

cawlin

0.0.1

Generated by Doxygen 1.9.8

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 <code>argparse::args_t</code> Class Reference	7
4.1.1 Constructor & Destructor Documentation	7
4.1.1.1 <code>args_t()</code>	7
4.1.1.2 <code>~args_t()</code>	8
4.2 <code>CawalignCodonScores</code> Class Reference	8
4.2.1 Constructor & Destructor Documentation	9
4.2.1.1 <code>CawalignCodonScores()</code>	9
4.2.2 Member Function Documentation	9
4.2.2.1 <code>nucleotide_diff()</code>	9
4.3 <code>CawalignSimpleScores</code> Class Reference	10
4.3.1 Constructor & Destructor Documentation	10
4.3.1.1 <code>CawalignSimpleScores()</code> [1/2]	10
4.3.1.2 <code>CawalignSimpleScores()</code> [2/2]	11
4.3.2 Member Function Documentation	11
4.3.2.1 <code>_init_alphabet()</code>	11
4.4 <code>ConfigParser</code> Class Reference	12
4.4.1 Constructor & Destructor Documentation	12
4.4.1.1 <code>ConfigParser()</code>	12
4.4.2 Member Function Documentation	12
4.4.2.1 <code>aConfig()</code>	12
4.4.2.2 <code>aConfigVec()</code>	13
4.5 <code>sequence_gap_structure</code> Struct Reference	13
4.6 <code>StringBuffer</code> Class Reference	14
4.6.1 Constructor & Destructor Documentation	14
4.6.1.1 <code>StringBuffer()</code>	14
4.6.1.2 <code>~StringBuffer()</code>	14
4.6.2 Member Function Documentation	14
4.6.2.1 <code>appendBuffer()</code>	14
4.6.2.2 <code>appendChar()</code>	15
4.6.2.3 <code>flip()</code>	15
4.6.2.4 <code>resetString()</code>	15
4.6.2.5 <code>swap()</code>	15
4.7 <code>Vector</code> Class Reference	16

4.7.1 Constructor & Destructor Documentation	16
4.7.1.1 Vector()	16
4.7.1.2 ~Vector()	16
4.7.2 Member Function Documentation	16
4.7.2.1 appendValue()	16
4.7.2.2 appendVector()	17
4.7.2.3 extractMin()	17
4.7.2.4 remove()	17
4.7.2.5 resetVector()	17
4.7.2.6 sort()	18
4.7.2.7 storeValue()	18
4.7.2.8 storeVector()	18
4.7.2.9 swap()	18
4.7.2.10 value()	19
4.8 VectorFP Class Reference	19
4.8.1 Constructor & Destructor Documentation	20
4.8.1.1 VectorFP()	20
4.8.1.2 ~VectorFP()	20
4.8.2 Member Function Documentation	20
4.8.2.1 appendValue()	20
4.8.2.2 appendValues()	20
4.8.2.3 storeValue()	21
5 File Documentation	23
5.1 alignment.h	23
5.2 argparse.hpp	24
5.3 configparser.hpp	25
5.4 scoring.hpp	27
5.5 stringBuffer.h	28
5.6 tn93_shared.h	29
Index	31

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

argparse::args_t	7
CawalignSimpleScores	10
CawalignCodonScores	8
ConfigParser	12
sequence_gap_structure	13
StringBuffer	14
Vector	16
VectorFP	19

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

argparse::args_t	7
CawalignCodonScores	8
CawalignSimpleScores	10
ConfigParser	12
sequence_gap_structure	13
StringBuffer	14
Vector	16
VectorFP	19

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

src/alignment.h	23
src/argparse.hpp	24
src/configparser.hpp	25
src/scoring.hpp	27
src/stringBuffer.h	28
src/tn93_shared.h	29

Chapter 4

Class Documentation

4.1 `argparse::args_t` Class Reference

Public Member Functions

- [args_t](#) (int, const char **)
- [~args_t](#) ()

Public Attributes

- FILE * **output**
- FILE * **reference**
- FILE * **input**
- [ConfigParser](#) * **scores**
- data_t **data_type**
- local_t **local_option**
- space_t **space_type**
- out_format_t **out_format**
- rc_t **reverse_complement**
- bool **quiet**
- bool **affine**
- bool **include_reference**

4.1.1 Constructor & Destructor Documentation

4.1.1.1 `args_t()`

```
argparse::args_t::args_t (
    int ,
    const char ** )
```

Constructor for [args_t](#), which parses command-line arguments and sets up configuration options.

This constructor processes the command-line arguments, setting up the program's input, output, reference files, and various other configuration options such as data type, space type, and output format. If necessary, default values are assigned to some options.

Parameters

<i>argc</i>	The number of command-line arguments.
<i>argv</i>	The array of command-line arguments.

4.1.1.2 ~args_t()

```
argparse::args_t::~~args_t ( )
```

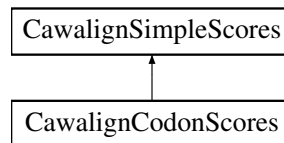
Destructor for [args_t](#), responsible for cleaning up any resources used (files or memory). It closes the input/output/reference files and deletes the scores object, if applicable.

