# TRACETUNER PREMIUM™ SOFTWARE MANUAL

March 2000

Version 1.1

**Paracel® Inc.**

80 South Lake Avenue
Suite 650
Pasadena, CA 91101
Phone: (626) 744-2000
Fax: (626) 744-2001
www.paracel.com

CHAPTER 1

# *Introduction*

Paracel's TraceTuner is a powerful software application that puts calibration and quality value tools in the hands of the user. With TraceTuner, you can enhance peak determination, adjust base calls, and calculate both error probabilities and quality values.

TraceTuner gives you the ability to:

- Reanalyze the peaks in an electropherogram
- Predict the base calling quality values
- Adjust ambiguous base calls

This manual describes Paracel's TraceTuner 1.1. TraceTuner 1.1 contains the following major enhancements:

- An enhanced peak processing technique is implemented which uses a simple electrophoresis model to compute and utilize "intrinsic" peak characteristics instead of the "apparent" ones used by TraceTuner 1.0.

- A new trace parameter has been implemented to improve the accuracy of quality value prediction.
- A new base recalling procedure has been implemented. It performs more aggressive editing of ABI base calls, in particular, rejection of some of the ABI base calls and insertion of new ones.

The core of the TraceTuner software is a set of algorithms optimized to review ABI electropherograms and base calls, and to adjust those base calls and assign quality values to each. The quality values correspond to predictions of the probability of each base being called in error, based on certain characteristics of the associated electropherogram peak and how it relates to the peaks around it. Since the nature of the peaks and traces is very dependent on the sequencing environment, the error probabilities must be carefully calibrated in similar conditions as the data to be analyzed. The default calibration included with TraceTuner was trained on ABI PRISM® 3700 DNA Analyzer sequence data using Pop5 Big Dye Terminator chemistry, and is suitable for use with similarly generated data. Pop6 Big Dye Terminator chemistry calibration is also included.

Originally, quality values were developed to support an automated assembly pipeline based on Dr. Phil Green's phred and phrap software. Because TraceTuner uses the same file formats and definitions of quality values as phred, it can similarly be used in the first step in a bioinformatics pipeline. This offers the added benefit of greater accuracy on sequences produced on the ABI PRISM® 3700 DNA Analyzer.

To provide accurate values under normal conditions, the TraceTuner software comes with standard calibrations for the ABI PRISM 3700 DNA Analyzer. However, since no two pieces of lab equipment are used in identical circumstances, Paracel also offers custom calibration of TraceTuner. This calibration allows you to obtain the longest, clearest reads possible, creating accurate data for use in assembly programs and trimming tools.

The TraceTuner software package also includes a C language Application Programmer's Interface. This allows you to incorporate TraceTuner functionality into your custom-written software applications.

TraceTuner is compatible with PE Corporation's BioLIMS and BASIS, University of Washington's Phred and Phrap, and Xiaoqiu Huang's CAP assembly programs.

TraceTuner is supported on the Sun® Microsystem's Solaris® 5.5.1, 5.6 and 5.7 operating systems running on a Sun, Compaq's DIGITAL™ UNIX® 4.0d running on a DEC Alpha, and Solaris 2.7 running on an Intel processor.

### UNIX Installation

1. Copy the compressed tar file `ttuner1.1<`*version*`>.tar.Z` with a name containing the version corresponding to your platform from the CD into your current directory
2. Uncompress and tar the `ttuner1.1<`*version*`>.tar` file by typing:

   ```
   zcat ttuner1.1<version>.tar | tar xvf -
   ```

   A directory called `ttuner1.1<`*version*`>` will be created.
3. To run TraceTuner, change your current directory to `ttuner1.1<`*version*`>`:

   ```
   cd ttuner1.1<version>
   ```

You can now run TraceTuner 1.1 by typing `ttuner` with the proper option(s). See the `README` file for details.

TraceTuner is also available for the Microsoft® Windows NT® 4.0. The PC version of TraceTuner is also included on the CD-ROM.

### Installation on a PC

To install TraceTuner, use the following steps:

1. Place the TraceTuner CD in the CD_ROM drive of your computer.
2. Click on the “My Computer” icon on your Windows NT Desktop.
3. Click on the CD-ROM icon.
4. Click on the PC executable file named `Tracetuner<`*version*`>_install.exe` to run the Installer.
5. Follow all the directions given by the Installer. Be sure to read all the instructions carefully. Once you have completed these instructions, TraceTuner may be run.
6. To start TraceTuner, click on the Start button on the bottom left side of your screen. Select “Programs” then “TraceTuner” from the Programs menu. The PC version of TraceTuner includes a launcher and .phd file Viewer.
7. You may also run the TraceTuner executable from the MS-DOS command line in a fashion similar to that for the UNIX version.

