PARACEL®
Applied High-Performance Computing

TraceTuner™ Premium User Manual

Revision 2.0.9 ***November 2003***

1055 East Colorado Blvd.
Suite 5000
Pasadena, California 91106-2341
Phone: (626) 744-2000
Fax: (626) 744-2001
www.paracel.com

Revision 2.0.9 (11/03) supersedes Revision 2.0 (09/01).

Table of Contents

CHAPTER 1

Introduction

Overview

High-throughput DNA sequencing technology is constantly improving and downstream software must keep pace with the improved quality of the data. The best possible base calls and an accurate knowledge of their reliability are key to obtaining the longest, clearest reads. Using novel algorithms and an intrinsic peak model, Paracel's TraceTuner software reanalyses the peaks in an electropherogram to re-call the bases and to estimate their quality accurately. TraceTuner has been designed to be adaptable and responsive to changing sequencing conditions, thus ensuring the best possible quality estimates.

TraceTuner gives you the ability to:

- Reanalyze the peaks in an electropherogram
- Resolve multiple bases using a proprietary intrinsic peak model
- Adjust ambiguous base calls
- Predict the base calling quality value
- Optionally call heterozygotes
- Determine alternative base calls
- Visualize all base calls and quality values

- Accept `.ab1, .scf, .abd` and `.abi` input sample files
- Accept UNIX-compressed and gzipped input sample files
- Generate Phred/Phrap-compatible `.phd`, `.qual`, `.seq` or `.poly` output files
- Produce and visualize local alignments between TraceTuner base calls and user-supplied reference sequences
- Integrate TraceTuner into other programs using a C language API

TraceTuner can be run from the *Launcher*, a Java user interface, or directly from the command line. Using the *Launcher*, you can run TraceTuner with as little input as sample input files, or you can run TraceTuner from the command line to take full advantage of the many available processing options. This flexibility makes TraceTuner perfect for both basic and advanced users.

Another Java user interface, the *Viewer*, provides a visual representation of the results of TraceTuner processing. In addition to displaying each base in the electropherogram with a letter designation, a quality value assignment and an associated peak, the *Viewer* also provides a visual display of the original ABI base calls, alternative base calls, heterozygous base calls and alignments between TraceTuner base calls and user-specified reference sequences. The Viewer also provides a search capability so you can locate occurrences of specified base sequences in sample files, TraceTuner output files or the consensus/reference file.

The core of the TraceTuner software is a set of algorithms optimized to review ABI or SCF format electropherograms and base calls, to adjust those base calls and to assign quality values to them. 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 adjacent peaks and similar peaks in the sample. Because peak shape and trace strength are very dependent on the sequencing environment, the error probabilities must be carefully calibrated with data similar to the data being analyzed. The generic calibrations included with TraceTuner were derived from ABI3700-Pop5, ABI3700-Pop6, ABI3100-Pop6, ABI377 Dye Primer andABI377 Dye Terminator data, and are suitable for use with similarly generated data.

Originally, quality values were developed to support an automated assembly pipeline based on the `phred` and `phrap` software developed by Dr. Phil Green. Because TraceTuner uses the same file formats and definitions of quality values as `phred`, it can be used in a similar manner as the first step in a bioinformatics pipeline. TraceTuner offers the added benefit of greater accuracy on sequences produced on the ABI PRISM 3700 DNA Analyzer and the ABI PRISM 377 DNA Sequencer, as well as the capability to call heterozygotes.

To provide accurate values under normal conditions, TraceTuner comes with standard calibrations for the ABI PRISM 3700 DNA Analyzer, the ABI PRISM 3100 Genetic Analyzer and the ABI PRISM 377 DNA Sequencer. However, since no two pieces of lab equipment are used in identical circumstances, Paracel also offers more accurate custom calibrations derived from user-supplied data for an additional charge.

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

The TraceTuner software package also includes a C language Application Program Interface (API). This API allows you to incorporate TraceTuner functionality into your custom-written software applications.

TraceTuner 2.0.9 is supported on the following configurations:

- Sun Microsystems Solaris 2.6 and above running on a Sun
- Compaq DIGITAL UNIX 4.0f running on a Compaq Alpha
- Solaris 2.7 running on an Intel processor
- RedHat LINUX `7.1` running on an Intel processor
- IRIX 6.5 or higher running on SGI
- Microsoft Windows NT. TraceTuner 2.0.9 has been thoroughly tested in the Microsoft Windows NT 4.0 environment.

For more information on TraceTuner, visit:

http://www.paracel.com/products/tracetuner.php

Contact customer support by e-mail at *support@paracel.com*.

UNIX/LINUX/IRIX Installation

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

   ```
   uncompress ttuner<version>-<arch>.tar.Z
   tar -xvf ttuner<version>-<arch>.tar
   ```

   A directory called `ttuner<`*version*`>-<`*arch*`>` will be created.

3. This release requires a node-locked license. In order to obtain a license, you must send us the MAC address of your computer. To determine the MAC address, add the following line into the `~/.cshrc` file (assuming you are using csh; if you are using a different shell, add the line to the appropriate configuration file):

   ```
   setenv TTHOME TT_INSTALLATION_DIRECTORY
   ```

4. Run the `get_license_info` script under `$TTHOME` and send the resulting `license.info` file to *Paracel Support* to get the node-locked license.

5. Once you get the license file, save it under `$TTHOME` .

6. To run TraceTuner, change to the directory `ttuner`<*version*>`-`<*arch*>:

   ```
   cd ttuner<version>-<arch>
   ```

7. You can now run TraceTuner by typing `ttuner` with the proper option(s). See the on-line help file for details.

PC Installation from Premium CD

To install TraceTuner on the Windows platform, follow these steps:

1. This release requires a node-locked license. In order to obtain a license, you must send us the MAC address of your computer. To determine the MAC address, open a DOS console by selecting *Start* > *Run* and typing `cmd` in the Run dialog box.

2. In the DOS console, type the command `ipconfig /all`. The MAC address appears in the “Physical Address” line as shown in Figure 1-1.

3. Send the MAC address to *Paracel Support* to get the node-locked license.

4. Place the TraceTuner CD in the CD-ROM drive of your PC. Navigate to the file `SETUP.EXE` in the directory `ttuner2.0.9_x86-win32-4.0`. Launch this executable file.

5. Follow all the directions given by the Installer. Be sure to read all the instructions carefully.

6. Add the following line to your `C:\autoexec.bat` file:

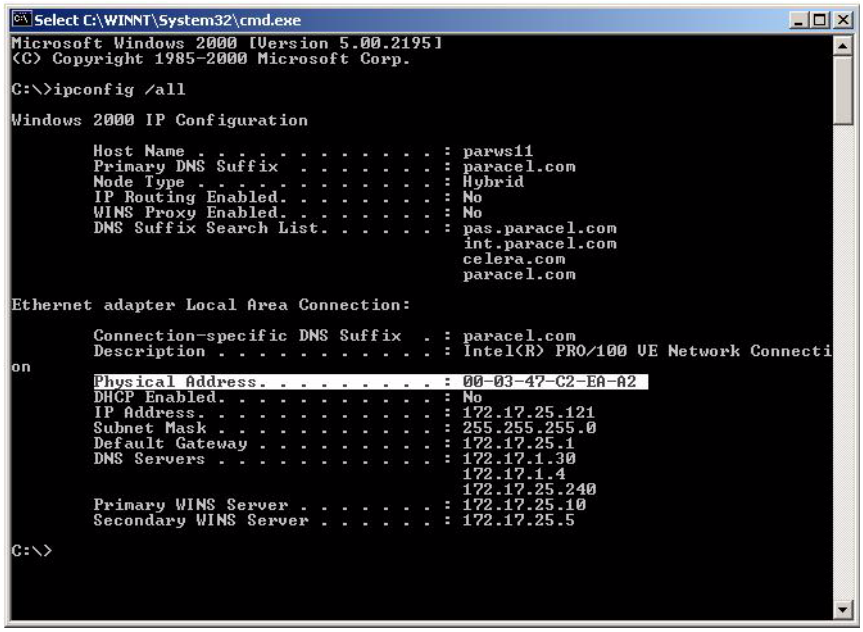
   ```
   set TTHOME=C:\Program Files\Paracel\TraceTuner
   ```

7. Save the license file you have received from Paracel in `TTHOME`.

8. To start TraceTuner, select *Start* > *Programs* > *Paracel*> *TraceTuner* > *TraceTuner Launcher.*
9. You can also run the TraceTuner executable from the MS-DOS command line in a fashion similar to that for the UNIX/LINUX/IRIX version.

Important Note: We do not recommend installing TraceTuner on the same PCs that directly control the ABI sequencers.

FIGURE 1-1: MAC Address in Windows DOS Console



Overview of TraceTuner Software

The TraceTuner installation consists of three software components:

- ***TraceTuner Launcher –*** The *Launcher* is a Java user interface application which allows you to select files, set TraceTuner processing parameters and access the *Viewer* without entering the command-line environment.
- ***TraceTuner Viewer –*** The *Viewer* is a Java application used to view:

- Traces (both raw and analyzed data)
- Original (usually ABI) base calls
- TraceTuner base calls and quality values
- TraceTuner alternative base calls and quality values
- Alignments between TraceTuner base calls and user-specified consensus sequences

- ***TraceTuner command-line program –*** This component provides access to all algorithms and data processing software and is run at an MS-DOS prompt or from the UNIX/LINUX/IRIX command line. This component can be used instead of the *Launcher* and the *Viewer*. Using TraceTuner from the command line is described in detail in Chapter 5, *Command Line Usage*.

Typographic Conventions

- Required elements. Each element that requires a choice of parameters is shown with curly braces "{ }".
- Optional elements are shown with square brackets "[ ]".
- Alternatives are separated by the pipe symbol "|".
- Commands are shown in `courier`.
- Variables are shown in angle brackets "< >" and *italics*.
- Line continuations are indicated by a backslash "\". Thus, all of the following would comprise a single record:

```
java -cp ttuner_tools.jar com.paracel.\
tt.run.TTrun
```

CHAPTER 2

Launcher

The TraceTuner *Launcher* is a Java user interface that lets you run TraceTuner without entering the command-line environment (see Figure 2-1).

To initialize the TraceTuner Launcher in UNIX/LINUX/IRIX, enter the command:

```
java -jar ttuner_tools.jar
```

or

```
java -cp ttuner_tools.jar com.paracel. tt.run.TTrun
```

Note: Java 1.3.0 or later is required. To check the version of Java installed, enter this command:

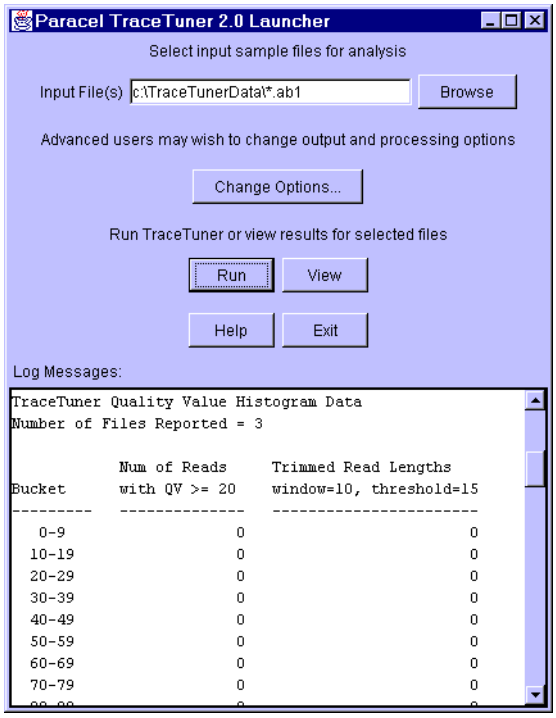
```
java -version
```

To start the TraceTuner Launcher in Windows, select *Start* > *Programs* >*Paracel*>*TraceTuner* > *TraceTuner Launcher.*

Launcher Main Window

If you know the entire pathname for the directory containing the sample files you wish to analyze, enter the pathname directly into the text entry field at the top of the *Launcher* main window. Alternatively, use the Browse function to locate the directory.

FIGURE 2-1: Launcher Main Window



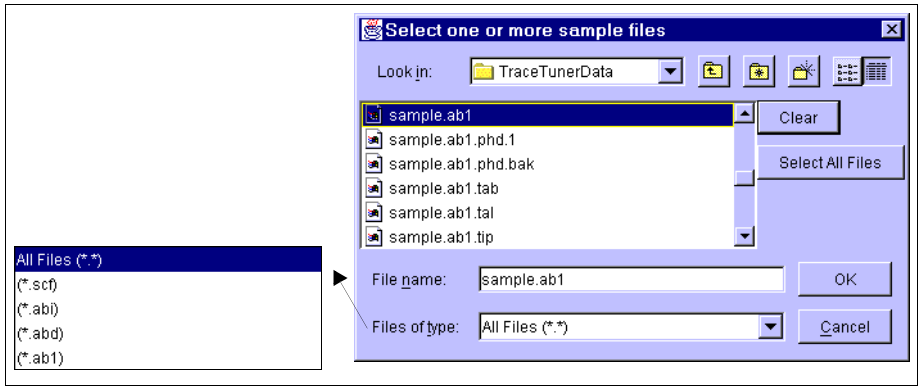
A typical Browse dialog box is shown in Figure 2-2. By default, the Browse dialog displays all files. TraceTuner accepts multiple input file types for a given run. Use the *Files of type* filter to limit the display to *.ab1, *.scf, *.abd, or *.abi files.

