Waveform Extraction Library

RIEGL
LASER MEASUREMENT SYSTEMS
www.riegl.com

Waveform Extraction Library

Printed: 2019-10-29

Table of Contents

Part 1 Introduction

1 Sampled and Digitized Waveform

During data acquisition the echo signals detected by the receiver frontend are digitized. The digitized echo signals are analyzed in real-time and whenever patterns of interest are identified, groups of samples (i.e. sample blocks) are stored for subsequent full waveform analysis. All sample blocks recorded for one laser shot, together with additional information about the laser shot, e.g., the beam direction at the laser pulse emission, is referred to as waveform data for this laser shot.

1 shows an example of such a waveform. The *start of range gate* , t_{sorg} , is the time instant when data sampling is initiated. This timestamp represents the primary key to access waveform data.

Due to the huge dynamic range the instrument is required to work with, there are two channels of different sensitivity. In 1 the sampling values of the high power channel and the more sensitive low power channel are referred to as S_{hp} and S_{lp} , respectively. The reference channel with its sampling values S_{ref} represents the emitted (reference) signal. Note, however, that this signal is not directly available for V-Line scanners. Instead, these scanners provide an accurate value , t_{ref} , the timestamp of laser pulse emission. For all other scanners t_{ref} has to be determined from the reference channel's waveform.

Each sample block SBL_i is stored with a timestamp t_{sosbl} [1] which marks the start of the sample block. Using the sampling interval T_s , which is provided in the meta data of the waveform file, the timestamp t_i of a particular sample of index i is:

$$t_i = t_{sosbl} + i \cdot T_s \tag{1}$$

The index of the first sample in a sample block is zero. Please note that t_{sosbl} in general does not coincide with an integer multiple of T_s with respect to t_{sorg} . This is especially true for sample blocks from different channels.

1 By default this timestamp is an absolute time, I.e the same domain as t_{sorg} . This behaviour can be changed to a relative time in order to increase resolution.