The PC version of the TraceTuner manual is listed in the same menu as the TraceTuner executable.

CHAPTER 2

# *Command Line Usage*

TraceTuner is executed from the command line.

Typing `ttuner` on the command line displays the usage.

## *Helpful Hints*

- Required elements. Each element that requires a choice of parameters is shown with curly brackets “{}”.
- Optional elements are shown with brackets “[]”.
- Commands are shown in `courier`.
- Variables are shown in brackets “< >” and *italics*.

# *Command Line Usage*

### USAGE

```
ttuner [-Q][-t <lookuptable>][-qr <output_file>]
     [-v][-nocall][-recalln][edited_bases]
     [-p|-pd <dir>][-q|-qd <dir>][-s|-sd <dir>]
     {-if <fileoffiles>|-id <dir>|sample_file...}
```

### ARGUMENTS

`-Q` (Optional) This parameter specifies that no status messages should be produced. If this parameter is used in conjunction with the `-v` option, the parameter that appears last on the command line takes precedence.

`-t` TraceTuner uses a lookup table. There is a generic lookup table compiled into the TraceTuner code. You can override the generic table usage by specifying a lookup table with the command line argument (`ttuner -t` *tablename*), or you can set the user's `LOOKUP_TABLE` environmental variable to point to a specific *lookup.table*. For example:

```
set LOOKUP_TABLE=3700_Pop6_BigDye.tbl
```

The order of precedence is as follows: command line specification, environment variable and then generic compiled version.

`-qr` (Optional) This option forces TraceTuner to generate a "quality report", which is similar to that generated by phred with the same option. The report shows the read length distribution for a given dataset, that is, a number of reads (frag-

ments) as a function of the number of high-quality bases (QV20 or better) in the read. It will be written into a file named *output_file*.

`-v` (Optional) This parameter specifies that TraceTuner produce additional process status messages. The more `v`'s entered on the command line, the more process status messages produced. The default (without any `-Q` or `-v` options) is level 1; after level 3 (`-vvv`), there is no change in verbosity. If this parameter is used in conjunction with the `-Q` option, the parameter that appears last on the command line takes precedence.

`-recalln` (Optional) This parameter forces TraceTuner 1.1 to call bases like TraceTuner 1.0, that is, only recall 'N's and do not change any other base calls which it reads from sample file. The option can not be used together with `-nocall`.

`-edited_bases` (Optional) This option forces TraceTuner to read edited base calls and locations from sample file(s) and "start" from them when recalling bases. By default, the TraceTuner reads and starts from called bases and locations.

`-nocall` (Optional) This parameter disables TraceTuner base calling and sets the current sequence to the base calls that are read from the input file. By default, the current sequence is set to the TraceTuner base calls. This parameter can not be used together with `-nocall`.

If you use `-nocall`, you will get "N's". If you don't use `-nocall`, you will not get "N's".

`-p` (Optional) This parameter specifies that TraceTuner output should be written to `.phd.1`-formatted files, in the current working directory.

`-pd` <*dir*> (Optional) This parameter specifies that TraceTuner output be written to `.phd.1`-formatted files in the specified directory.

`-q` (Optional) This parameter specifies that TraceTuner write its output to `.qual`-formatted quality value files in the current directory.

`-qd` <*dir*> (Optional) This parameter specifies that TraceTuner write its output to `.qual`-formatted files in the specified directory.

`-s` (Optional) This parameter specifies that TraceTuner write its (potentially) recalled bases in FastA format in the current working directory.

`-sd` <*dir*> (Optional) This parameter specifies that TraceTuner write its (potentially) recalled bases in FastA format in the specified directory.

`-if` <*fileoffiles*> (Required if neither `-id` nor `sample_file` is specified.) This parameter specifies that TraceTuner read *fileoffiles*, and treat each line as a sample filename. An example of a *fileoffiles* is as follows:

```
\home\watson\ttuner\test_data\data_file1
\home\watson\ttuner\test_data\data_file2
\home\watson\ttuner\test_data\data_file3
```

`-id` <*dir*> (Required if neither `-if` nor `sample_file` is specified.) This parameter specifies that TraceTuner process every file in the specified directory.