The usual file selection options apply:

- Click to select a single file
- <shift> click to select multiple contiguous files
- <ctrl> click to deselect a file or select multiple separate files

Click OK to load the selected files and return to the Launcher main window.

FIGURE 2-2: Browse Dialog Box



Selected files are shown in the *Input File(s)* textbox of the Launcher main window (Figure 2-1) as follows:

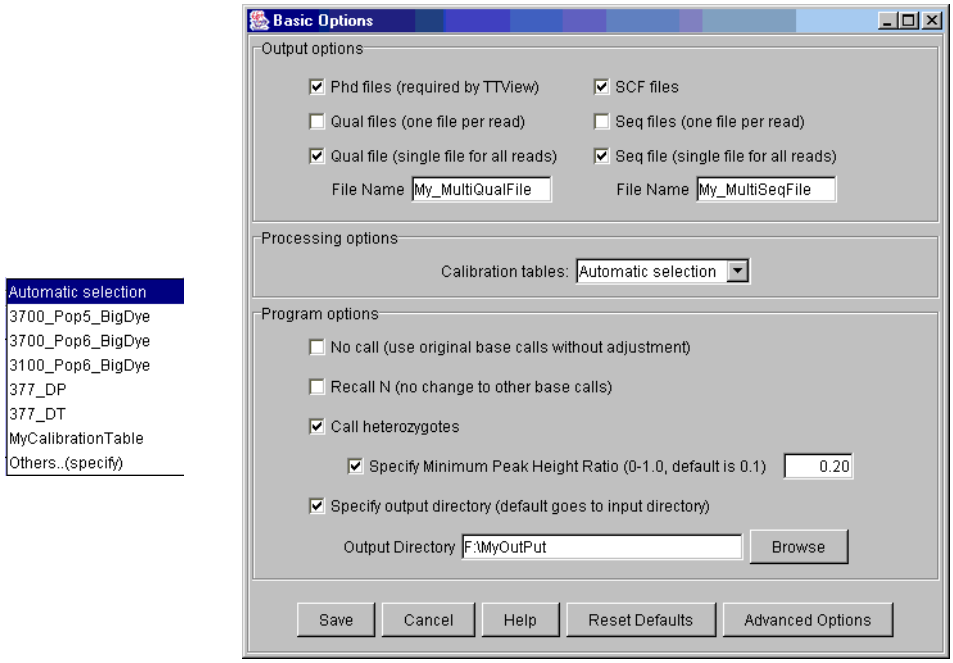
- A *single* file appears in *Input File(s)* as *directory*/*filename*
- If *multiple* *.ab1, *.scf, *.abd, or *.abi files have been selected, they appear as *directory*/<list>.ext, where .ext is typically .ab1, .scf, .abd, or .abi. However, it is not mandatory that input filenames have extensions.
- If *all* files of a type have been selected, they appear as *directory*/*.ext.
- If *multiple* files of differing types have been selected, they appear as *directory*/<list>.

Launcher Options

TraceTuner parameters are divided into two categories: *Basic Options* and *Advanced Options*. These parameters can be set in the *Launcher Basic Options* and the *Launcher Advanced Options* windows. To access the *Launcher Basic Options* window, click CHANGE OPTIONS in the *Launcher Main Window*.

Basic Options

FIGURE 2-3: Launcher Basic Options



Important Note: The *Basic Options* window is *non-modal*, meaning that it is not necessary to exit the window in order to run the application and view the results. You may find it helpful to leave the *Basic Options* window open to review which options you have selected while viewing the TraceTuner output.

Basic Output Options

Six basic file output options are available:

- `.phd` **format files –** `Phd`-format output files are named by appending `.phd.1` to the input file names. A `.phd` output file consists of a comment section containing a synopsis of the program parameters and a results section listing the revised base calls, assigned quality values and peak locations. For an example of a `.phd` output file, see *PHD File Format* on page 4-4.

The .phd format is the preselected default output for TraceTuner. You should leave the .phd output file option selected, as the .phd.1 file is required by the TraceTuner Viewer.

- .scf **format files –** Scf-format binary output files are named by replacing the .ab1 extension of the input files with the .scf extension. If an input file is a .scf file or has no extension, the output file is named by appending a second .scf to the filename. For details about .scf files, see *SCF File Format* on page 4-7.
- **FastA format** .qual **files (one file per read) –** Qual quality value files are named by appending .qual to the input filenames. This option specifies that a separate .qual file be generated for each input file.
- **FastA format** .qual **file (single file for all reads) –** The output file is a .qual file with a user-specified name containing quality values of *all* the reads processed in this run. Specify the output filename in the field that appears when you select this option. The output file is saved in the directory specified in the *Output Directory* field (see Figure 2-3). If no directory is specified, the file will be saved to the same directory as the sample file(s).
- **FastA format** .seq **files (one file per read) –** The output is a FastA-formatted sequence file named by appending the .seq extension to the input filename. A separate .seq file is generated for each input file.
- **FastA format** .seq **files (single file for all reads) –** The output is a sequence output file with a user-specified name containing the base calls of *all* the reads processed in this run. Specify the filename in the field that appears when you select this option. The output file is saved in the directory specified in the *Specify Output Directory* option (see Figure 2-3). If no directory is specified, the files will be saved to the same directory as the sample file(s).
- .Poly **files –** This output is currently available only from the command line and is invoked using the -d option with the ttuner executable. The output directory is specified using the -dd <*dir*> option. The output directory is specified using the -dd <*dir*> option. If no directory is specified, the files will be saved to the same directory as the sample file(s). This output format is compatible with programs that accept Phred/Phrap output files.

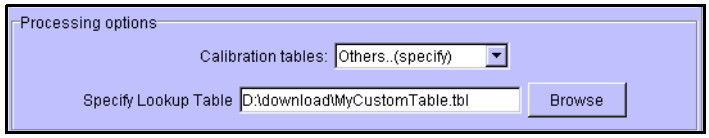
Multiple output formats may be selected at one time. Output formats are described in greater detail in Chapter 4, *Example Output*.

Basic Processing Options

The *Calibration Tables* pulldown menu contains the currently available calibration tables (Figure 2-3). The standard options are:

- **Automatic selection –** This default setting instructs TraceTuner to select from among the built-in calibration tables. TraceTuner makes the selection based on the chemistry used to generate the sample file. Selecting any other option automatically turns off automatic selection.
- **ABI3700_Pop5_BigDye –** Built-in calibration table: if the type of chemistry cannot be determined from the sample file, this table is used by default.
- **ABI3700_Pop6_BigDye –** Built-in calibration table
- **ABI3100_Pop6_BigDye –** Built-in calibration table
- **ABI377_DT –** (Dye Terminator) – Built-in calibration table
- **ABI377_DP –** (Dye Primer) – Built-in calibration table
- **Others –** Allows you to use a lookup table not listed in the pulldown menu. Selecting this option opens a text entry field and an accompanying browse feature (Figure 2-4). TraceTuner will use this user-specified look-up table during the current and future sessions until you specify one of the other options or select a different look-up table.

FIGURE 2-4: User-Specified Calibration Table



To add a custom lookup table to the pulldown menu permanently so that it is available for use in all future sessions, move the lookup table file into the same directory as the TraceTuner executable file (…*TraceTuner/ttuner.exe*). The filename must end in `.tbl`. After doing this, you must close the *Basic Options* window and reopen it in order for the table to appear in the pulldown menu.

Important Note: The calibration table must be compatible with the current version of TraceTuner: both the calibration table and TraceTuner must have the same `x.x` version number. For example, TraceTuner 2.0 is compatible with calibration tables that begin with 2.0. A calibration table version 2.0.x will also be compatible with TraceTuner version 2.0, but a calibration table version 1.1.x or 1.2.x will not be.

Basic Program Options

Important Note: The *No Call*, *Recall N* and *Call Heterozygous Bases* options are mutually exclusive. The default setting for all three options is 'deselected'.

- **No Call Option –** Does not edit or improve the original base calls in the sample file, but simply assigns quality values to the original base calls.
- **Recall N –** Recalls only bases that were designated as N. All other bases remain as they were originally called, but are relocated to the positions of corresponding peaks.
- **Call Heterozygous Bases –** Automatically detects heterozygous bases while editing and improving the original base calls in the sample file. See Table 3-1 for a tabulation of the IUB codes used to designate heterozygous bases. Quality values are assigned to all the bases that are called.
  - **Specify Minimum Peak Height Ratio –** Uses the specified value as the threshold ratio of the shorter peak height to the higher peak height when calling heterozygous bases. If the ratio is less than the specified value, the shorter peak is considered as noise, and the higher peak is called as the "pure" base.

    The default ratio is 0.1. To enter a non-default value, select the option and enter the value in the entry field (see Figure 2-3). Valid values range from 0 to 1.0. This option is available only when the *Call Heterozygous Bases* option is selected. If you check this box, you must enter a value.
- **Specify an Output Directory Option –** The default is to write output files to the same directory as the sample files. To save output to a different location, select the option and enter the pathname in the entry field (see Figure 2-3) or use the Browse option to select a directory. If you check this box, you must supply a directory pathname. If you specify an alternative output directory, the specification remains in effect for future TraceTuner sessions until you change the specification.

Basic Option Management

The following options are available for managing the settings made in the *Basic Options* window:

- SAVE – Saves the specified options and exits the window. The saved options remain in effect until new options have been selected and resaved.
- CANCEL – Cancels any selection made since the last SAVE.

- HELP – Accesses on-line *Help* for the *Basic Options* window.
- RESET DEFAULTS – Resets the basic options to their default values.
- ADVANCED OPTIONS – Accesses the *Advanced Options* window.

Advanced Options

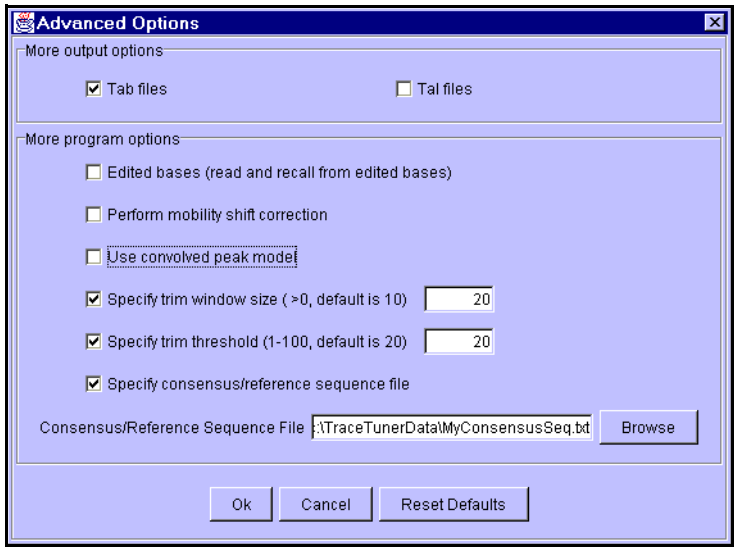
The *Advanced Options* window is accessed from the *Basic Options* window.

Advanced Output Options

There are two advanced output options available for creating `.tab` and `.tal` files. If you rerun a set of sample files and deselect any of these options while allowing the output files to be saved to the same directory as the earlier run, the deselected output files will display a "stale" status due to the mismatch in creation times.

- **TAB files –** Select this option to create `.tab` files. The `.tab` files contain TraceTuner alternative bases. Only the highest alternative bases are displayed by the *Viewer*. For a definition of Alternative Bases, see the Glossary. For details on viewing alternative bases, see page 3-10. See *TAB File Format* on page 4-7 for an excerpt of a `.tab` file.
- **TAL files –** Select this option to create `.tal` files. These files contain alignment data that TraceTuner generates when checks the TraceTuner base calls against a user-specified reference sequence. In order for TraceTuner to create `.tal` files for the selected sample files, it is necessary to supply a FastA-formatted reference sequence file. For details on viewing alignments, see page 3-9. See *SEQ File Format* on page 4-5 for an excerpt of a typical FastA file.

FIGURE 2-5: Launcher Advanced Options



Advanced Program Options

- **Edited Bases –** Reads the *edited* bases and locations from the sample files and starts from these when recalling bases. By default, TraceTuner reads and starts from *called* bases and locations. For the distinction between edited and called bases, consult the Glossary.
- **Perform mobility shift correction –** Attempts to make TraceTuner space base call positions evenly.
- **Use convolved peak model –** Specifies that TraceTuner calculate peak shape using a convolved model; this model assumes that the initial distribution of DNA fragments in the electrophoretic cell has non-zero width. The simple gaussian peak model (the default) assumes zero width for this initial distribution.
- **Specify trim window size –** Specifies a non-default value for the *trim window size*. By default, TraceTuner uses a moving average of 10 bases to trim the beginning and the end of a sequence. The *trim window size* is the number of bases used in calculating the moving average. The trimming stops when the average quality value of bases in the window reaches the *trim threshold value*. To specify a non-default value for the trim window size, check this option and enter the desired

value (Figure 2-5). Valid values are any positive integer. See page 3-10 for details on viewing trimmed bases. Values for trimming boundaries are stored in the `.phd.1` output file.

- **Specify trim threshold –** Specifies a non-default quality value for the *trim threshold* used to determine where sequence ends should be trimmed. The default value for this parameter is 20. This option can be used to specify the threshold for the average quality value of bases in the moving average calculation discussed immediately above. To specify a non-default value for the trim threshold, check this option and enter the desired value (Figure 2-5). Valid values range from 0 to 100.
- **Specify consensus/reference sequence file –** Specifies a FastA-formatted consensus sequence against which the bases called by TraceTuner can be aligned using the Smith-Waterman alignment algorithm. To specify a consensus/reference sequence file, check this option and enter the pathname of the file or use the Browse feature (Figure 2-5). See page 3-9 for details on viewing the alignment between TraceTuner base calls and a consensus sequence. Alignment data are stored in the `.tal` output file.