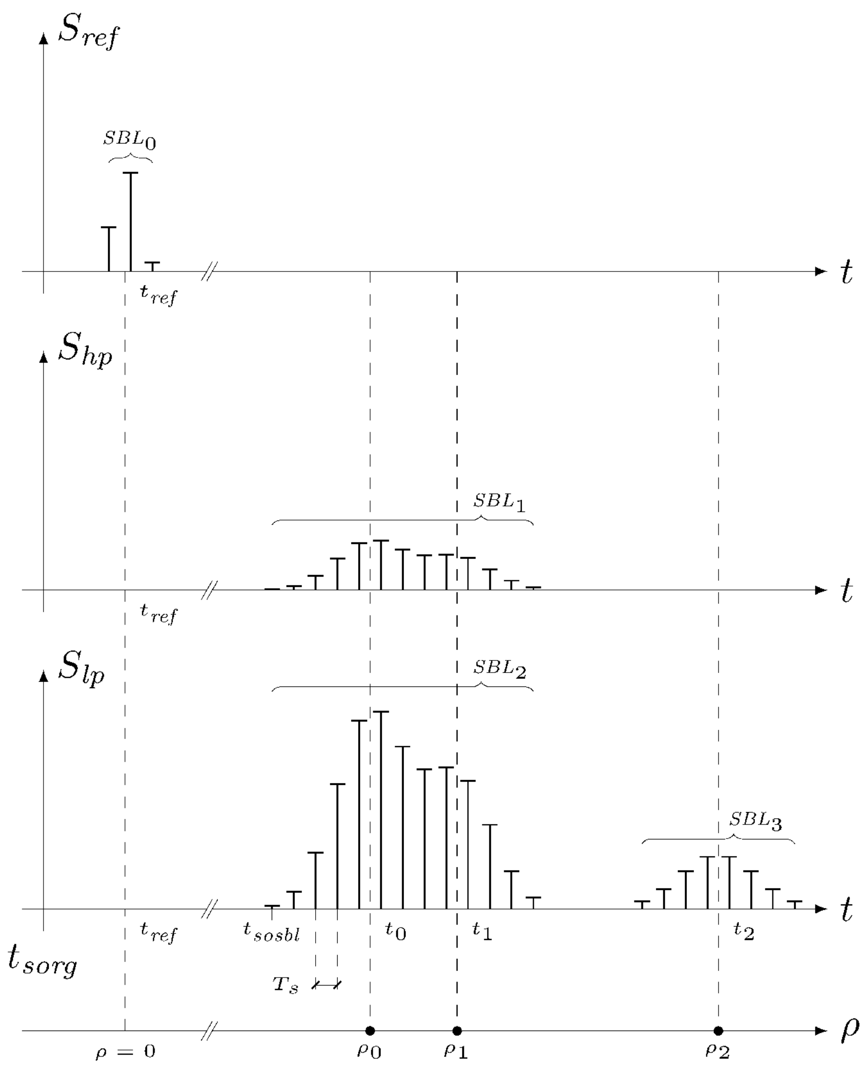

Figure 1: Digitized samples on a waveform related to one laser shot.

2 Calculation of range

The arrival times t_j of the echo signals detected for a laser shot are determined by analyzing the corresponding sample blocks. These are referred to the reference pulse to calculate the ranges to the individual targets ρ_j by applying :

$$\rho_j = \frac{v_g}{2} \cdot \left(t_j - t_{ref} \right) \tag{2}$$

where v_g is the group velocity of light (about 300,000,000 m/s).

1 illustrates the transition from the time domain to the range domain for the waveform. The instant of laser pulse emission t_{ref} corresponds to the origin of the range axis. For higher levels of the returned echo signal the estimation of the correct range will be complicated by the non-linearity of the signal detection process. Under the assumption a

gaussian pulse estimator is used for determination of target range, a calibration table[2] is made available to compensate for the range deviation.

3 *Determination of target location*

2 shows target locations $\vec{r}_j$ in the scanner's own coordinate system (x, y, z). Each waveform data record contains the position vector $\vec{o}$ and the direction vector $\vec{d}$ which describe the location of the emitted pulse at the instant t_{ref} and the direction of the beam, respectively.

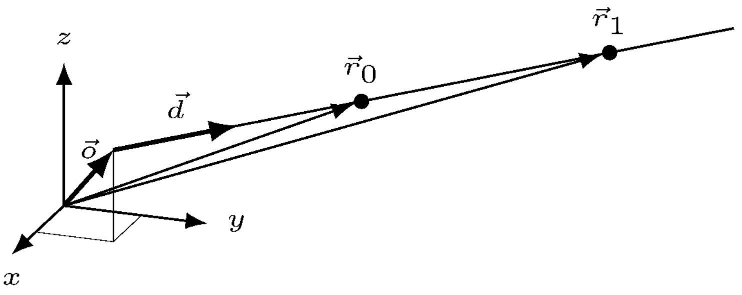

Figure 2: Target locations in the scanner's coordinate system

The target locations $\vec{r}_j$ are calculated using:

$$\vec{r}_j = \vec{o} + \vec{d}\,\rho_j \quad . \tag{3}$$

It is also possible to assign locations for the samples $\vec{s}_i$ of the waveform in the scanner's own coordinate system related to the timestamps t_i using the equation:

$$\vec{s}_i = \vec{o} + \vec{d}\,\frac{v_g}{2}\left(t_i - t_{ref}\right) \tag{4}$$

Where the timestamps t_i result from (1). A visualization of the samples in space is presented in 3.

2 For SDF Files available only as an add-on option to RiWaveLib.

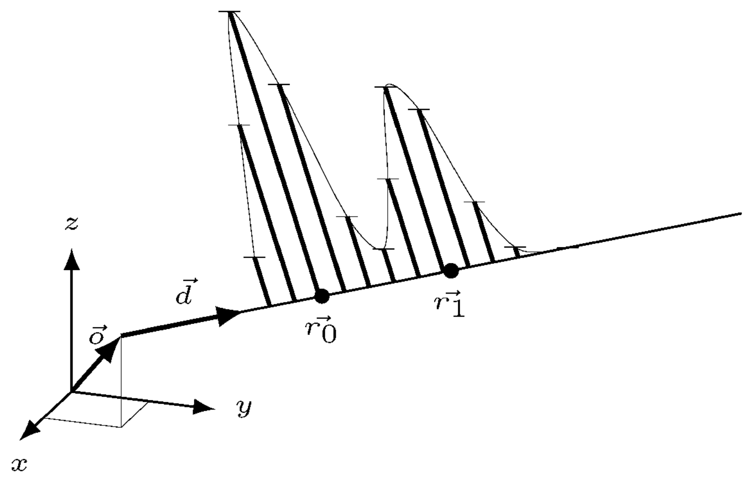

Figure 3: Representation of samples on the laser beam axis.