*sample_file* (Required if neither `-id` nor `-if` is specified.) This parameter specifies the sample file(s) to be used in the TraceTuner process.

At least one of the `-p`, `-pd`, `-q`, `-qd`, `-s`, `-sd` or `-qr` options must be specified, so that TraceTuner knows which format to use for output (more than one option may be specified, if desired, in which case the output will be written in each of the specified formats).

If the `-if` or `-id` arguments are not specified, sample file names must be provided on the command line.

## Examples of Command Line Requests

***Example 1***

`ttuner -F -Q -pd` *`/mydir/seqoutput`* `-if` *`/mydir/dnaseqs`*

In this example, the TraceTuner executable is invoked with the following specifications:

- `-F` Processing will not be stopped if an error is encountered.
- `-Q` No status messages will be produced.
- `-pd` The output will be in Phred formatted files and will be saved in the *seqoutput* directory.
- `-if` The *fileoffiles* named *dnaseqs* containing the input sequences will be used in the TraceTuner process.

***Example 2***

`ttuner -F -pd` *sept15* `-qd` *sept15* `-sd` *sept15* `-id`
*/home/watson/ttuner/test_data*

- `-F` Processing will not be stopped if an error is encountered.
- `-pd` The output will be in Phred formatted files and will be saved in the *sept15* directory.
- `-qd` The output will also be in `.qual` formatted files and will be saved in the *sept15* directory.
- `-sd` The (potentially) recalled bases will be written in the *sept15* directory in FastA format.
- `-id` Every file in the */home/watson/ttuner/test_data* directory will be processed.

***Example 3***

```
ttuner -vv -sd sept16 test_data1.ab1
test_data2.ab1 test_data3.ab1
```

- `-vv` Additional process messages will be produced.
- `-sd` The (potentially) recalled bases will be written in the directory in FastA format.
- `test_data1.ab1`, `test_data2.ab1` and `test_data3.ab1` are input data files

CHAPTER 3

# *Example Output*

This chapter includes examples TraceTuner output files. These files can be in any of the following formats:

- .phd files
- .qual files
- .fasta files
- Quality Value report

## *Phd File Format*

Phd files are the default mode for TraceTuner. Phd files include a header with data describing the output along with the revised base calls, assigned quality values and peak locations. For ease of recognition, the following sample of an abbreviated .phd file is provided.

```
BEGIN_SEQUENCE A02_005.ab1

BEGIN_COMMENT

CHROMAT_FILE: AO2_005.ab1
ABI_THUMBPRINT: 0
PHRED_VERSION: TT_1.1
CALL_METHOD: ttuner
QUALITY_LEVELS: 44
TRACE_ARRAY_MIN_INDEX: 0
TRACE_ARRAY_MAX_INDEX: 12923
CHEM: term
DYE: big

END_COMMENT

BEGIN_DNA
c 14 11
g 13 25
c 12 38
...
a 9 12902
a 9 12911
c 9 12911
END_DNA

END_SEQUENCE
```

## *Quality Value Files*

Quality value files are another form of output. Only the quality value information will be given. The following is an example of a Quality Value file. Each value corresponds to the quality value of the nucleotide in that position, i.e. the first value is the quality value of the first nucleotide in the sequence. The values are separated by whitespace.

```
>HP0001
  7 12  8  8  8 11 15 26 21 21 11 11  8 11 19 21 24
 19 15  8 11 19 20 28 28 33 27 27 27 27 23  8  8 18
 23 27 27 23 27 27 30 36 36 35 35 32 32 32 32 27 28
 27 25 21  8 23 27 28 28 27 15 21 23 23 34 32 32 32
 32 32 32 32 32 32 32 33 33 35 33 33 33 33 34 34 32
 32 32 32 32 32 32 32 32 32 32 32 32 32 31 33 33 33
 32 32 32 32 32 32 27 27 27 25 27 27 27 32 30 30 30
 32 26 28 20  9  9 23 27 27 27 10 14 23 27 31 31 31
 31 29 32 16 11 22  9  8  8  8  8 29 29 31 28 30 32
 30 30 25 21 21 32 25 13 14 18 12 12 18  9  8 12 13
```