Important Note: The *Advanced Options* window is *modal*, thus it must be closed in order to proceed with the application.

Running TraceTuner

Run Options

Run/Stop

Once you have set all desired parameters, click RUN to process the input files. While TraceTuner is processing the input files, the RUN button changes into a STOP button, thus making it possible to interrupt the run for any reason.

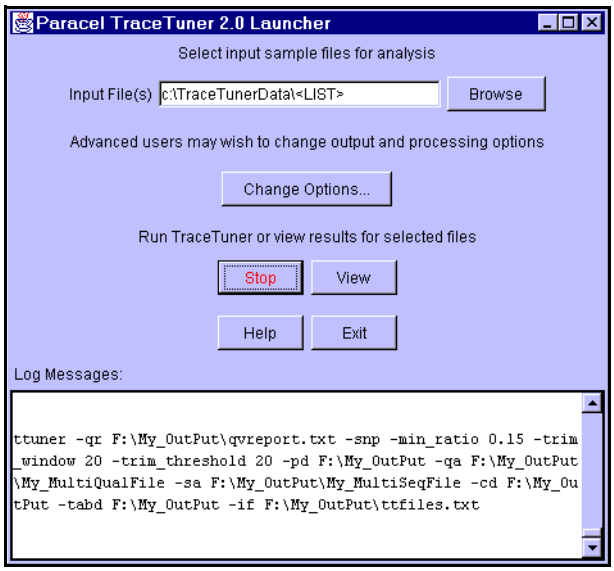
Important Note: If you have not specified an output directory, output files are saved by default to the input directory. If files are about to be overwritten during this process, a warning will appear.

View

It is not necessary to wait for TraceTuner to finish processing all sample files before viewing results.

Important Note: By default, when using the *Launcher*, the quality values and the adjusted base calls are written into `.phd.1` files with the same root name and in the same folder as the original sample files. The original sample files are not altered by TraceTuner. While TraceTuner is processing the sample files, status messages will appear in the *Log Messages* window (see Figure 2-6). Also, The log messages are also recorded in the file `ttlog.txt`, which resides in the output directory, so the whole directory can be archived as a record of the session.

FIGURE 2-6: Launcher Main Window during Run



Help

The HELP button accesses context-sensitive on-line help.

Exit

Press EXIT to quit the *Launcher*, terminate any currently running TraceTuner jobs and close the application.

MS-DOS Prompt

When the TraceTuner *Launcher* is started on a PC, an MS-DOS prompt appears briefly in addition to the *Launcher* window. This prompt displays messages from the Java environment running the *Launcher* and the *Viewer*. Normally, this prompt may be ignored.

Log Files

TraceTuner generates two log files of interest to the user. The first is `ttlog.txt`, which contains the same messages that appear in the *Launcher* message window. Messages for each subsequent run during a session are appended to this file which resides in the output directory. A new `ttlog.txt` log file is generated at the beginning of each session. Thus, if new results are written to the same output directory as in a previous session, the earlier file will be overwritten.

A second log file, `ttstderr.log`, resides is in the same directory as the TraceTuner executable. Error messages and diagnostics for the Java interface are appended to this file in each subsequent session, thus providing a history of TraceTuner program execution. If you encounter problems when executing TraceTuner, you may be requested to send this file to Paracel for analysis.

If the `ttuner` executable is run from command line, all output (including error messages) will be dumped to the screen.

CHAPTER 3

Viewer

Overview

The *Viewer* is used to view the trace information from the original sample file, revised base calls, including heterozygous base calls, and quality values from the `.phd` files produced by TraceTuner.

TraceTuner 2.0.9 can also generate two advanced output formats, namely `.tab` and `.tal` files. These files contain information on alternative base calls and sequence alignments, respectively. The data in these files can also be viewed with the TraceTuner 2.0.9 Viewer.

While TraceTuner processes the sample files, messages are displayed in the *Log messages* window (Figure 2-1) and are appended to the `ttlog.txt` file.

You can view program results at any time by clicking VIEW in the *Launcher* main window. You do not have to wait until TraceTuner has processed all sample files in the run to view the currently available results.

You can also bring up a standalone Viewer by using the following command:

```
java -cp ttuner_tools.jar com.paracel.tt.\
   run.TTView <samplefile> [<phd.file>]
```

If the Viewer is launched in this way, it will view only one file set at a time. If the <phd_file> is not specified, the Viewer will look for it in the <sample_file>

directory. The Viewer will also pick up the auxiliary files (`.tal`, `.tab`) if they are present in the <`phd_file`> directory. Note also that Java 1.3.0 or later is required.

Important Note: The Viewer reads *file sets*, meaning the original `.ab1` or `.scf` sample file and the main `.phd` output file, plus any additional `.tab` and `.tal` output files. The Viewer does not read UNIX-compressed or gzipped files. TraceTuner needs file sets (minimally the sample file and the `.phd` output file) in order to construct the display. If you specify an alternative output directory when you process a set of sample file(s), you must point TraceTuner to that same output directory again in order to review the results of that run. Specifying an alternative output directory is discussed in *Basic Program Options* on page 2-7.

Viewer Main Window

It is possible to have multiple *Viewer* windows open at the same time. The number in the title bar of a Find dialog (see page 3-7) corresponds to the number in the title bar of the Viewer window from which the Find dialog was opened. If the Viewer window is numbered as [3], the Find dialog opened from it will always be numbered as [3], no matter how many times the user closes and re-opens it.

Base Color Codes

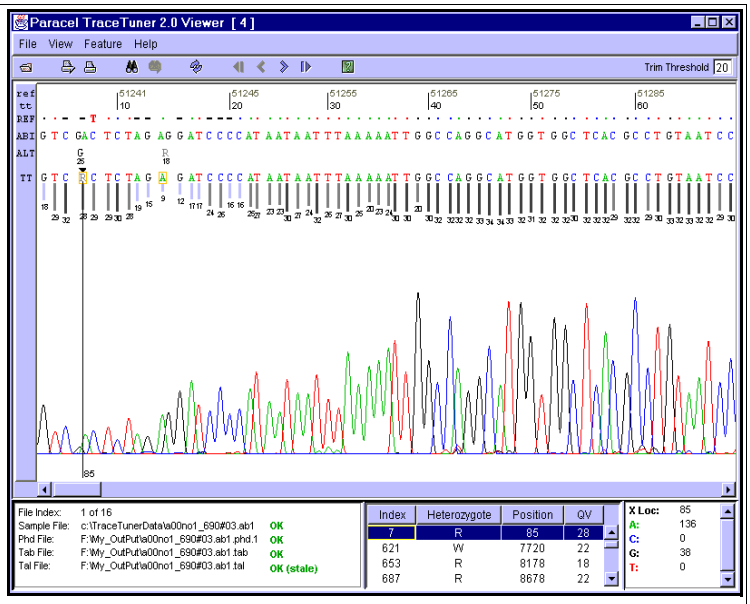
The different bases are color coded in the main *Viewer* window:

- Adenine -> green
- Cytosine -> blue
- Guanine -> black
- Thymine -> red

Original base calls may contain some ‘N’s. These are used to represent peaks for undetermined bases. N’s are shown in light gray.

FIGURE 3-1: TraceTuner Viewer Main Window

1. Right panel legend:
   - ph Phred base call index
   - ref Reference seq. index
   - tt TraceTuner index
   - REF Reference seq.
   - ABI Original base calls
   - ALT Alternative base calls
   - TT TraceTuner base calls
   - PH Phred base calls
2. File index always displayed (lower left).
3. Select *View > List detected heterozygotes* to display listing (lower center): double-click on an entry to show heterozygote call (indicated by a ▼ mark above TT calls); click on a heading to sort calls by index number, heterozygote code, etc.
4. Select *View > Show signal value* to display signal values (lower right): click anywhere in main window to display scan position (x *loc*) and base signal values.
5. Select *Feature > Display original (ABI) base calls* to view original calls.
6. Double-click in a trace curve to locate the base peak (indicated here by the vertical line at 85).



File Status Messages

In addition to the file index number of the read and the pathnames for the sample and output files, TraceTuner can display a number of status messages for the files. These include:

- OK – Meaning varies with type of file:
  - For sample files, generally indicates that file was read and validated without difficulty;
  - For `.tal` files, alignment was produced, no repeats found.
- OK (Stale) – The output file was created at an earlier time than the other output files
- Cannot read – user does not have the permission to read the file
- Invalid data – typically indicates that missing header line(s) or invalid data in header lines

- No good alignments found – for `.tal` files only
- Possible repeats – for `.tal` files only
- Invalid alignment data – for `.tal` files only
- No alternative base calls – for `.tab` files only
- Invalid alternative base call data – for `.tab` files only
- Invalid for unknown reason
- Does not exist

TraceTuner base calls may contain *heterozygous bases* (namely, *M*, *R*, *W*, *S*, *Y* and/or *K*). Heterozygous bases are called and visualized using the pertinent IUB nucleotide codes for base combinations.

TABLE 3-1: IUB Nucleotide Codes

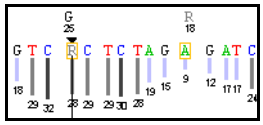
IUB Code	Matches
M	A and C
R	A and G
W	A and T
S	C and G
Y	C and T
K	G and T

Heterozygous bases are also shown in light gray. The letter designations and the electropherogram both utilize this color coding scheme.

Quality Values

Quality values are shown as bars below the letter designations for the bases. The color of the bar indicates the quality value range: light gray indicates QV < 20, gray indicates 20 ≤ QV < 30, and dark gray indicates QV ≥ 30. Numerical quality values are shown below the bars. A quality value of 10 is usually considered poor, since it corresponds to a predicted probability of base calling error of 10%, whereas 20 is seen as good quality (1% error), and 30 or above is seen as excellent (≤ 0.1% error).

FIGURE 3-2: Quality Values



Quality values are related to base call error probabilities by the formula:

$$QV = -10 \log_{10}(P_e) \quad \textbf{(3-1)}$$

where

P_e is the probability that the base call is an error. Thus, a 1/1000 (0.1%) probability that the base call is in error would yield a quality value of $-10 \cdot (-3) = 30$.

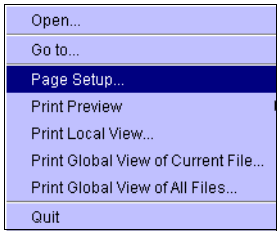
Base Call Alignment

Original base calls and TraceTuner base calls frequently do not line up. This occurs because TraceTuner can add, delete or reposition base calls during processing. TraceTuner reassesses the electropherogram, adding base calls where the peaks calculated by the model meet its criteria and deleting base calls where they do not.

File Index

The index number of the sample file, its name and status, plus name and status of all corresponding output files currently loaded in the *Viewer* are shown in the subwindow at the lower left of the *Viewer* main window. By clicking on the arrows in the second tier of buttons above the *Viewer* main window, you can navigate through the batch of files currently in the *Viewer*.

Viewer Menu Options



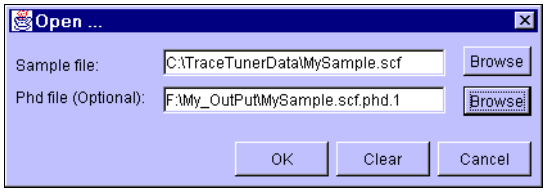
File Menu

Many of the *File* menu options on the *Viewer* window toolbar are self-evident and require little discussion. However, several functions are peculiar to TraceTuner:

- **Open –** accesses a dialog box that you can use to reload a *single* sample file / output file set (Figure 3-3). If the sample file directory also contains the output files (TraceTuner default), you need only load the `.ab1` or `.scf` sample file and the *Viewer* will also read the `.phd` file and the auxiliary `.tab` and `.tal` files from the same directory, if they are present. If the sample file and the `.phd` output file reside in different directories, the *Viewer* will also read the auxiliary files from

the directory where the `.phd` file resides. When the file set has been loaded, the usual name and status information for the files is displayed in the file index sub-window in the *Viewer* window.

FIGURE 3-3: View File Open Dialog



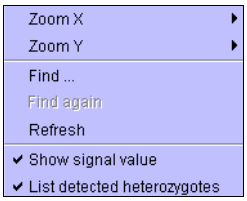
If the sample file or the `.phd` file is invalid, no trace information or called bases will be displayed in the *Viewer*.

Remember that only one file set may be opened at a time using this method.

- **Go to –** allows you to enter the index number of a sample file and to jump directly to that file set in the *Viewer*. This option facilitates navigation when large numbers of files have been loaded into the Viewer.
- **Page Setup –** displays a standard printer page setup dialog. You can change any of the available options. Landscape orientation is recommended and is set as the default.
- **Print Preview –** two Print Preview options are available:
  - **Print Local View –** Displays a thumbnail preview of the part of the trace that is currently displayed in the *Viewer*. The default magnification of the thumbnail is 10%. You can change the magnification by selecting one of the preset values from the pulldown menu or you can enter a value in the text field and press <Enter>. The printed page will contain the current display at full size.
  - **Print Global View of Current File –** Displays thumbnails of the entire file currently loaded in the *Viewer*. Again, the default magnification of the thumbnail preview is 10% and can be changed as described above. The preview page(s) show the entire trace. Each printed page will contain multiple rows of the electropherogram as specified in *Page Setup*.
- **Print Local View –** Prints the part of the trace that is currently displayed in the *Viewer* at full size without previewing the trace.