Part 2 Accessing the waveforms

The waveform extraction library allows access to the waveform data (sample data) in waveform data files. It consists of two DLLs (dynamic linked libraries),

- sdfifc.dll for accessing sample data files (.sdf) from Q560, Q680 and Q680i, and
- wfmifc-mt.dll for accessing waveform data files (.wfm) from V-line instruments (VZ- and VQ- scanners).

Both libraries implement the waveifc interface. This chapter describes the interface (API) for using the dynamic linked libraries (DLLs).

1 *General*

All functions return a result code indicating successful execution or failure. An error message can be retrieved using the fwifc_get_last_error function. If the operation succeeded, the return code is 0 (zero).

Please note that the memory allocation for all data takes place in the library so one does not have to take care of allocating buffer for the results. Returned pointers are only valid until a new function call is issued.

2 *Opening and closing a waveform file*

After loading the appropriate library use the fwifc_open function to open a data file and to obtain a handle for the file required for further operations.

```
fwifc_file handle;
fwifc_int32_t res;
if (res = fwifc_open("C:\Datafiles\101213_114721.sdf", &handle))
{
  fwifc_csz message;
  fwifc_get_last_error(&message);
  fprintf(stderr, message);
  exit(res);
}
```

When opened successfully, fwifc_open returns 0 and a valid handle for accessing the file.

Please note that error handling is generally not implemented in the following examples. When using the libraries the return value of the functions to handle errors should always be checked properly.

Additional information, necessary for decoding the waveforms is obtained by the fwifc_get_info function:

```
fwifc_csz instrument;          /* instrument type */
fwifc_csz serial;              /* serial number of the instrument */
fwifc_csz epoch;               /* type of time_external, e.g., date/time */
                               /* "2010-11-16T00:00:00" or */
                               /* "DAYSEC" or "WEEKSEC" or */
                               /* "UNKNOWN" if not known */
fwifc_float64_t v_group;       /* group velocity in meters/second */
fwifc_float64_t sampling_time; /* sampling interval in seconds */
fwifc_uint16_t flags;          /* GPS synchronized, ... */
fwifc_uint16_t num_facets;     /* number of scan mirror facets */
fwifc_get_info(
  handle,
  &instrument,
  &serial,
  &epoch,
  &v_group,
  &sampling_time,
  &flags,
  &num_facets
);
```

The fwifc_close function closes the file and de-allocates all resources:

```
fwifc_close(handle);
```

3 Navigating the file

Use fwifc_seek(handle, record_index), fwifc_seek_time(handle, internal_time) or fwifc_seek_time_external(handle, external_time) to seek a record and fwifc_read to read it. Use the wavefic_tell function to determine the current record number.

To determine the number of records in the waveform file, seek to the last record:

```
fwifc_seek(handle, 0xFFFFFFFF);
```

Now determine the current record number:

```
fwifc_uint32_t number_of_records;
fwifc_tell(handle, &number_of_records);
```

A function may return the error code FWIFC_ERR_MISSING_INDEX if an index requested is neither usable nor found. In this case use the fwifc_reindex function to (re)create the file:

```
fwifc_reindex(handle);
```

Note that this operation may last some time as the complete data file must be processed to create this index file.

4 Retrieving waveform data

Sample data files contain sample data records, sample data records contain laser shot information and sample blocks. To sequentially process the file use the fwifc_read function.

```
fwifc_float64_t time_sorg;      /* start of range gate in s */
fwifc_float64_t time_external;  /* external time in s relative to epoch */
fwifc_float64_t origin[3];      /* origin vector in m */
fwifc_float64_t direction[3];   /* direction vector (dimensionless) */
fwifc_uint16_t facet;           /* scan mirror facet number */
                                /* (0 to num_facets-1) */
fwifc_uint32_t sbl_count;       /* number of sample blocks */
fwifc_uint32_t sbl_size;        /* size of sample block in bytes */
fwifc_sbl_t* psbl;              /* pointer to first sample block */

fwifc_read(
  handle,
  &time_sorg,
  &time_external,
  &origin[0],
  &direction[0],
  &flags,
  &facet,
  &sbl_count,
  &sbl_size,
  &psbl
);
```