## *FastA File Format*

The FastA output option writes a FastA file containing the revised base calls only. Quality value information is not available. In FastA format, each sequence is preceded by a description line (header string) that starts with a greater-than sign.

```
>HP0001
ATGGCGACACGAACTCAAGCAGGGGGGCTGTGGTTGAATTGTTGTATGCGTTT
GAGAGCGGTAATGAAGAAATTAAAAAAATCGCTTCTAGCATGTTAGAAGAAAA
AAAGATTAAAAACAACCAACTCGCTTTCGCTTTAAGCCTTTTTAATGGCGTGT
TAGAAAAAATCAATGAAATTGACGCCCTCATCGAGCCGCATTTAAAAGACTGG
GATTTCAAGCGATTAGGGAGCATGGAAAAGGCGATTTTACGCTTAGGAGCGTA
TGAAATTGGCTTCACGCCCACGCAAAACCCTATCATCATCAATGAATGCATAG
AGCTTGGCAAACTCTACGCTGAGCCTAACACCCCTAAATTTTTAAACGCTATC
TTGGATTCTTTGAGCAAAAAGCTCACTCAAAAACCCTTG
```

Spaces, tabs, and carriage returns in the sequence are ignored. The start of a new sequence is indicated by the presence of a new description line.

## *Quality Value Report*

The Quality Value Report is created by using the `-qr` *<output_file>* option. This report is another way to demonstrate the high quality of the reads in a group of samples. This file shows the "bucket", the number of reads with a quality value 20 or above in a given "bucket" and the number of files used to create the report. Buckets are used for sorting various reads by length.

```
TraceTuner Quality Value Histogram Data
Number of Files Reported = 1304

            Num of Reads
Bucket      with QV >= 20
---------   --------------
   0-9                  20
  10-19                  7
  20-29                 11
  30-39                 13
  40-49                 18
  50-59                 24
  60-69                 31
  70-79                 23
  80-89                 30
  90-99                 23
.............................
 690-699                67
 700-709                72
 710-719                66
 720-729                80
 730-739               107
 740-749                77
 750-759                67
 760-769                43
 770-779                22
 780-789                 8
 790-799                 2

Average                538
```

CHAPTER 4

# *Application Programmer's Interface*

This chapter describes the data structures and functions that constitute the TraceTuner Application Programmer's Interface (API). It is arranged alphabetically by structure followed by functions.

The API allows users to create their own programs using various TraceTuner functions, allowing TraceTuner to be integrated into a variety of custom environments.

All of these functions may be found in `libtt.a`, in the installation directory.

## *Structures*

The Structures listed below are described in the following section.

`BtkMessage`

`BtkLookupEntry`

`BtkLookupTable`

`Options`

### STRUCTURE

```
typedef struct {
int             code;
char            text[BTKMESSAGE_LENGTH];
} BtkMessage;
```

### DESCRIPTION

`BtkMessage` is the data structure used by all API functions to return error information.

### FIELDS

| | |
|---|---|
| *code* | A 32-bit value used to return an error code |
| *text* | A textual error message |

### STRUCTURE

```
typedef struct btk_quality_lookup_entry {
double phr3;
double phr7;
double psr;
double pre;
int qval;
} BtkLookupEntry;
```

### DESCRIPTION

This structure represents a single line of the quality value lookup table. The four `double` parameters are the threshold values for each quality value entry. Each base call's parameters must be less than or equal to each of the four fields of the table entry to be assigned to the associated quality value.

Applications usually do not need to manipulate these values directly. The function `Btk_read_lookup_table()` will fill in the values from the lookup table file.

### SEE ALSO

`Btk_read_lookup_table()`

`BtkLookupTable`

`Btk_destroy_lookup_table()`

### STRUCTURE

```
typedef struct btk_quality_lookup_table {
int                num_entries;
BtkLookupEntry    *entries;
} BtkLookupTable;
```

### DESCRIPTION

A collection of `BtkLookupEntry` structures that together describe a quality value lookup table.

Applications usually do not need to manipulate these values directly. The function `Btk_read_lookup_table()` will fill in the values from the lookup table file.

### SEE ALSO

`Btk_read_lookup_table()`

`BtkLookupEntry`

`Btk_destroy_lookup_table()`

### STRUCTURE

```
typdef struct{
  char        file_name[200];
  int         Verbose;
  int         nocall;
  int         recalln;
  int         edited_bases;
}Options;
```

### DESCRIPTION

This data structure is used to pass some of the command line options of the function `Btk_compute_qv()`.