- **Print Global View of Current File –** Prints the entire trace displayed in the active instance of the *Viewer* without previewing the trace. Each page contains multiple rows of the electropherogram as specified in *Page Setup*.
- **Print Global View of All Files –** Prints entire traces of all files currently loaded in the *Viewer*. Each page contains multiple rows of the electropherogram as specified in *Page Setup*. This print option does not support printing to a non-system default printer or to a file.
- **Quit –** closes the current instance of the *Viewer*.

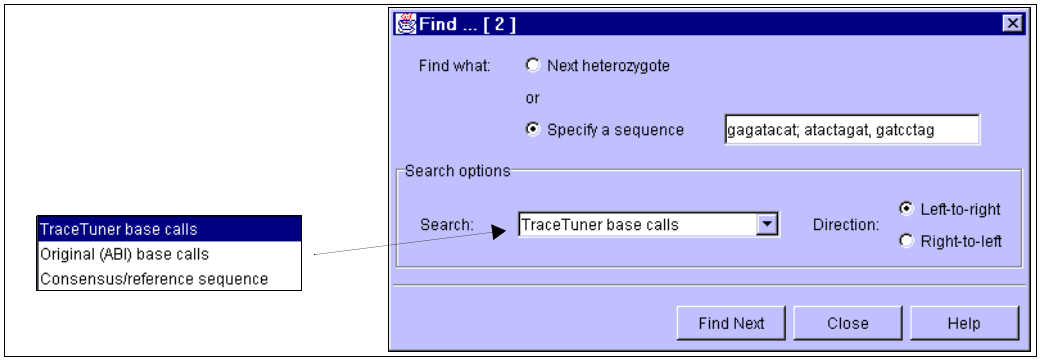
View Menu



Many of the *View* menu options on the Viewer window toolbar are self-evident and require little discussion.

- **Zoom x –** Adjusts the horizontal magnification in the range of 5 to 5000%. Default is 100%. Values other than those given in pull-down menu can be entered in the text field.
- **Zoom y –** Identical to *Zoom x* but in the vertical direction.
- **Find –** Offers a number of useful search functions (Figure 3-4).

FIGURE 3-4: Viewer Search Functions



- **Find what –** Specifies whether to locate the next heterozygote or to locate occurrences of one or more base sequences in the specified file. When entering multiple sequences, separate them with whitespace, commas, or semicolons.

  Note that the *Find* feature performs *case-insensitive* literal matching.

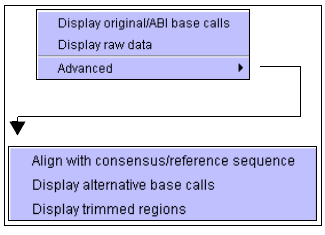
- **Search options –** Specifies which bases to search (TraceTuner base calls, original base calls or consensus/reference sequence) and the direction of the search.

- **Find again –** Performs the same function as the FIND NEXT feature in the *Find…* dialog box.
- **Show signal values –** Adds an untitled window at the bottom right of the *Viewer.* When you click on or near the traces, this subwindow displays:
  - the scan position where you clicked
  - the corresponding signal values at the scan position
- **List detected heterozygotes –** Adds an untitled window at the bottom center of the *Viewer*; this subwindow displays:
  - **Index –** The base index number for the heterozygote base call
  - **Heterozygote –** The IUB nucleotide code for the heterozygote base call; see Table 3-1 for a listing of the IUB codes for nucleotide combinations
  - **Position –** The scan position (*x-loc*) of the heterozygote base call
  - **QV –** The quality value for the heterozygote base call.

  Double-clicking on any entry in this listing takes you directly to the heterozygote base call. Note that you can sort heterozygote base calls by any of the heading categories by simply clicking on the heading.

Feature Menu

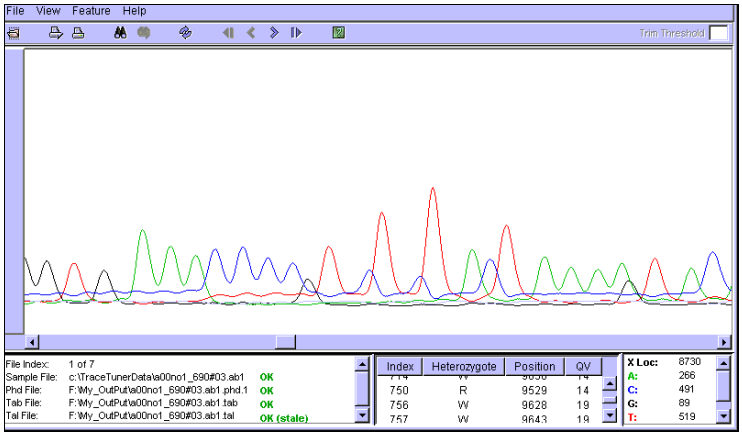
The Feature menu contains options for displaying the original (ABI) base calls, raw data, or other advanced display options discussed in detail below.



- **Display original (ABI) base calls –** Displays the original base calls above the TraceTuner base calls. Note that this feature is disabled if the *Display raw data* option is selected, since the original base calls are pertinent only to analyzed data.
- **Display raw data –** Displays the raw data from the sample file. Note that selecting this feature automatically disables all *Advanced…* options in the side menu since the latter are derived from analyzed data and are irrelevant to raw data; moreover no trace information is displayed at the top of the Viewer (Figure 3-5).

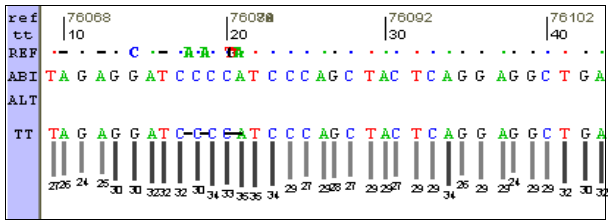
  Note: When the input to TraceTuner is a `.scf` input file, the *Display raw data* option is disabled because this file format contains no raw data.

FIGURE 3-5: Raw Data Display from .ab1 File



- **Advanced –** Accesses a side menu with the following options:
  - **Align with consensus/reference sequence –** Displays the best alignment region between the TraceTuner base calls and the consensus/reference sequence.

FIGURE 3-6: Consensus/Reference Sequence Alignment



The best-aligned consensus/reference sequence is shown beneath the index marks with color-coded dots for the consensus/reference bases that match the TraceTuner called bases literally, black dashes for the alignment gaps, and color-coded base codes for the literal mismatches. The index marks of the consensus/reference sequence are shown above the TraceTuner base call index marks.

The alignment data are stored in the `.tal` file. The consensus/reference file must be in FastA format.

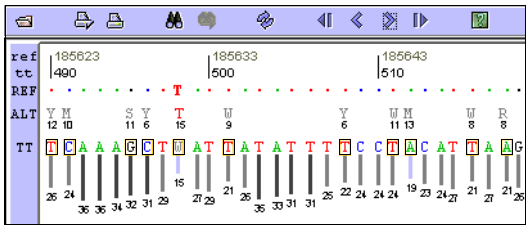
This feature will be disabled if:

- a `.tal` file does not exist for the sample file;

- possible repeats or no good alignments are found;
- the *Display raw data* option has been selected.

- **Display alternative base calls –** Displays, for each called base, the alternative base call with the second highest quality value and the quality value from the `.tab` file above the corresponding TraceTuner base call. The third highest and all other alternative base calls are not displayed. The TraceTuner base calls appear boxed in orange.

FIGURE 3-7: Alternative Base Calls

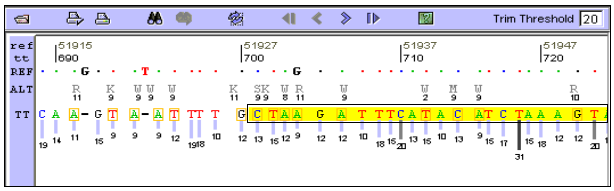


This feature is disabled if:

- there is no `.tab` file corresponding to the sample file,
- the `.tab` file contains no TraceTuner alternative base calls, or
- the *Display raw data* option has been selected.

- **Display trimmed regions –** Displays the trimmed/low quality regions of the TraceTuner base calls on a yellow background. The specified trim threshold value is shown at the right end of the *Viewer* window toolbar.

FIGURE 3-8: Trimmed Base Region



The trim data are stored in the `.phd` file.

This feature will be disabled if:

- the `.phd` file does not exist,
- no valid trim data are found in the `.phd` file, or

- the *Display raw data* option has been selected.

A *single* click on or near the traces displays the scan position (*X-Loc*) and corresponding signal values in the *Signal Values* window in the lower right of the *Viewer* (see Figures 3-1and 3-5). A *double click* on or near the traces or on the TT base calls locates the exact peak location of nearest TT base call, marking the location with a vertical line that passes through the called base, the quality value bar and the peak of the trace. The peak location is indicated at the bottom of the line. (Note the scan position “85” at the bottom of the *Viewer* in Figure 3-1.) If ABI called bases are displayed, one can look at the ABI peak locations by double-clicking at the ABI bases. Similarly one can also look at the TraceTuner alternative base call (TAB) locations by double-clicking at the alternative base calls, if they are displayed. This is the peak location.

Help

Context sensitive on-line help can be obtained at any time by clicking HELP .

This page intentionally left blank.

CHAPTER 4

Example Output

Overview

This chapter includes examples of TraceTuner input and output files:

- *Quality Value Report*
- *PHD File Format*
- *QUAL File Format*
- *SEQ File Format*
- *SCF File Format*
- *TAB File Format*
- *TAL File Format*

Quality Value Report

When running TraceTuner with multiple sample files, TraceTuner generates a *Quality Value Report*. From the command line, this report is generated by using the `-qr` <*output_file*> option. When running the TraceTuner from the *Launcher* Java user

interface, the report is automatically generated whenever multiple sample files are submitted in a run. The report is displayed in the *Launcher Log Messages* window and is appended to the log messages file `ttlog.txt`. The most recent version is also saved in the output directory as a separate file `qvreport.txt`.

The Quality Value Report is a concise way to demonstrate the distribution of the length of the reads in a group of samples, where the read length is defined as the number of bases with a QV≥ 20 in a sample. The report also lists the distribution of files by length after they have been trimmed in accordance with the moving average QV criteria discussed in *Advanced Program Options* on page 2-9. After TraceTuner determines the trimmed lengths of the samples, it uses the length of the longest file to create a number of *buckets*, each ten bases in width. The report shows how many reads with a QV≥ 20 fall into each bucket. Bucket "width" and QV threshold used for distributing the reads into the buckets are set at 10 bases and QV=20, respectively, and cannot be changed by the user. By contrast, trim "window" width and trim threshold value are user-modifiable.

Consider the following case where 8 sample files ranging from 820 to 890 bases in length were submitted in the run.

```
TraceTuner Quality Value Histogram Data
Number of Files Reported = 8

            Num of Reads     Trimmed Read Lengths
Bucket      with QV >= 20    window=15, threshold=15
---------   --------------   -----------------------
   0-9                  0                         0
  10-19                 0                         0
  20-29                 0                         0
  30-39                 0                         0
  40-49                 0                         0
  50-59                 0                         0
... (multiple lines omitted for sake of brevity)
 600-609                2                         0
 610-619                0                         0
 620-629                0                         0
 630-639                1                         0
 640-649                2                         0
 650-659                0                         0
 660-669                0                         0
 670-679                1                         0
 680-689                2                         0
 690-699                0                         0
... (multiple lines omitted for sake of brevity)

 740-749                0                         0
 750-759                0                         0
 760-769                0                         2
 770-779                0                         2
 780-789                0                         1
 790-799                0                         3

Average               644
```

Of the eight sample files, two contained between 600 and 609 bases with QV≥ 20, one fell into the 630–639 bucket, two into the 640–649 bucket, etc. The average number of bases with QV≥ 20 was 644.

Using the specified criteria, three files were trimmed down to 790–799 bases in length, one to 780–789, two to 770–779, and so forth.