After reading a record iterate the sample blocks to retrieve the sample values:

```
// iterate sample blocks (sbi = 0 to sbl_count-1):
for (int sbi = 0; sbi < sbl_count; ++sbi)
{
  fprintf(stdout, "Start of sample block: %f\n", psbl->time_sosbl);
  // psbl->time_sosbl is the internal timestamp of the sample block
  // (= 1st sample)
  // to calculate this timestamp as external time use:
  // time_external + (time_sosbl - time_sorg)
  // iterate sample values of block (si = 0 to psbl->sample_count-1):
  fwifc_sample_t* ps = psbl->sample;
  for (int si = 0; si < psbl->sample_count; ++si)
  {
    fwifc_sample_t sample_value = *(ps);
    ps++;
    // do something useful with sample_value
  }
  psbl++;
}
```

In case the data originates from an sdf file, the precise emission time t_{ref} has to be derived from the sampled start pulse in the reference channel sample block (psbl->channel == 3). In case the data originates from a wfm file, t_{ref} is derived from a fake sample block containing no samples but only a timestamp (psbl->time_sosbl == t_{ref}).

Part 3 Reference

This section describes functions, data types, and constants of the library.

1 Error codes

The following error codes may be returned by functions:

```
FWIFC_END_OF_FILE -1
FWIFC_NO_ERROR 0
FWIFC_ERR_BAD_ARG 1
FWIFC_ERR_UNSUPPORTED_FORMAT 2
FWIFC_ERR_MISSING_INDEX 3
FWIFC_ERR_UNKNOWN_EXCEPTION 4
FWIFC_ERR_NOT_IMPLEMENTED 5
FWIFC_ERR_RUNTIME 6
```

The error message can be retrieved using the fwifc_get_last_error function.

2 Data types

```
typedef int fwifc_int32_t;
typedef short unsigned int fwifc_uint16_t;
typedef unsigned int fwifc_uint32_t;
typedef int fwifc_bool;
typedef const char* fwifc_csz;
typedef unsigned char fwifc_byte;
typedef double fwifc_float64_t;
typedef fwifc_uint16_t fwifc_sample_t;   /* sample value */
typedef struct fwifc_sbl_struct {        /* sample block structure: */
  fwifc_float64_t time_sosbl;            /* start of sample block in s */
  fwifc_uint32_t channel;                /* 0:high, 1:low, 2:saturation, */
                                         /* 3:reference */
  fwifc_uint32_t sample_count;           /* number of sample values */
  fwifc_uint32_t sample_size;            /* size of 1 sample value in bytes */
  fwifc_sample_t *sample;                /* pointer to first sample value */
} fwifc_sbl_t;
struct fwifc_file_t;
typedef struct fwifc_file_t* fwifc_file; /* file handle */
```

3 fwifc_open

```
FWIFC_API fwifc_int32_t fwifc_open
(
  fwifc_csz path, /* path and file name of the data file */
  fwifc_file *file /* handle for accessing the file */
);
```

After loading the library use the fwifc_open function to open a data file and obtain a handle for the file required for further operations.

Module sdfifc.dll is used to open sample data files (.sdf) whereas wfmifc-mt.dll is used to access waveform data files (.wfm).

```
fwifc_file handle;
fwifc_open("C:\\Datafiles\\101213_114721.sdf", &handle);
```

When opened successfully, fwifc_open returns 0 and a valid handle for accessing the file.

4 fwifc_close

```
FWIFC_API fwifc_int32_t
fwifc_close
(
  fwifc_file file
);
```

Closes the file and frees all allocated resources.

5 fwifc_get_library_version

```
FWIFC_API fwifc_int32_t
fwifc_get_library_version
(
  fwifc_uint16_t *api_major, /* major version number */
  fwifc_uint16_t *api_minor, /* minor version number */
  fwifc_csz *build_version,  /* build information string */
  fwifc_csz *build_tag       /* build tag information */
);
```

Use this function to retrieve information about the module interface.

- api_major and api_minor will receive the version number (major.minor) of the interface. A change of the minor version number means that functions have been added but the interface is still compatible. A change of the major number means a significant change of the interface functions so the interface may not be compatible with the previous version.
- build_version string contains information about the target system of the build process.
- build_tag string contains information about the build process.

6 fwifc_get_last_error

```
FWIFC_API fwifc_int32_t
fwifc_get_last_error
(
  fwifc_csz *message
);
```

Use this function to retrieve the error message of the last operation. Memory allocation for the message string is managed by the library. Please note that the string pointer is valid as long as no other function of the library is called.