### FIELDS

*file_name* — This field is a char array containing the sample file name.

*Verbose* — This field determines whether or not to write status messages to stderr, and in what quantity: type 0 for no status messages, 1 for some status messages and 3 for all the status messages.

*nocall* — This field specifies whether to skip recalling bases. A value of 0 means not to skip recalling bases and any non-zero number means to skip recalling bases.

*recalln* — This argument specifies whether to use a Trace-Tuner 1.0-like base recalling procedure. A value of zero specifies to use a new base calling algorithm which changes the length of the called bases and location arrays. Any non-zero number specifies to use the best guess and does not change the length of the array of called bases and locations.

*edited_bases* — This field specifies whether to start base calling from edited bases. A value of 0 specifies to read

called bases and locations from sample files and to start from them when recalling bases. Unity (1) specifies to read and use edited bases and locations from sample files.

**SEE ALSO**

`Btk_compute_qv()`

## *Functions*

The Functions listed below are described in the following section.

```
Btk_compute_qv()

Btk_destroy_lookup_table()

Btk_output_fasta_file()

Btk_output_phd_file()

Btk_output_quality_values()

Btk_read_lookup_table()

Btk_read_sample_file()

Btk_release_file_data()
```

### FUNCTION

```
int Btk_compute_qv(
int             *numbases,
char            **bases,
int             **locs,
int             datalen,
int             **chromatogram,
char            basechars,
BtkLookupTable  *table,
int             *qualval,
Options         options,
BtkMessage      *message
);
```

### DESCRIPTION

This function computes quality values and updates base calls and locations for the chromatograms passed as arguments. The input value of **num_bases* is the initial length of the arrays of bases and locations as returned by the function *Btk_read_sample_file*. These arrays should be allocated and populated with the data from the sample file. The user should also allocate **num_bases* ints for the array qualval and pass the pointer to this array. This functions will reallocate the array as needed, compute the quality values and place them into the array.

### ARGUMENTS

| | |
|---|---|
| **num_bases* | Pointer to the input length of the array of called bases and the array of peak locations |
| ***bases* | Pointer to array of called bases |
| ***locs* | Pointer to array of called peak locations |
| *datalen* | Input length of chromatograms array |

| | |
|---|---|
| *chromatograms* | Input arrays which store chromatographic data for each of the dyes |
| *basechars* | Array of bases corresponding to the colors `0,1,... NUM_COLORS-1` |
| **table* | Pointer to a populated `BtkLookupTable` structure returned by `Btk_read_lookup_table()`, which represents a lookup table. An application may pass `NULL` for this argument to direct the function to use default lookup table (which is based on 3700 data). |
| ***qualval* | Pointer to an array of quality values |
| *Options* | Structure, members of which are some of the command line options (sample file name, Verbose, _nocall, recalln and edited_bases) |
| *message* | Pointer to the user-supplied error message structure `BtkMessage` |

**SEE ALSO**

`Options`

`Btk_read_sample_file()`

### FUNCTION

```
void Btk_destroy_lookup_table(
BtkLookupTable    *table
);
```

### DESCRIPTION

This function reclaims storage used by a lookup table that was created by the `Btk_read_lookup_table()` function. Applications should not attempt to use a `BtkLookupTable` after passing it to this function.

### ARGUMENTS

| | |
|---|---|
| *table* | The pointer returned by `Btk_read_lookup_table()`. |

### SEE ALSO

`Btk_read_lookup_table()`

`BtkLookupTable`

### FUNCTION

```
BtkLookupTable *Btk_read_lookup_table(
char             *path
);
```

### DESCRIPTION

This function reads a quality value lookup table file and converts it into a `BtkLookupTable` structure. This function must be called with the name of a valid lookup table file before quality values can be calculated. When the application is finished with it, the resources used by the table should be returned to the system by calling `Btk_destroy_lookup_table()`.

### ARGUMENTS

| | |
|---|---|
| *path* | The path name of the file containing the quality value lookup table, often '`lookup.table`'. |

### SEE ALSO

`Btk_destroy_lookup_table()`

`BtkLookupTable`

### FUNCTION

```
int Btk_output_fasta_file(
char            *file_name,
char            *path,
char            *called_bases,
int             num_bases,
int             verbose
);
```

### DESCRIPTION

This function writes out the base call sequence in FastA format, to the specified file.