PHD File Format

Files in `.phd` format are the default output mode for TraceTuner. These files include a header with data describing the output along with the revised base calls, assigned quality values and peak locations. Base calls include simple nucleotides plus heterozygotes. An abbreviated sample of a `.phd` file is provided.

```
BEGIN_SEQUENCE A02_005.ab1
BEGIN_COMMENT
CHROMAT_FILE: A02_005.ab1
ABI_THUMBPRINT: 0
PHRED_VERSION: TT_2.0
CALL_METHOD: ttuner
QUALITY_LEVELS: 44
TRACE_ARRAY_MIN_INDEX: 0
TRACE_ARRAY_MAX_INDEX: 12923
TRIM: 40 789 0.010000 \\trim boundaries; error probability
CHEM: term
DYE: big
END_COMMENT

BEGIN_DNA
g 13 25
c 12 38
...
w 10 9443
a 21 9455
m 14 9466
g 11 9484
c 15 9497
c 15 9513
a 12 9527
c 14 9545
m 14 9556
...
a 9 12910
c 9 12911
END_DNA
END_SEQUENCE
```

QUAL File Format

A Quality Value output file begins with a header line marked with a right angle bracket: “>”. The `.qual` file header line contains the sample filename, the number of bases in the file (in this case, 800), the index of the first high quality base (8), and the length of the high quality region (792). The rest of the file consists of the quality values of the nucleotide calls in the sequence. The values are separated by whitespace. The example below is the `.qual` file corresponding to the FastA `.seq` file shown on page 4-5.

```
>Bnel22_E07_Bnel22_051.ab1 800 8 792
  7 10 11 13  9 11  8  7 11 13 23 23 23 25 25 23 25
 25 23 28 27 22 24 31 31 31 33 35 27 23 28 28 25 24
 25 32 31 33 31 27 27 31 33 30 28 39 33 33 33 33 33
 33 33 31 27 28 28 31 31 30 28 31 33 34 35 33 33 31
 29 34 29 27 33 33 33 33 20 21 17 17 22 24 26 29 29
 28 33 24 26 26 27 28 29 28 27 36 33 33 29 30 33 27
 27 28 21 25 21 21 22 24 33 30 32 35 31 32 32 34 35
 33 19 27 22 22 25 27 27 35 30 30 29 23 29 29 27 33
 29 33 26 31 31 31 31 33 32 32 30 30 29 29 21 22 37
 37 37 37 37 37 34 33 33 31 33 31 28 30 28 31 35 34
 37 33 29 31 30 33 29 24 11  9 23 16 28 33 29 34 36
 .
 (multiple lines omitted for sake of brevity)
 .
  25 28 25 30 29 24 25 30 20 35 29 28 22 31 28 21 21
 18 10 25 25 15 16 12 25 17 32 26 12 27 18 21 28 25
 25 23 28 22 26 14 31 23 28 28 13 25 15 28 25 28 28
 30 15 27 19 19 14 15 10 19 16 11 21 16  6 20 25 22
 22 26 33 28 22 31 15 22 10 24 16 28 34 23 30 29 28
 24 15 31 12 31 27 10 29 27  9 15 13 10 13 24 21 26
 17 22 19 31 26 35 37 24 32 32 26 24 29 14 31 24 17
 10 17 11 21 26 18 18 29 12 17 23 27 16 19 15  8 21
 12 16 15 17 10  8 22  8 11 13 17 16 11 14 12  7  8
 20 11  6  8  8 16 14 16 11 31 22 16  8 16  9  6  8
 10  6 25 28 14 10 28 12 18 21 20 28 19 32 27 31 14
 19  8 21 25  7 19 11 17 29 13 19 15 11 24 10 12 11
 15  9 10  8  6  8  7  8 19 21 24 28 33 29 23 13 10
  8 10 12 11 11 23 14 19 15  9 17 25 13 23 12  9 15
 10  9 10 11 11 13 10 12 18 20 24 21 10  6  6  7  7
  6  8  8  8 13 14 14 10 14 15  9 29 22 14 19 16 16
 13 17 15  7  9 12  9 11  8 11  9 10  9 10 11 11 19
 14 16 11 13 10 13 19 10 24 25 32 34 26 12 17 11 14
 14
```

SEQ File Format

A `.seq` file as a FastA file which consists of a header line and base calls, and can contain multiple sequences. Quality value information is not included. In FastA format files, each sequence is indicated by a header string starting with a right angle

bracket: “>”. The header line contains the sample filename and, optionally, other descriptive information. For example, the number of bases in the file (in this case, 800), the index of the first high quality base (8), and the length of the high quality region (792). There are several methods used to parse FastA description lines. A full discussion of these parsing options goes beyond the scope of this manual. The example below is the FastA `.seq` file corresponding to the `.qual` file shown on page 4-5.

```
>Bnel22_E07_Bnel22_051.ab1 800 8 792
TAGCTTGACCATGATTACGCCAAGCTCTAATACGACTCACTATAGGGAAA
GCTGGTACGCCTGCAGGTACCGGTCCGGAATTCCCGGGTCGACCTCGAGC
TCGAGGTGCAGTTTTATTTTGACGCACATAACGTATTGTCATCAGTCCGA
AGAATCTTTTGTAAAGCTGGAAGTTCCACCTCCTGAGAATCAGGATTCCG
GGAGCAGCATCTGCCGGAGTACCTGGCAGATAGCTCTGAGATACCCCTTG
TGCTGCACTTCCTCCGGCAGCTTCAGCATGCTCGCCACCTTCTCGAACTT
CTCCCGCGCCAGGTCCACCTCCGCTGGCGACGCTACGGCGGACACCGAAC
CCGGAACCACCCGGAGGTCCCTATGGAGCTCCGGCGGCGATAGCGCCCAC
TGCATCGCCCAGATGGAGAGGGGCCGCATGTAGTGCAGCGACCGGTACCC
GCCGCCGGCGTCGGACGTCCACGCTTCCGGCGTCTGGAACGCGTACCCGA
ACCCGTCCCTGCCCCACCCGGCGTCGTGCGCTCCCTTCGCCGTCCGGAAC
GCCGCCTCCGTCATGCCCTCGTGGAGCATGGCGGCGGCCACGCCGTAGGT
GACCCCGACCCACACCTCCTTGGACTGCAACGACGACGCGTCCACGGGCG
CCGTCCGGCCGCATCCCGTTCACGGGCCCCACCGCGCCGCCTGACGCGCA
TCACGTTGTAGTCCAGCACCCGTCCCCAGCGCGCTGGAGCCTTTTTCTCC
TTCACGATGGGCTCAGCCCGACGCGCGCGCGTACCACTGGCGGGGAGCTG
```

Spaces, tabs, and carriage returns in the sequence are ignored. If the file contains multiple sequences, the start of a new sequence is indicated by the presence of a new description line.

Consensus/reference sequence data must be submitted to TraceTuner in FastA file format.

SCF File Format

SCF format files are associated with the Staden Package. SCF format files store data from DNA sequencing instruments in binary format, and are thus not human readable. Each file contains the data for a single read and includes its called sequence, its trace sample positions and numerical estimates of the accuracy of each base read. For a general information about the SCF file format, see the Staden Package website at:

http://www.mrc-lmb.cam.ac.uk/pubseq/overview.html

Poly File Format

TraceTuner includes a command-line option to generate output files in the `.poly` file format. This option is provided so that users who are accustomed to using down-stream applications that accept Phred/Phrap output files can easily migrate to TraceTuner. A few lines of a `.poly` file are included below:

```
PDA01RCH12 1.0 1.0 1.0 1.0 1.0
C 6 4753.0 0.744146 N -1 -1.0 -1.0 421.0 518.0 736.0 360.0
C 22 17036.5 4.359584 N -1 -1.0 -1.0 173.0 760.0 209.0 180.0
T 38 9781.0 3.467191 N -1 -1.0 -1.0 203.0 511.0 107.0 442.0
A 52 6762.5 10.478251 N -1 -1.0 -1.0 359.0 312.0 112.0 353.0
.
.
.
```

(In the interest of brevity, superfluous zeros have been truncated from real numbers in the sample.)

TAB File Format

A `.tab` file contains a listing of alternative base call data. This file will be empty unless the basic option *Call heterozygous bases* is selected or the command line option `-het` is used. The second best, or alternative base calls, which may be simple nucleotide base calls and heterozygous base calls, are listed in the `.tab` file. This file is generated when the *TAB file* advanced program option has been selected. TraceTuner uses the information in the `.tab` file when you select *Feature* > *Advanced* > *Display alternative base calls* in the *Viewer.*

Information in a `.tab` file includes:

NUM_ABC	Number of alternative bases; thus the number of lines in the data portion of the file excluding the header
base2	IUB nucleotide code for a given alternative base call; multiple alternatives are possible at a given position
qv2	Quality value of the alternative base
pos2	Location of a given alternative base in terms of scan position
ind2	Order index of the called base

An excerpt of a .tab file is shown below:

```
# CHROMAT_FILE: sample3.ab1
# SOFTWARE_VERSION: TT_2.0alpha
# NUM_ABC: 581
# NUM_SUBSTITUTIONS: 581
# NUM_DELETIONS: 0
# base2   qv2       pos2       ind2
   Y       6          19         1
   C       9          19         1
   M      12          23         2
   C      16          19         2
   W      19          72         5
   M      19         136        10
 (multiple lines omitted for sake of brevity)
   S       9        9336       775
   M       8        9342       776
   S       9        9348       776
   A       9        9333       776
   G       9        9346       776
```

TAL File Format

A `.tal` file contains the local (Smith-Waterman) alignment of the base sequence generated by TraceTuner against a correct consensus/reference sequence. It is used for verification purposes where the correct sequence is known.

The meanings of several of the header items are self-evident and do not require further elaboration. Header information also includes:

`Match` Match premium – default calculation value is +20

`MisMatch` Mismatch penalty – default calculation value is -5

`Insertion` Insertion penalty – default calculation value is -3

`Deletion` Deletion penalty – default calculation value is -3

`RepeatFraction` If TraceTuner finds two or more regions in the consensus/reference sequence that are very similar to the fragment sequence, these regions are interpreted as repeats in the consensus sequence and the alignment is not accepted. “Very similar” means that the fraction of bases matching within the best-alignment region exceeds `RepeatFraction`. The alignment is not produced because it would not be possible to tell which region to use. Default value is 0.90.

`ALIGN_STATUS` `Possible repeats` – similiarity of two or more regions in the consensus sequence to the fragment sequence exceeds `RepeatFraction`

`OK` – alignment was produced; no repeats found

`No good alignments`

`ALIGN_LENGTH` Length of the best-alignment region

```
#CHROMAT_FILE: MySampleFile.ab1
#CONSENSUS_FILE: c:\TTData\MyConsensusSeq.txt
#TRIM: 0 795
#SOFTWARE_VERSION: TT_2.0
#
#Match = 20
#MisMatch = -5
#Insertion = -3
#Deletion = -3
#RepeatFraction = 0.900000
#ALIGN_STATUS: OK
#ALIGN_LENGTH: 867
#    Sample   Cons            is_match
2      G      76062      G      1
3      T      76063      T      1
4      C      76063      -      0
5      A      76063      -      0
6      C      76064      G      0
7      T      76065      T      1
7      -      76066      G      0
8      C      76067      G      0
9      T      76068      T      1
10     A      76068      -      0
11     G      76069      G      1
12     A      76069      -      0
13     G      76070      G      1
.
 (multiple lines omitted for sake of brevity)
.
834    G      76909      G      1
835    T      76910      T      1
836    A      76911      A      1
837    A      76912      A      1
838    A      76913      A      1
839    A      76914      A      1
839    -      76915      C      0
840    C      76916      C      1
841    C      76917      C      1
842    C      76918      C      1
842    -      76919      A      0
843    T      76920      T      1
```

CHAPTER 5

Command Line Usage

As an alternative to using the *Launcher* and *Viewer* described in Chapters 2 and 3, TraceTuner may be run from the MS-DOS or UNIX/LINUX/IRIX command line.

To run TraceTuner from the UNIX/LINUX/IRIX command line, type commands similar to

```
ttuner -pd /mydir/outputdir  -id /mydir/inputdir
```

where *outputdir* and *inputdir* are the TraceTuner output and input directories, respectively.

To run TraceTuner from the PC command line, open an MS-DOS prompt. Change to the directory (*installdirectory*`\tt`) with the TraceTuner executable (`ttuner.exe`) and then enter a command line similar to:

```
ttuner -pd \mydir\outputdir -id \mydir\inputdir
```

Command Line Usage for ttuner Executable

USAGE

```
ttuner
[-h]
[-Q] [-V]
[-nocall]
[-recalln]
[-edited_bases]
[-ipd <dir>]
[-shift]
[-convolved]
[-het]
[-min_ratio <phr>]
[-t <lookup_table>]
[-C <consensus_file>]
[-cv3]
[-pop5][-pop6][-3100][-377dp][-377dt]
[-trim_window <window_size>]
[-trim_threshold <min_average_quality>]
{  -p   | -pd <dir>  |
   -q   | -qd <dir>  | -qa <file> |
   -c   | -cd <dir>  |
   -d   | -dd <dir>  |
   -s   | -sd <dir>  | -sa <file> |
   -qr <output_file>|
   -tal | -tald <dir>|
   -tab | -tabd <dir>}
{ <sample_file(s)> | -id <dir> | -if <file_of_files> }
```

ARGUMENTS

`-h` (Optional) Instructs ttuner to display the help screen for the command line arguments and which also provides contact information for TraceTuner support.

`-Q` (Optional) Turns status messaging off. If used in conjunction with the `-V` option, the parameter that appears last on the command line takes precedence.

`-V` (Optional) Specifies that TraceTuner produce additional process status messages. The default (without either the `-Q` or `-V` option) is verbosity level 1. The more `V`'s entered on the command line, the higher the level of verbosity. After level 3 (`-VV`), there is no

change in verbosity. If used in conjunction with the `-Q` option, the parameter that appears last on the command line takes precedence.

`-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 cannot be used together with `-recalln`.

If you use `-nocall`, you will get Ns. If you don't use `-nocall`, you will not get Ns.

`-recalln` (Optional) This parameter specifies to recall Ns, but not to change, insert or delete any other bases. All bases are relocated to the positions of corresponding intrinsic peaks. This option cannot 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, TraceTuner reads and starts from called bases and locations.

`-ipd <`*dir*`>` (Optional) Instructs TraceTuner to read original basecalls and locations from input `phd`-formated file(s) located in specified directory <*dir*> rather than from sample file(s). The electropherograms will be read from the sample file(s) as usual. The name of the input `phd` file should match the name of the sample file and have extension `.phd.1`. This option overrides the `-edited_bases` option and, in particular, allows using Phred's base calls or starting from them when recalling bases.

`-shift` (Optional) This option forces TraceTuner to correct mobility shifts in traces so as to make called bases more evenly spaced.