7 fwifc_reindex

```
FWIFC_API fwifc_int32_t
fwifc_reindex
(
  fwifc_file file
);
```

(Re-)Create index file for active file handle. An index file is required for navigating in the data file (.sdf or .wfm file). If an index already exists, it is disposed and rebuilt. Please note that this operation may take some time (blocking).

The new index file is placed in the same directory the data file is located in. Please make sure you have sufficient rights to create/rewrite the index file in this directory.

8 fwifc_set_sosbl_relative

```
FWIFC_API fwifc_int32_t
fwifc_set_sosbl_relative
(
    fwifc_file file
    , fwifc_int32_t value    /* 1=relative, 0=absolute (default)*/
);
```

The t_{sosbl} timestamp is in the same time-domain as t_{sorg} by default. The resolution of the latter is lower however in order to allow for large values without truncation. It's main use is as unique identifier for a certain pulse. Consequently for large values of t_{sosbl} this would mean a precision loss with respect to the lowest significant digits. Setting the t_{sosbl} to relative readouts avoids this loss of precision. Care must be taken still when adding or subtracting t_{sorg} and t_{sosbl} values to avoid extinction of the least significant digits.

9 fwifc_get_info

Use fwifc_get_info() to retrieve information about the file:

```
FWIFC_API fwifc_int32_t fwifc_get_info
(
  fwifc_file file,               /* file handle from fwifc_open */
  fwifc_csz *instrument,         /* instrument type */
  fwifc_csz *serial,             /* serial number of the instrument */
  fwifc_csz *epoch,              /* type of time_external, e.g., date/time */
                                 /* "2010-11-16T00:00:00" if known */
                                 /* or "DAYSEC" or "WEEKSEC" */
  fwifc_float64_t *v_group,      /* group velocity */
  fwifc_float64_t *sampling_time, /* sampling interval in seconds */
  fwifc_uint16_t *flags,         /* GPS synchronized, ... */
  fwifc_uint16_t *num_facets     /* number of scan mirror facets */
);
```

Where flags are defined as:

```
FWIFC_FLAGS_SYNCHRONIZED = 0x01; /* GPS synchronized */
```

If epoch is a timestamp time_external is the number of seconds from the offset specified by epoch. If epoch is “DAYSEC” or “WEEKSEC” time_external is given in seconds of the day or seconds of the week, respectively.

10 *fwifc_get_calib*

```
FWIFC_API fwifc_int32_t
fwifc_get_calib
(
    fwifc_file file
    , fwifc_uint16_t  table_kind  /* one of the FWIFC_CALIB_xxx constants */
    , fwifc_uint32_t   *count     /* length of returned table */
    , fwifc_float64_t* *abscissa  /* table values, valid until next call into*/
    , fwifc_float64_t* *ordinate  /* library. */
);
```

This function may be used to retrieve several kinds of calibration tables. The kind of table is provided by the caller by *table_kind* the values of which are chosen from the predefined set of constants:

```
#define FWIFC_CALIB_AMPL_CH0          0
#define FWIFC_CALIB_AMPL_CH1          1
#define FWIFC_CALIB_RNG_CH0           2
#define FWIFC_CALIB_RNG_CH1           3
```

Tables 0 and 1 are applicable to the high- and low-power channels. Assuming a Gaussian pulse estimation they allow a conversion from the estimated maximum of the pulse in units of digitizer steps into a readout of Decibels, which expresses a ratio of the amplitude with respect to the detection limit.

Tables[3] 2 and 3 again are applicable to the high- and low-power channels allowing a compensation of the non-linear behavior of the receiver circuitry for stronger signals. Applicability of these tables again assumes that a Gaussian pulse estimator has been used for target range estimation. The tables then allow a conversion of the estimated maximum of the pulse in units of digitizer steps into a correction of the pulse location given in units of seconds.

The tables are made of *count* pairs of *(abscissa, ordinate)* values.