### ARGUMENTS

| | |
|---|---|
| *file_name* | Name of the sample file |
| *path* | Path name of the directory where the `.seq` file is to be written |
| *called_bases* | Array of base calls |
| *num_bases* | Number of elements in *called_bases* |
| *verbose* | Determines whether or not to write status messages to `stderr`, and in what quantity; type 0 for no status messages, 1 for some status messages, 2 for more status messages and 3 for all the status messages. |

### SEE ALSO

`Btk_output_phd_file()`

`Btk_output_quality_values()`

### FUNCTION

```
int Btk_output_phd_file(
char            *file_name,
char            *path,
char            *called_bases,
int             *called_locs,
int             *quality_values,
int             num_bases,
int             num_datapoints,
int             nocall,
char            *chemistry,
int             verbose
);
```

### DESCRIPTION

This function writes out the base calls, called locations, and quality values to the directory specified by the *path*. The file written has the same name as the sample file (whose name is in the *file_name* argument), with a suffix of '`.phd.1`'. The file is in Phred format.

### ARGUMENTS

| | |
|---|---|
| *file_name* | Name of the sample file |
| *path* | Path name of the directory in which the `.phd` file should be written |
| *called_bases* | Array of base calls |
| *called_locs* | Array of called base locations |
| *quality_values* | Array of quality values |
| *num_bases* | Number of elements in the *called_bases*, *called_locs* and quality values arrays |
| *num_datapoints* | Number of data points in the read |

| | |
|---|---|
| *nocall* | This field specifies whether to skip recalling bases. A value of 0 means not to skip recalling bases and any non-zero number means to skip recalling bases. |
| *chemistry* | Optional string that describes the chemistry used (primer or terminator) |
| *verbose* | Determines whether or not to write status messages to `stderr`, and in what quantity; type 0 for no status messages, 1 for some status messages, 2 for more status messages and 3 for all the status messages. |

**SEE ALSO**

`Btk_output_fasta_file()`

`Btk_output_quality_values()`

### FUNCTION

```
int Btk_output_quality_values(
char            *file_name,
char            *path,
int             *quality_values,
int             num_values,
int             verbose
);
```

### DESCRIPTION

This function writes out quality values, in FastA format, to the directory specified by the *path* argument.

### ARGUMENTS

| | |
|---|---|
| *file_name* | Name of the sample file |
| *path* | Pathname of the directory in which the `.qual` should be written |
| *quality_values* | Array of quality values |
| *num_values* | Number of elements in *quality_values* |
| *verbose* | Determines whether or not to write status messages to `stderr`, and in what quantity; type 0 for no status messages, 1 for some status messages, 2 for more status messages and 3 for all the status messages. |

### FUNCTION

```
int Btk_read_sample_file(
char            *file_name,
int             *num_bases,
char            **called_bases,
int             **called_locs,
int             edited_bases,
int             *num_values,
int             **avals,
int             **cvals,
int             **gvals,
int             **tvals,
char            **call_method,
char            **chemistry,
BtkMessage      *message,
int             verbose
);
```

### DESCRIPTION

This function reads the sample file named `file_name` into the other arguments. It assumes that the file is in ABI format. When finished with the data, the application can return the resources consumed by the data to the system by calling `Btk_release_file_data()`.

### ARGUMENTS

| | |
|---|---|
| *file_name* | Name of the sample file |
| *num_bases* | Address of a variable that will be set to the number of bases |
| *called_bases* | Address of a variable that will be set to called bases |
| *called_locs* | Address of a variable that will be set to called base locations |
| *edited_bases* | This argument specifies whether to start base calling from edited bases. A value of 0 specifies to |

| | |
|---|---|
| | read called bases and locations from sample files and to start from them when calling bases. Unity (1) specifies to read and use edited bases and locations from sample files. |
| *num_values* | Address of a variable that will be set to the number of trace points |
| *avals* | Address of a variable that will be set to an array of A trace points |
| *cvals* | Address of a variable that will be set to an array of C trace points |
| *gvals* | Address of a variable that will be set to an array of G trace points |
| *tvals* | Address of a variable that will be set to an array of T trace points |
| *call_method* | Optional address of a variable that will be set to a string that describes the base calling method |
| *chemistry* | Optional address of a variable that will be set to a string that describes the chemistry used (i.e. primer or terminator) |
| *message* | Pointer to the caller-supplied error message structure `BtkMessage` |
| *verbose* | Determines whether or not to write status messages to `stderr`, and in what quantity; type 0 for no status messages, 1 for some status messages, 2 for more status messages and 3 for all the status messages. |

**SEE ALSO**

`Btk_release_file_data()`

### FUNCTION

```
void Btk_release_file_data(
char            *called_bases,
int             *called_locs,
int             **chromatogram,
char            *call_method,
char            *chemistry
);
```

### DESCRIPTION

This function reclaims the resources consumed by the data read in by `Btk_read_sample_file()`.