`-convolved` (Optional) This option forces TraceTuner to use the 'convolved' model of peak shape instead of the simpler 'gaussian' model, which is used by default.

`-het` (Optional) Specifies that TraceTuner make heterozygous base calls. See Table 3-1, *IUB Nucleotide Codes*, for a listing of the alternative bases used when making heterozygous base calls.

`-min_ratio <`*phr*`>` (Optional) Specifies a threshold ratio of heights of two peaks which may be eventually called as heterozygotes. This parameter is used only together with the `-het` option, that is, when TraceTuner attempts to make heterozygous base calls. The `min_ratio` parameter is used to sort out "good" second peak candidates from "noise" which should not be even considered. Only second peaks for which this ratio exceeds `min_ratio` are considered good candidates. The default value for `min_ratio` is 0.15.

`-t` <*lookup_table*> Instructs TraceTuner to use the specified external calibration lookup table. If this parameter is not used, TraceTuner attempts to read the location of the lookup table file using the environment variable `LOOKUP_TABLE`, or automatically selects the appropriate calibration table from ABI3700_Pop5_BigDye, ABI3700_Pop6_BigDye, ABI3100_Pop6_BigDye, ABI377_DT (Dye Terminator) or ABI377_DP (Dye Primer) based on the chemistry used when preparing the sample files. You can set the `LOOKUP_TABLE` environment variable to point to a specific lookup table:

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

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

`-C` <*consensus_file*> (Optional) Specifies that TraceTuner align the called bases with the sequence in the specified FastA-formatted consensus/reference file using the Smith-Waterman alignment algorithm. If this parameter is set, either the `-tal` or `-tald` output file parameter must also be set.

`-cv3` (Optional) This flag works only with the `-c` or `-cd` option. It forces ttuner to output version 3 of `.scf` files. The default output of `.scf` files is version 2. This option must be used together with either `-c` or `-cd <dir>`.

`-pop5` Forces TraceTuner to use the ABI3700_Pop5_BigDye built-in calibration table regardless of the chemistry used to make the electropherogram. By default, the table is selected automatically from the list of the five built-in generic tables.

`-pop6` Analogous to the `-pop5` parameter, but for the ABI3700_Pop6_BigDye calibration table.

`-3100` Analogous to the `-pop5` parameter, but for the ABI3100_Pop6_BigDye calibration table.

`-377dp` Analogous to the `-pop5` parameter, but for the ABI377_DP (Dye Primer) calibration table.

`-377dt` Analogous to the `-pop5` parameter, but for the ABI377_DT (Dye Terminator) calibration table.

`-trim_window` <*window_size*> (Optional) Specifies the size of the moving average of bases used to trim the ends of a sequence. The default size is 10. Trimming stops when the moving average of the quality values reaches `min_average_quality`. Trim data are stored in the `.phd` output file.

`-trim_threshold` <*min_average_quality*> (Optional) Specifies the threshold for the aforementioned quality value moving average. The default value is 20.

Important Note: At least *one* of the following output options *must* be selected: `-p`, `-pd`, `-q`, `-qd`, `-qa`, `-s`, `-sd`, `-sa`, `-c`, `-cd` or `-qr` so that TraceTuner knows which format to use for output. More than one option may be specified, in which case the output will be written in each of the specified formats.

`-p` Specifies that TraceTuner output be written to `.phd`-formatted files in the current working (i.e. sample file) directory. Note that `.phd` files are needed to utilize the Viewer.

`-pd` <*dir*> Analogous to `-p`, but output file is written to the specified directory.

`-q` Specifies that TraceTuner write *one* `.qual`-formatted quality value output file for *each* sample file to the sample file directory.

`-qd` **<*dir*>** Analogous to `-q`, but output file is written to the specified directory.

`-qa` **<*file*>** Specifies that TraceTuner aggregate the quality value output for *all* sample files into a *single* `.qual`-formatted output file and write the output to the specified file.

`-c` Specifies that TraceTuner output results in a `.scf`-formatted file in the sample file directory.

`-cd` <*dir*> Analogous to `-c`, but output file is written to the specified directory.

`-d` Specifies that TraceTuner output results in a `.poly`-formatted file in the sample file directory. If `-d` or `-dd` option is used without the `-het` option, a `.poly` file is produced, but mixed bases are not called.

`-dd` <*dir*> Analogous to `-d`, but output file is written to the specified directory.

`-s` Specifies that TraceTuner write its (potentially) recalled bases in one FastA-formatted sequence file for each sample file in the sample file directory.

`-sd` <*dir*> Analogous to `-s`, but output file is written to the specified directory.

`-sa` <*file*> Specifies that TraceTuner aggregate its (potentially) recalled bases for *all* sample files into a *single* FastA-formatted sequence file containing multiple sequences and write the output to the specified file.

`-qr` <*output_file*> Specifies that TraceTuner write a report file containing statistics on the number of input files that contain bases with QV ≥ 20 to the specified file. The format of the will be similar to that of `qvreport.txt`, an example of which can be seen on page 4-1.

`-tal` Specifies that TraceTuner align its base calls with the bases in <*consensus_file*> and output the results in a file with `.tal` extension in the sample file directory. The

Viewer requires `.tal` files in order to display TraceTuner base calls and local alignments with reference sequences.

`-tald` <*dir*> Analogous to `-tal`, but results are written to a file in the specified directory.

`-tab` Specifies that TraceTuner make alternative (heterozygous) base calls and output the results in a file with `.tab` extension in the sample file directory. The *Viewer* requires `.tab` files in order to display TraceTuner alternative base calls.

`-tabd` <*dir*> Analogous to `-tab`, but results are written to a file in the specified directory.

Important Note: One of the following output options *must* be selected: *sample_file(s)*, `-id` or `-if`. If the `-id` or `-if` arguments are not specified, sample file names must be provided on the command line.

sample_file(s) Specifies the input sample file(s) to be processed by TraceTuner. TraceTuner accepts UNIX-compressed and gzipped sample files.

`-id` <*dir*> Specifies that TraceTuner process every file in the specified directory.

`-if` <*file_of_files*> Specifies that TraceTuner read *file_of_files* and treat each line as a sample filename. An example of a *file_of_files* is as follows:

```
/home/username/ttuner/test_data/data_file1
/home/username/ttuner/test_data/data_file2
/home/username/ttuner/test_data/data_file3
```

Examples of Command Line Requests

Example 1 – Input from list of files

```
ttuner -Q -pd /mydir/outputdir -if
 /mydir/inputfile
```

`-Q` No status messages will be produced.

`-pd` The output will be in `.phd` formatted files and will be saved in the `outputdir` directory.

`-if` The *file_of_files* named `inputfile` containing the names of sample files will be used by TraceTuner.

Example 2 – Input from directory

```
ttuner -pd sept15 -qd sept15 -sd sept15 -id
 /home/username/ttuner/test_data
```

`-pd` The output will be in `.phd` 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 sample file in the `/home/username/ttuner/test_data` directory will be processed.

Example 3 – Input from multiple files

```
ttuner -nocall -c -sd sept16 test_data1.ab1
test_data2.ab1 test_data3.ab1
```

`-nocall` This parameter disables the TraceTuner base-calling feature.

`-c` This parameter specifies to output `.scf` files into the current directory.

`-sd` The output will be written in the directory `sept16` in FastA format.

The files `test_data1.ab1, test_data2.ab1` and `test_data3.ab1` are input sample files.

This page intentionally left blank.

CHAPTER 6

TraceTuner API

API Overview

This chapter describes the data structures, defined constants and functions that constitute the TraceTuner *Application Programming Interface* (API).

The API provides users with the capability of creating their own programs using the various TraceTuner functions, thus allowing TraceTuner to be integrated into a variety of custom environments.

The functions may be found in the `libtt.a` file in the installation directory. The data structures, defined constants and function prototypes may be found in the various header files (`*.h`) as indicated in the documentation.

The API data structures, defined constants and functions are grouped separately, then presented in alphabetical order. Note that the names of the data structures do not contain underscores and are in mixed case, while the defined constants are all uppercase, thus allowing the user to distinguish them easily from the API functions.

TABLE 6-1: TraceTuner API Tokens and Header Files

Token	Header Files
Data Structures	
BtkLookupEntry	Btk_lookup_table.h
BtkLookupTable	Btk_lookup_table.h
BTKMESSAGE	Btk_qv.h
Options	Btk_qv_data.h
TraceParamEntry	Btk_lookup_table.h
Defined Constants	
NAME_NONE	Btk_qv_io.h
NAME_FILES	Btk_qv_io.h
NAME_DIR	Btk_qv_io.h
NAME_MULTI	Btk_qv_io.h
Functions	
Btk_compute_qv()	Btk_qv.h Btk_lookup_table.h Btk_qv_data.h Btk_compute_qv.h
Btk_destroy_lookup_table()	Btk_lookup_table.h
Btk_get_3700pop5_table()	Btk_default_table.h
Btk_get_3700pop6_table()	Btk_default_table.h
Btk_get_3100pop6_table()	Btk_default_table.h
Btk_get_377dp_table()	Btk_default_table.h
Btk_get_377dt_table()	Btk_default_table.h
Btk_output_fasta_file()	Btk_qv.h Btk_match_data.h Btk_qv_data.h Btk_qv_io.h
Btk_output_phd_file()	Btk_qv.h Btk_match_data.h Btk_qv_data.h Btk_qv_io.h
Btk_output_quality_values()	Btk_qv.h Btk_match_data.h Btk_qv_data.h Btk_qv_io.h

Token	Header Files
Btk_output_scf_file()	Btk_qv.h Btk_match_data.h Btk_qv_data.h Btk_qv_io.h
Btk_output_tal_file()	Btk_qv.h Btk_match_data.h Btk_qv_data.h Btk_qv_io.h
Btk_read_lookup_table()	Btk_lookup_table.h
Btk_read_sample_file()	Btk_qv.h Btk_match_data.h Btk_qv_data.h Btk_qv_io.h
Btk_release_file_data()	Btk_qv.h Btk_match_data.h Btk_qv_data.h Btk_qv_io.h

API Data Structures

BtkLookupEntry

Header file

```
#include "Btk_lookup_table.h"
```

Structure

```
typedef struct _btk_quality_lookup_entry {
   char         qval;
   char         phr3i;
   char         phr7i;
   char         psr7i;
   char         presi;
   short        sind;
} BtkLookupEntry;
```

Description

This structure represents a single line of the body of the quality value lookup table. The four parameters *phr3i*, *phr7i*, *psr7i*, and *presi* are the order indices of the trace parameter threshold values corresponding to each quality value. Each index is an integer between 0 and `num_tpar_entries`, where `num_tpar_entries` is the number of lines in the header of the 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.

Fields

qval	Quality value
phr3i	Index of peak height ratio 3 threshold
phr7i	Index of peak height ratio 7 threshold
psr7i	Index of peak spacing ratio threshold
presi	Index of peak resolution threshold
sind	Order index of the entry which will replace the current entry if the table is reversely sorted in quality value

See also

`BtkLookupTable`
`TraceParamEntry`
`Btk_get_3700pop5_table()`
`Btk_read_lookup_table()`
`Btk_destroy_lookup_table()`

BtkLookupTable

Header file

```
#include "Btk_lookup_table.h"
```

Structure

```
typedef struct _btk_quality_lookup_table {
   int                    num_tpar_entries;
   TraceParamEntry        *tpar;
   int                    num_lut_entries;
   BtkLookupEntry         *entries;
} BtkLookupTable;
```

Description

A collection of `TraceParamEntry` and `BtkLookupEntry` structures that together describe a quality value lookup table. The `TraceParamEntry` structures form the table's header which consists of four columns and is *num_tpar_entries* lines long. The `BtkLookupEntry` structures form the body of the 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

`BtkLookupTable`
`TraceParamEntry`
`Btk_get_3700pop5_table()`
`Btk_read_lookup_table()`
`Btk_destroy_lookup_table()`

BTKMESSAGE

Header file

```
#include "Btk_qv.h"
```

Structure

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

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

Fields

code	An integer value used to return an error code
text	A text error message.

Options

Header file

```
#include "Btk_qv_data.h"
```

Structure

```
typdef struct{
   int      edited_bases;
   char     file_name[MAX_NAM_LENGTH];
   int      inp_phd;
   char     inp_phd_dir[MAX_NAM_LENGTH];
   int      gauss;
   char     lut_name[MAX_NAM_LENGTH];
   float    min_ratio;
   int      nocall;
   int      respace;
   double  *sf;
   char     path[MAX_NAM_LENGTH];
   int      recalln;
   int      renorm;
   int      shift;
   int      het;
   int      tab;
   char     tab_dir[MAX_NAM_LENGTH];
   int      tal;
   int      time;
   int      tip;
   char     tip_dir[MAX_NAM_LENGTH];
   int      Verbose;
}Options;
```

Description

The `Options` data structure is used to pass some of the command-line options of the function `Btk_compute_qv()`. The default value for `MAX_NAM_LENGTH` is 200.

