0]

#

SDNN (standard deviation of normal-to-normal intervals) =
30.8761964331
HRVI (HRV-Index) = 4.8
SDANN (standard deviation of averaged normal-to-normal
intervals) = nan
rmssd (root mean of squared successive differences) =
33.8395373153
pNN50 ( ) = 7.14285714286
TP (total power) = 482.603095557
ULF (ultra low frequency power) = 0.0
VLF (very low frequency power of short-term recordings) =
32.8891498386
LF (low frequency power) = 157.213725652
LF_NORM (normalised low frequency power) = 34.9586058313
HF (high frequency power) = 292.500220066
HF_NORM (normalised high frequency power) = 65.0413941687
LF_HF_RATIO (LF/HF ratio) = 0.53748241836
POWER_LAW (power law behavior) = 0.0
TACHO_INDEX (Event numbers used for HRV calculations) = None
USER_BAND (frequency power in a user-selected frequency band)
= None
```

A.7.4. Matlab/Octave Version

```
rasch@rasch:~> octave
GNU Octave, version 2.1.50 (i686-pc-linux-gnu).
...

octave:1> ra=ra_lib_init
ra = 145019552
octave:2> meas=ra_meas_open(ra, '/home/rasch/ecgs/AT011030_rdi.ART', 0)
meas = 145245776
octave:3> eva=ra_eval_get_def(meas)
```

```
eva = 145613504
octave:4> pl=ra_plugin_get_by_name(ra, 'hrv', 0)
pl = 145340240
octave:5> proc=ra_proc_init(ra, meas, 0, eva, pl, 0, 0)
proc = 146187632
octave:6> ra_proc_do(proc)
ans = 0
octave:7> [v, n, d]=ra_result_get(proc);
octave:8> num=length(v)
num = 16
octave:9> for i=1:num
> printf("%s (%s) = %f\n", n(i,:), d(i,:), v{i});
> endfor
SDNN (standard deviation of normal-to-normal intervals) = 39.136515
HRVI (HRV-Index) = 10.482558
SDANN (standard deviation of averaged normal-to-normal intervals)
= 11.164071
rmssd (root mean of squared successive differences) = 27.114662
pNN50 ( ) = 6.163243
TP (total power) = 1347.525433
ULF (ultra low frequency power) = 156.680947
VLF (very low frequency power of short-term recordings) = 491.156903
LF (low frequency power) = 513.812519
LF_NORM (normalised low frequency power) = 73.434563
HF (high frequency power) = 185.875064
HF_NORM (normalised high frequency power) = 26.565437
LF_HF_RATIO (LF/HF ratio) = 2.764290
POWER_LAW (power law behavior) = -0.657030
TACHO_INDEX (Event numbers used for HRV calculations) = 1.000000
octave:9>
```