### ARGUMENTS

| | |
|---|---|
| *called_bases* | Pointer to called bases originally set by `Btk_read_sample_file()` |
| *called_locs* | Pointer to called base locations originally set by `Btk_read_sample_file()` |
| *chromatogram* | Array of pointers to color specific trace points originally set by `Btk_read_sample_file()` |
| *call_method* | One of the strings originally set by `Btk_read_sample_file()` |
| *chemistry* | One of the strings originally set by `Btk_read_sample_file()` |

### SEE ALSO

`Btk_read_sample_file()`

CHAPTER 5

# *Sample API Program*

This chapter provides a basic programming example using the TraceTuner API functions.

The following example illustrates how to use a few functions from the API; one from each category.

This example program reads a lookup table, reads one or more sample files, computes the quality values and writes out the quality values in FastA format.

```
/**
 **  Copyright (c) 2000 Paracel, Inc. All rights
reserved
 **
 **  example.c - Sample application program to output
.qual files from
 **             ABI sample file inputs using the Trace-
Tuner library.
 **/
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include "Btk_qv.h"
#include "Btk_qv_data.h"
#include "Btk_lookup_table.h"
#include "Btk_compute_qv.h"
#include "Btk_qv_io.h"

int
main(int argc, char *argv[])
{
    int i, n;
    char *lookup_table, *smp, *smptail;
    BtkLookupTable *table;
    int nbases, nvals;
    char *bases;
    int *locations, *vals[4], *qv;
    BtkMessage  msg;
    Options options;

    if(argc < 2) {
       fprintf(stderr, "usage: %s <samplefiles...>\n",
argv[0]);
        exit(0);
    }
    /*
     *  If a non-standard lookup table is specified as
an environmental
     *  variable use it, otherwise NULL gets the
default.
     */
    lookup_table = getenv("LOOKUP_TABLE");

    if (lookup_table != NULL) {
        if ((table =
Btk_read_lookup_table(lookup_table)) == NULL) {
            fprintf(stderr, "%s: couldn't read lookup
table\n",
                lookup_table);
            exit(1);
        }
        fprintf(stderr, "%s: %d entries\n",
lookup_table,
            table->num_entries);
    }
    else {
```

```
        fprintf(stderr, "%s: using built-in table\n",
argv[0]);
        table = NULL;
    }

    for (n=1; n < argc; n++) {
        smp = argv[n];
        if ((smptail = strrchr(smp, '/')) != NULL)
            smptail++;
        else
            smptail = smp;

       if (Btk_read_sample_file(smp, &nbases, &bases,
0, &locations, &nvals,
            &vals[0], &vals[1], &vals[2], &vals[3],
NULL, NULL, &msg, 0) != 0)
        {
            fprintf(stderr, "%s: couldn't read sample
file\n", smptail);
            continue;
        }

        qv = (int *)malloc(nbases * sizeof(int));

        options.nocall = 0;
        options.recalln = 0;
        options.edited_bases = 0;
        strcpy(options.file_name, smptail);

      if (Btk_compute_qv(&nbases, &bases, &locations,
nvals, vals, "ACGT",
            table, &qv, options, &msg) != 0) {

            fprintf(stderr, "%s: %s\n", smptail,
msg.text);
            goto cleanup_a_file;
        }

        Btk_output_quality_values(smp, NULL, qv,
nbases, 0);

        cleanup_a_file:
        if (qv != NULL) {
            free(qv);
            qv = NULL;
```

```
            }
            if (bases != NULL) {
                free(bases);
                bases = NULL;
            }
            if (locations != NULL) {
                free(locations);
                locations = NULL;
            }
            for (i=0; i<4; i++) {
                if (vals[i] != NULL) {
                    free(vals[i]);
                    vals[i] = NULL;
                }
            }
        }

        Btk_destroy_lookup_table(table);

        return(0);
    }
```