Fields

edited_bases Specifies base calling from edited bases. A value of 0 specifies that called bases and locations be read from sample files and to start from them when recalling bases. A value of 1 specifies to read and use edited bases and locations from sample files.

file_name Char array containing the sample file name.

inp_phd Specifies reading original basecalls and locations from input `phd` file rather than from sample file.

inp_phd_dir Specifies the directory where the input phd file resides.

gauss Determines which model of peak shape to use. A value of 1 specifies the ‘gaussian’ model (default). A value of 0 specifies the

	'convolved' model which is generally more accurate but which takes longer to compute.
lut_name	Name of a lookup table. Permitted values are `"pop5"`, `"pop6"`, `"3100"`, `"377dp"` and `"377dt"`.
min_ratio	Specifies the threshold ratio of heights of the lowest peak to the highest peak at a given position. A value of 0.1 has been found to return good results.
nocall	Specifies base recalling. A value of 0 recalls the bases; any non-zero number skips recalling.
respace	Specifies *respacing* multiple peak positions in the *data expansion* part of the processing. This parameter causes reprocessing of groups of poorly resolved peaks so as to force the peak spacing to be close to the local average peak spacing of the ABI base caller. Any non-zero value causes this reprocessing; a value of 0 suppresses the reprocessing.
sf	Scaling factors used for testing and development.
path	Specifies sample file path.
recalln	Specifies the use of a simpler, TraceTuner 1.0-like base-recalling procedure: Ns are recalled to the best guess; other bases are not changed. A value of 0 specifies a more advanced base-recalling algorithm which, generally, changes the length of the called bases and location arrays. Any non-zero integer specifies to use the best guess and does not change the length of the array of called bases and locations.
renorm	Used for testing and development.
shift	Specifies mobility shift corrections. A value of 0 skips the corrections. Any non-zero integer value enables shift corrections.
het	Specifies heterozygote base calling. A value of 0 skips calling heterozygote bases. Any non-zero integer value enables processing of heterozygote bases.
tab	Specifies `.tab` file output. A value of 0 skips output. A non-zero value enables output.
tab_dir	Specifies the `.tab` file output directory.
tal	Specifies `.tal` file output.

time	Used for testing and development.
tip	Used for testing and development.
tip_dir	Used for testing and development.
Verbose	Sets verbosity level for status messages written to `stderr`: 0 for none; 1 for minimal and 3 for maximum.

See also `Btk_compute_qv()`

TraceParamEntry

Header file

```
#include "Btk_lookup_table.h"
```

Structure

```
typedef struct _trace_param-threshold_entry {
   double      phr3t;
   double      phr7t;
   double      psr7t;
   double      prest;
} TraceParamEntry;
```

Description

This structure represents a single line in the header of the quality value lookup table. The four parameters *phr3t*, *phr7t*, *psr7t*, and *prest* are the trace parameter threshold values corresponding to each quality value. The header of the lookup table consists of four columns, where each column contains sorted threshold values for a given trace parameter and is `num_tpar_entries` long, where `num_tpar_entries` is the number of header lines in the lookup table.

The second part of the lookup table refers to the threshold indices, which are the order indices of the thresholds in each column. The value of each of the four trace parameters *phr3t*, *phr7t*, *psr7t*, and *prest* for a base call must not exceed the value in the corresponding threshold values in order for the base call to be assigned the associated quality value, *qval*.

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

`BtkLookupTable`
`TraceParamEntry`
`Btk_get_3700pop5_table()`
`Btk_read_lookup_table()`
`Btk_destroy_lookup_table()`

API Constants

Defined Constants

Header file

```
#include "Btk_qv_io.h"
```

Defined constants

```
#define NAME_NONE         0
#define NAME_FILES        1
#define NAME_DIR          2
#define NAME_MULTI        8
```

Description

These defined constants are used in the interface of the two functions `Btk_output_quality_values()` and `Btk_output_fasta_file()`. The values are used to specify output destination(s). For example, to choose the destination corresponding to `NAME_FILES`, one might make the call:

```
Btk_output_quality_values( NAME_FILES, ... );
```

Because the constant is interpreted as a bit mask, one can use it to specify several destinations by doing a bitwise-OR operation of the constants. For example, to choose the destinations corresponding to both `NAME_FILES` and `NAME_MULTI`, one might make the call:

```
Btk_output_quality_values( NAME_FILES |
              NAME_MULTI,...);
```

Explanation of values

`NAME_NONE`	This constant has no effect on the output destination.
`NAME_FILES`	Output destination directory is the "current" directory.
`NAME_DIR`	Output destination directory is given by the `path` argument to the function.
`NAME_MULTI`	The output is appended to a file specified as an additional parameter given to the function.

See also

`Btk_output_fasta_file()`
`Btk_output_quality_values()`

API Functions

Btk_compute_qv()

Header files

```
#include "Btk_qv.h"
#include "Btk_lookup_table.h"
#include "Btk_qv_data.h"
#include "Btk_compute_qv.h"
```

Function

```
int Btk_compute_qv(
   int              *num_called_bases,
   char             **called_bases,
   int              **called_peak_locs,
   int                num_datapoints,
   int              **chromatogram,
   char             *color2base,
   BtkLookupTable   *table,
   int              **quality_values,
   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_called_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_called_bases* ints for the array *quality_values* and pass the pointer to this array. This function will reallocate the array as needed, compute the quality values and place them into the array.

Arguments

num_called_bases	Pointer to the input length of the array of called bases and the array of peak locations
called_bases	Pointer to array of called bases
called_peak_locs	Pointer to array of called peak locations
num_datapoints	Input length of chromatogram array
chromatogram	Input arrays which store chromatographic data for each of the dyes
color2base	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.
quality_values	Pointer to an array of quality values
options	Structure, members of which are some of the command-line options (e.g., *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()`

Btk_destroy_lookup_table()

Header files

```
#include "Btk_lookup_table.h"
```

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	Pointer to the lookup table.

See also

`BtkLookupTable`
`Btk_get_3700pop5_table()`
`Btk_read_lookup_table()`

Btk_get_3700pop5_table()

Btk_get_3700pop6_table()

Btk_get_3100pop6_table()

Btk_get_377dp_table()

Btk_get_377dt_table()

Functions

```
Btk_lookup_table *Btk_get_3700pop5_table()
Btk_lookup_table *Btk_get_3700pop6_table()
Btk_lookup_table *Btk_get_3100pop6_table()
Btk_lookup_table *Btk_get_377dp_table()
Btk_lookup_table *Btk_get_377dt_table()
```

Description

These functions use data compiled into the executable code to generate a `BtkLookupTable` structure. One of them (or `Btk_read_lookup_table()`) must be called before quality values can be calculated. When the application is finished with the function, the resources used by the table should be returned to the system by calling `Btk_destroy_lookup_table()`.

Arguments

None

See also

`BtkLookupTable`
`Btk_destroy_lookup_table()`
`Btk_read_lookup_table()`

Btk_output_fasta_file()

Header files

```
#include "Btk_qv.h"
#include "Btk_qv_data.h"
#include "Btk_match_data.h"
#include "Btk_qv_io.h"
```

Function

```
int Btk_output_fasta_file(
   int         FastaType,
   char       *file_name,
   char       *path,
   char       *multiseqFileName,
   char       *called_bases,
   int         num_bases,
   int         left_trim_point,
   int         right_trim_point,
   int         verbose
);
```

Description

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

Arguments

FastaType	Bit mask specifying where output will go. *FastaType* is obtained by using one of the pre-defined masks or by OR-ing together two or more of the predefined masks. These masks and their associated output destinations are described in the documentation for *Defined Constants* on page 6-2 and summarized here: `NAME_DIR` --> directory given by *path* `NAME_FILES` --> current directory `NAME_MULTI` --> appended to *multiseqFileName*
file_name	Name of the sample file
path	Path name of the directory where the `.seq` file is to be written
multiseqFileName	Name of file associated with `NAME_MULTI` bit of the bit mask specified by *FastaType*. Otherwise it is ignored.
called_bases	Array of base calls
num_bases	Number of elements in *called_bases*
left_trim_point	Base position of the left boundary of trimmed sequence
right_trim_point	Base position of the right boundary of trimmed sequence

verbose Not used in this function.

See also
`Btk_output_phd_file()`
`Btk_output_scf_file()`
`Btk_output_tal_file()`
`Btk_output_quality_values()`

Btk_output_phd_file()

Header files

```
#include "Btk_qv.h"
#include "Btk_qv_data.h"
#include "Btk_match_data.h"
#include "Btk_qv_io.h"
```

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            left_trim_point,
   int            right_trim_point,
   float          trim_threshold,
   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	Pathname 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 each trace.
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)
left_trim_point	Base position of the left boundary of trimmed sequence
right_trim_point	Base position of the right boundary of trimmed sequence
trim_threshold	Probability of error corresponding to `min_average_quality`
verbose	Sets verbosity level for status messages written to `stderr`: 0 for none; 2 for maximum.

See also

`Btk_output_fasta_file()`
`Btk_output_scf_file()`
`Btk_output_tal_file()`
`Btk_output_quality_values()`

Btk_output_quality_values()

Header files

```
#include "Btk_qv.h"
#include "Btk_qv_data.h"
#include "Btk_match_data.h"
#include "Btk_qv_io.h"
```

Function

```
int Btk_output_quality_values(
   int          *QualType,
   char         *file_name,
   char         *path,
   char         *multiqualFileName,
   int          *quality_values,
   int           num_values,
   int           left_trim_point,
   int           right_trim_point,
   int           verbose
);
```

Description

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

Arguments

QualType	Bit mask specifying where output will go. *QualType* is obtained by using one of the pre-defined masks or by OR-ing together two or more of the pre-defined masks. These masks, and their associated output destinations are described in the documentation for *Defined Constants* on page 6-2 and summarized here: `NAME_DIR` --> directory given by *path* `NAME_FILES` --> current directory `NAME_MULTI` --> appended to *multiqualFileName*
file_name	Name of the sample file
path	Pathname of the directory in which the `.qual` file should be written
multiqualFileName	Name of file associated with `NAME_MULTI` bit of the bit mask specified by *QualType*. Otherwise it is ignored.
quality_values	Array of quality values
num_values	Number of elements in *quality_values*
left_trim_point	Base position of the left boundary of trimmed sequence

right_trim_point	Base position of the right boundary of trimmed sequence
verbose	Not used in this function.

See also `Btk_output_fasta_file()`
`Btk_output_scf_file()`
`Btk_output_phd_file()`

Btk_output_scf_file()

Header files

```
#include "Btk_qv.h"
#include "Btk_qv_data.h"
#include "Btk_match_data.h"
#include "Btk_qv_io.h"
```

Function

```
int Btk_output_scf_file(
    char     *file_name,
    char     *path,
    int       scf_version,
    char     *called_bases,
    int      *called_locs,
    int      *quality_values,
    int       num_bases,
    int       num_datapoints,
    int     **avals,
    int     **cvals,
    int     **gvals,
    int     **tvals,
    int       nocall,
    char     *chemistry,
    int       verbose
);
```

Description

This function writes out analyzed chromatograms, base calls, base locations and other data to the specified file in `.scf` format, located in the directory specified by *`path`*. The name of the output file is formed from the name of input sample file using the following rule: if the input file name has an extension `.ab1` or `.abi`, then this extension will be replaced by `.scf`; otherwise, the suffix `.scf` will be appended to the name of the input sample file.

Arguments

file_name	Name of the input sample file
path	Path name of the directory where the `.scf` file is to be written
scf_version	Integer parameter `scf_version`, which may be either 2 or 3. When the `ttuner` command line arguments `-c` or `-cd <dir>` are used, `scf_version` by default will be set to 2. However, if `-cv3` command line flag is additionally used, then `scf_version` will be reset to 3.
called_bases	Array of base calls
called_locs	Array of called base locations
quality_values	Array of quality values assigned to bases

num_bases	Number of elements in *called_bases* array
num_datapoints	Number of datapoints in the trace
avals	Array of datapoints in the A trace
cvals	Array of datapoints in the C trace
gvals	Array of datapoints in the G trace
tvals	Array of datapoints in the T trace
nocall	Specifies base recalling. A value of 0 recalls bases; any non-zero number skips base recalling.
chemistry	Optional string that describes the chemistry used (primer or terminator)
verbose	Sets verbosity level for status messages written to `stderr`: ≤ 2 for none; > 2 for maximum.

See also `Btk_output_scf_file()`
`Btk_output_phd_file()`
`Btk_output_quality_values()`

Btk_output_tal_file()

Header files

```
#include "Btk_qv.h"
#include "Btk_qv_data.h"
#include "Btk_match_data.h"
#include "Btk_qv_io.h"
```

Function

```
int Btk_output_tal_file(
   char          *file_name,
   char          *path,
   char          *consensus_name,
   char          *consensus_seq,
   char          *called_bases,
   int            num_called_bases,
   int            Match,
   int            Substitution,
   int            Insertion,
   int            Deletion,
   float          RepeatFraction,
   int            Verbose
);
```

Description

If a region of similarity between TraceTuner's sequence of called bases and the reference/consensus sequence is detected, this function writes out TraceTuner's base calls which belong to the similarity region, the indices of the called bases, the consensus/reference sequence bases which belong to the similarity region, their indices and whether each particular TraceTuner base call matches the base in consensus/reference sequence. The name of the output file is formed by appending the `.tal` extension to the name of the input sample file.

Arguments

`file_name`	Name of the sample file
`path`	Pathname of the directory in which the `.tal` file should be written
`consensus_name`	Name of the FastA file which contains the consensus/reference sequence
`consensus_seq`	Array of consensus/reference bases
`Match`	Premium score for matching a called base to the base of the consensus sequence
`Substitution`	Penalty score for a substitution error in a called base
`Insertion`	Penalty score for an insertion error in a called base.
`Deletion`	Penalty score for a deletion error in a called base.

`RepeatFraction`	Threshold fraction of matched bases. If TraceTuner's sequence of called bases is similar to two or more regions in the consensus sequence so that the fraction of matched bases is equal to this parameter or higher, then all these regions will be considered as repeats. The value used by TraceTuner is 0.90.
`Verbose`	Sets verbosity level for status messages written to `stderr`: 0 for none, 1 for minimal and 2 for maximum.

See also

`Btk_output_fasta_file()`
`Btk_output_phd_file()`
`Btk_output_scf_file()`
`Btk_output_quality_values()`

Btk_read_lookup_table()

Header file