3 Please note that a separate license is needed to use these tables. (See License Manager)

11 fwifc_read

Use the fwifc_read function to read a sample data record.

```
FWIFC_API fwifc_int32_t fwifc_read
(
  fwifc_file file,
  fwifc_float64_t *time_sorg,     /* start of range gate in s */
  fwifc_float64_t *time_external, /* external time in s relative to epoch*/
  fwifc_float64_t *origin,        /* origin vector in m */
  fwifc_float64_t *direction,     /* direction vector (dimensionless) */
  fwifc_uint16_t *flags,          /* GPS synchronized, ...*/
  fwifc_uint16_t *facet,          /* scan mirror facet number */
  fwifc_uint32_t *sbl_count,      /* number of sample blocks */
  fwifc_uint32_t *sbl_size,       /* size of sample block in bytes */
  fwifc_sbl_t* *sbl               /* pointer to first sample block */
);
```

Where flags are defined as:

```
FWIFC_FLAGS_SYNCHRONIZED = 0x01; /* GPS synchronized */
FWIFC_FLAGS_SYNC_LASTSEC = 0x02; /* synchronized within last second */
FWIFC_FLAGS_HOUSEKEEPING = 0x04; /* this is a housekeeping block */
```

In case sbl_count is 0 (zero) no sample block is available in this record.

If a sample data record was successfully read from the file, then the sbl pointer points to the first sample block.

A sample block is described by the fwifc_sbl_t type and contains sample values of a channel (reference pulse, low power, high power).

The library takes care of memory allocation for the data. Please note that the memory is valid until next function call only. For further processing copy all required data right after reading.

12 fwifc_seek

```
FWIFC_API fwifc_int32_t
fwifc_seek
(
  fwifc_file file,
  fwifc_uint32_t index
);
```

Seek to record index. First index is 1 (one). To find the last record in the file use:

```
fwifc_seek(handle, 0xFFFFFFFF); // highest possible index
```

Use fwifc_tell to determine the number of records in the file:

```
fwifc_tell(handle, &num_records);
```

13 fwifc_seek_time

```
FWIFC_API fwifc_int32_t
fwifc_seek_time
(
  fwifc_file file,
  fwifc_float64_t time
);
```

Seek to time (internal timestamp in seconds).

14 fwifc_seek_time_external

```
FWIFC_API fwifc_int32_t
fwifc_seek_time_external
(
  fwifc_file file,
  fwifc_float64_t time
);
```

Seek to external time in seconds (day or week seconds). This operation requires GPS synchronized scan data.

15 fwifc_tell

```
FWIFC_API fwifc_int32_t
fwifc_tell
(
  fwifc_file file,
  fwifc_uint32_t *index
);
```

Returns the index of the record, that will be read on the next call of fwifc_read.

Part 4 License Manager[4]

Some of the functions available with this library may need a separate license. The validity of such a license will be assessed by presence of a dongle that can be bought at Riegl's. Support of Riegls License Server will be available in the future.

1 Developer

A developer who is using the RiWaveLib within his own software product will need to distribute to the end-user a separate additional DLL, *lmifc32.dll* or *lmifc64.dll* matching the version of wfmifc. Naming on Linux is a little different, following the usual naming conventions on this operating system. The DLL shall be placed into the same directory as the library DLL. It is recommended to integrate the procedure outlined in the next section into the setup program of the application to make the process easier for the end user. Both of the required programs, **Install.exe** and **instlic.exe** can run as a non-interactive process when provided with proper command line switches. Hint: The *Install.exe* has a help menu that show you the possible command line switches. The *instlic.exe* will give information when invoked as *instlic –help*.

2 User

You should have received a 32 digit hexadecimal **license string** and a **dongle** that has to be attached to a USB port of the computer. ATTENTION: On windows operating system do not plug in the dongle until the dongle driver has been installed.

- Install the windows dongle drivers: Invoke keylok_install*.exe* (on Windows) *or keylok_install* (on Linux) from the *bin* subdirectory. Select the USB checkbox and press OK. If you inadvertently have plugged in the dongle before you have started the Install.exe you need to use the *keylok_install.exe* to remove everything again and restart from a fresh state: Remove all dongles from the system, run *keylok_install.exe*, select *Uninstall* check box, when the message that the uninstall finished is displayed reboot the system, make sure that your dongle is not attached, re-run *Install.exe*.
- Insert your dongle into a free USB slot and wait for the driver installer to complete.
- Add your license key: Open a command shell and invoke *instlic.exe* from the bin subdirectory. You will get a menu from which you can enter your license code and save it in the computer.
- You should now be able to access the library functions which are under license control.

4 Currently only the RiWaveLib for SDF files offers additional licensed options and needs LicenseManager.