```
#include "Btk_lookup_table.h"
```

Function

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

Description

This function reads a quality value lookup table file and converts it into the BtkLookupTable structure. This function must be called with the name of a valid lookup table file before quality values can be calculated. Alternatively, one of the following functions can be called to generate a default lookup table:

- Btk_get_3700pop5_table()
- Btk_get_3700pop6_table()
- Btk_get_3100pop6_table()
- Btk_get_377dp_table()
- Btk_get_377dt_table()

When the application is finished it, the resources used by the table should be returned to the system by calling Btk_destroy_lookup_table().

Arguments

path	The pathname of the file containing the quality value lookup table.

See also

BtkLookupTable
Btk_get_3700pop5_table()
Btk_destroy_lookup_table()

Btk_read_sample_file()

Header files

```
#include "Btk_qv.h"
#include "Btk_qv_data.h"
#include "Btk_match_data.h"
#include "Btk_qv_io.h"
```

Function

```
int Btk_read_sample_file(
   char          *file_name,
   int           *num_bases,
   char         **called_bases,
   int            edited_bases,
   int          **called_locs,
   int           *num_values,
   int          **avals,
   int          **cvals,
   int          **gvals,
   int          **tvals,
   char         **call_method,
   char         **chemistry,
   char          *ConsensusName,
   char         **ConsensusSeq,
   int            ConsensusSpecified,
   Options        options,
   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 an array of called bases
edited_bases	Specifies whether to start base calling from edited bases. A value of 0 reads *called* bases and locations from sample files and starts from them when calling bases. A value of 1 reads and uses *edited* bases and locations from sample files.
called_locs	Address of a variable that will be set to an array of called base locations

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	As above for C trace points
gvals	As above for G trace points
tvals	As above for 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)
ConsensusName	Optional argument used for development. Pass `NULL` for normal use.
ConsensusSeq	As above
ConsensusSpecified	Optional argument used for development. Pass `0` (zero) for normal use.
options	Structure, members of which are some of the command-line options (e.g., *file_name*, *Verbose*, *nocall*, *recalln* and *edited_bases*)
message	Pointer to the caller-supplied error message structure `BTKMESSAGE`
verbose	Sets verbosity level for status messages written to `stderr`: 0 or 1 for minimal; 2 for moderate; 3 for maximum.

See also `Btk_release_file_data()`

Btk_release_file_data()

Header files

```
#include "Btk_qv.h"
#include "Btk_qv_data.h"
#include "Btk_match_data.h"
#include "Btk_qv_io.h"
```

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()`

This page intentionally left blank.

CHAPTER 7 *Sample API Program*

Overview

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 `.qual` format.

The source code of `example.c` is included on the CD. To make the example executable, after completing the UNIX/LINUX/IRIX installation, change your directory to `ttuner<`*version*`>` and enter the command:

```
make -f example.mk
```

An executable called `example` will be created. To run this executable, enter:

```
example <sample_file>
```

Sample Source Code

```
/**  $Revision: 2.28 $
 **  example.c - Sample application program to output .qual files from
```

```
 **              ABI sample file inputs using the TraceTuner API library.
 **/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <float.h>

#include "Btk_qv.h"
#include "Btk_qv_data.h"
#include "Btk_lookup_table.h"
#include "Btk_default_table.h"
#include "Btk_compute_qv.h"
#include "Btk_match_data.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, *chemistry = "";
    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 environment
     *  variable, use it, otherwise NULL gets the default.
     */
    lookup_table = getenv("LOOKUP_TABLE");

    if (lookup_table != NULL) {
        if (strcmp(lookup_table, "pop5") == 0) {
            table = Btk_get_3700pop5_table();
        }
        else if (strcmp(lookup_table, "pop6") == 0) {
            table = Btk_get_3700pop6_table();
        }
```

```
    else if (strcmp(lookup_table, "377dp") == 0) {
        table = Btk_get_377dp_table();
    }
    else if (strcmp(lookup_table, "377dt") == 0) {
        table = Btk_get_377dt_table();
    }
    else {
        if ((table = Btk_read_lookup_table(lookup_table)) == NULL) {
            fprintf(stderr, "Couldn't read lookup table '%s'.\n",
                    lookup_table);
            exit(1);
        }
    }
}
else {
    table = NULL;
}

for(n=1; n < argc; n++) {
    smp = argv[n];
    if ((smptail = strrchr(smp, '/')) != NULL)
        smptail++;
    else
        smptail = smp;
    /* Setting default options
     */
    options.nocall = 0;
    options.recalln = 0;
    options.edited_bases = 0;
    options.gauss = 1;
    options.shift = 0;
    options.renorm = 0;
    options.tip = 0;
    options.tal = 0;
    options.tab = 0;
    options.het = 0;
    options.time = 0;
    options.min_ratio = 0.15;
    options.inp_phd = 0;
    options.inp_phd_dir[0] = '\0';
    options.file_name = '\0';
    strcpy(options.lut.name, "pop5");

    if (Btk_read_sample_file(smp, &nbases, &bases, 0, &locations,
        &nvals, &vals[0], &vals[1], &vals[2], &vals[3],
```

```
    NULL, &chemistry, NULL, NULL, 0, options, &msg, 0) != 0)
{
    fprintf(stderr, "%s: couldn't read sample file\n", smptail);
    continue;
}

qv = (int *)malloc(nbases * sizeof(int));
strcpy(options.file_name, smptail);

if (table == NULL) {
    if (strstr(chemistry, "POP5")) {
        table = Btk_get_3700pop5_table();
    }
    else if (strstr(chemistry, "POP6") &&
             strstr(chemistry, "3700")) {
        table = Btk_get_3700pop6_table();
    }
    else if (strstr(chemistry, "3100")) {
        table = Btk_get_3100pop6_table();
    }
    else if (strstr(chemistry, "DyePrimer") ||
             strstr(chemistry, "DP")){
        table = Btk_get_377dp_table();
    }
    else if (strstr(chemistry, "ET")        ||
             strstr(chemistry, "DyeTerm")   ||
             strstr(chemistry, "AnyPrimer") ||
             strstr(chemistry, "Any-Primer")) {
        table = Btk_get_377dt_table();
    }
    else {
    fprintf(stderr,
            "Can't select the lookup table automatically. \n"
            "Using built-in ABI 3700 Pop-5 table.\n");
    table = Btk_get_3700pop5_table();
    }
}

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;
}
fprintf(stderr, "%s: %d bases. ", smp, nbases);
fprintf(stderr, "QVs are output to .qual file\n");
```

```
        Btk_output_quality_values(NAME_FILES, smp, NULL, "", qv, nbases, 0,
                                  nbases - 1, 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);
}
```

This page intentionally left blank.

Glossary

Entries

Alternative base call When TraceTuner calls heterozygotes, it considers one or more "candidate" base calls, pure or heterozygote, for each expected location of a called base. Each "candidate" base call is formed by peaks or couples of peaks present near this expected location. TraceTuner then evaluates the quality value of each candidate, selects the one having the highest quality value, calls and outputs it into a `.phd` file. The second highest quality value and all other considered candidate base calls are not called and are output to the `.tab` file. The TraceTuner Viewer visualizes only the second highest quality value candidate.

Apparent height The intensity of the analyzed signal produced by the ABI basecaller at the apparent position of a peak.

Apparent position The position in an electropherogram which bisects the peak area. The peak area is defined as the area below a part of electropherogram located between beginning and end of a peak, as determined by TraceTuner.

Bucket After determining the trimmed lengths of the samples, TraceTuner uses the greatest length, *max_length*, to create *max_length/10 buckets*, each ten bases in width. TraceTuner distributes reads into the appropriate buckets according to the number of bases in the sample with a QV≥ 20. Sorting reads into buckets provides an overview of the distribution of read quality and trimmed lengths for all the samples in a run.

Edited base Base call produced by any third party software, such as Phred, or by "manual" editing of an original/ABI base call, and stored in ABI sample file as edited base. The edited base is stored together with its location in an electropherogram, which is called an *edited location*. The called and edited bases stored in an ABI sample file may or may not be identical. An SCF file contains only one string of bases, so in this case called bases are always identical to edited bases.

Heterozygous base A call of heterozygous bases. Assignments are made using the IUB code for nucleotide pairs (see Table 3-1). See also *SNP*.

Intrinsic signal An analyzed signal (a peak) in an electropherogram which corresponds to an individual DNA fragment. If there are no other peaks of the same dye color in the vicinity of given DNA peak, then the intrinsic signal simply coincides with the analyzed, or observed signal. If two or more DNA peaks of the same dye color overlap, the intrinsic signal from each peak is determined by TraceTuner using theoretical models of peak shapes that are built into the software.

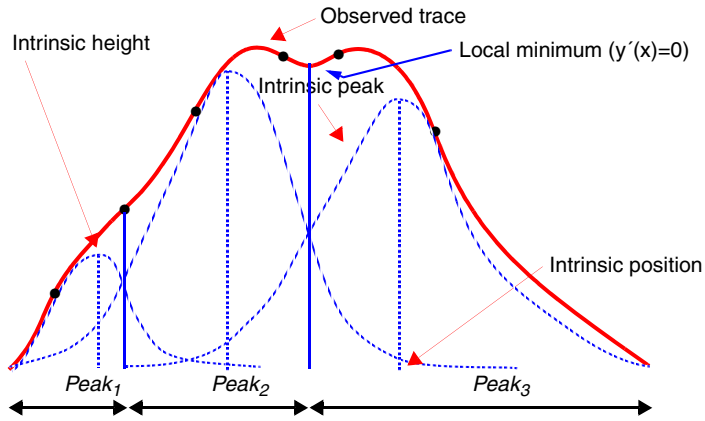
Intrinsic height The height of the TraceTuner intrinsic peak (Figure G-1).

Intrinsic peak Intrinsic peaks are the peaks into which TraceTuner resolves the observed trace. A working assumption of TraceTuner is that the resolution of the curve should involve the smallest possible number of intrinsic peaks.

In the case of simple peaks, no resolution is required since the apparent peak and the intrinsic peak are identical.

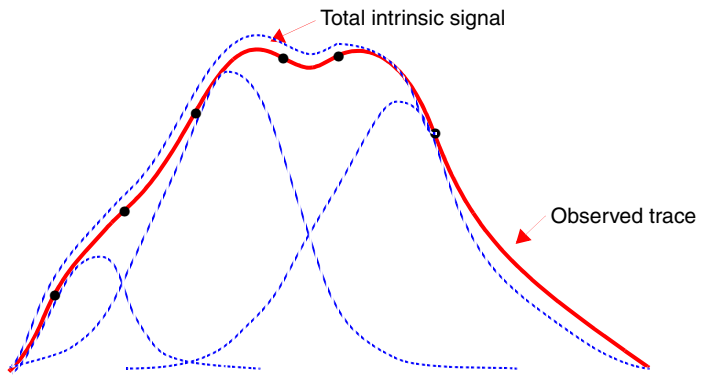
In the case of multiple peaks, TraceTuner resolves the observed trace into intrinsic peaks. As seen in Figure G-1, *Phred* uses the area under the curve between adjacent inflection points to calculate the peaks and the base positions. By contrast, TraceTuner uses the entire area under the observed trace by extending the lines separating the peak areas until they touch or until the trace curve drops to zero (Figure G-1). As TraceTuner scans a sample file, it collects information about the shape of the simple peaks occurring in that particular sample file. TraceTuner uses these shapes in a proprietary iterative fitting process to resolve multiple peaks into a number of simple, intrinsic peaks.

FIGURE G-1: Peak Areas in TraceTuner



TraceTuner then sums the intrinsic peak curves to form a *total intrinsic signal* (Figure G-2).

FIGURE G-2: Total Intrinsic Signal in TraceTuner



Intrinsic position The *x-loc* of the vertical line marking the intrinsic height.

Minimum peak height ratio The threshold ratio of heights of two peaks which may be considered as candidates for heterozygote base calls. If the actual ratio of peak heights is below this threshold, they will not be considered as candidates for the heterozygote base call.

Mobility shift corrections TraceTuner calls bases at the positions of intrinsic peaks as determined from analyzed electropherograms. Under certain circumstances, because of insufficient processing of electropherograms by the original base caller, the locations of the bases called by

TraceTuner may be unevenly spaced. The `-shift` option forces TraceTuner to perform, if possible, an additional correction of the locations of intrinsic peaks, thus making TraceTuner's base calls more evenly spaced.

Observed signal The analyzed signal produced by the original basecaller. If two peaks produced by different DNA fragments but attached to the dye of the same color overlap, then the intensity of the observed signal will be a sum of individual, or intrinsic, signals from these peaks. See also *Intrinsic signal*.

SNP Short for *Single Nucleotide Polymorphism*. A type of sequence variation polymorphism consisting of a change in only one base. See entry for *Heterozygous base*.

Trace parameters Combinations of peak characteristics, such as peak height, position or shape, which characterize the local "environment" of a given called peak in an electropherogram. During the training procedure, this information, together with additional information about the correctness of each particular base call, is used to generate a lookup table. Like Phred, TraceTuner then computes trace parameters and uses the stored lookup table to determine a quality of a given base call. However, the trace parameters used by TraceTuner are different from those used by Phred.

Index