The documentation for this class was generated from the following files:

- src/argparse.hpp
- src/argparse.cpp

4.2 CawalignCodonScores Class Reference

Inheritance diagram for CawalignCodonScores:



Public Member Functions

- [CawalignCodonScores](#) ([ConfigParser](#) *)

Public Member Functions inherited from [CawalignSimpleScores](#)

- [CawalignSimpleScores](#) (const char *_alphabet, const cawalign_fp *_scoring_matrix, const cawalign_fp _↔_open_gap_reference, const cawalign_fp _open_gap_query, const cawalign_fp _extend_gap_reference, const cawalign_fp _extend_gap_query)
- [CawalignSimpleScores](#) ([ConfigParser](#) *)
- void [_init_alphabet](#) (long not_found=-1)

Static Public Member Functions

- static int [nucleotide_diff](#) (long, long)

Public Attributes

- [Vector](#) `translation_table`
- [VectorFP](#) `s3x1`
- [VectorFP](#) `s3x2`
- [VectorFP](#) `s3x4`
- [VectorFP](#) `s3x5`
- `cawalign_fp frameshift_cost`
- `cawalign_fp synonymous_penalty`
- [StringBuffer](#) `amino_acids`
- `int stop_codon_index`
- `int mismatch_index`

Public Attributes inherited from [CawalignSimpleScores](#)

- [StringBuffer](#) `alphabet`
- `unsigned int D`
- `long char_map [255]`
- [VectorFP](#) `scoring_matrix`
- `cawalign_fp open_gap_reference`
- `cawalign_fp open_gap_query`
- `cawalign_fp extend_gap_query`
- `cawalign_fp extend_gap_reference`
- `char gap_char`

4.2.1 Constructor & Destructor Documentation

4.2.1.1 CawalignCodonScores()

```
CawalignCodonScores::CawalignCodonScores (
    ConfigParser * settings )
```

Constructs a [CawalignCodonScores](#) object using configuration settings.

This constructor initializes the codon scoring system using values from a [ConfigParser](#). It sets up the codon translation table, stop codon index, mismatch index, and scoring matrices for codon alignments. Throws errors if the amino acid alphabet is incomplete or the translation table is invalid.

Parameters

<code>settings</code>	A pointer to a ConfigParser object containing configuration settings.
-----------------------	---

4.2.2 Member Function Documentation

4.2.2.1 nucleotide_diff()

```
int CawalignCodonScores::nucleotide_diff (
    long c1,
    long c2 ) [static]
```

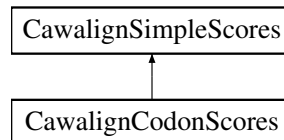
Calculates the number of nucleotide differences between two codons.

The documentation for this class was generated from the following files:

- src/scoring.hpp
- src/scoring.cpp

4.3 CawalignSimpleScores Class Reference

Inheritance diagram for CawalignSimpleScores:



Public Member Functions

- [CawalignSimpleScores](#) (const char *_alphabet, const cawalign_fp *_scoring_matrix, const cawalign_fp _↔ open_gap_reference, const cawalign_fp _open_gap_query, const cawalign_fp _extend_gap_reference, const cawalign_fp _extend_gap_query)
- [CawalignSimpleScores](#) ([ConfigParser](#) *)
- void [_init_alphabet](#) (long not_found=-1)

Public Attributes

- [StringBuffer](#) **alphabet**
- unsigned int **D**
- long **char_map** [255]
- [VectorFP](#) **scoring_matrix**
- cawalign_fp **open_gap_reference**
- cawalign_fp **open_gap_query**
- cawalign_fp **extend_gap_query**
- cawalign_fp **extend_gap_reference**
- char **gap_char**

4.3.1 Constructor & Destructor Documentation

4.3.1.1 CawalignSimpleScores() [1/2]

```

CawalignSimpleScores::CawalignSimpleScores (
    const char * _alphabet,
    const cawalign_fp * _scoring_matrix,
    const cawalign_fp _open_gap_reference,
    const cawalign_fp _open_gap_query,
    const cawalign_fp _extend_gap_reference,
    const cawalign_fp _extend_gap_query )
  
```

Constructs a [CawalignSimpleScores](#) object with a custom alphabet and scoring matrix.

This constructor initializes the scoring system with a user-specified alphabet and scoring matrix, as well as gap penalties for both reference and query sequences. Throws an error if the alphabet is empty.

Parameters

<code>_alphabet</code>	The alphabet for scoring (e.g., nucleotides or amino acids).
<code>_scoring_matrix</code>	A pointer to the scoring matrix values.
<code>_open_gap_reference</code>	Gap opening penalty for reference sequences.
<code>_open_gap_query</code>	Gap opening penalty for query sequences.
<code>_extend_gap_reference</code>	Gap extension penalty for reference sequences.
<code>_extend_gap_query</code>	Gap extension penalty for query sequences.

4.3.1.2 CawalignSimpleScores() [2/2]

```
CawalignSimpleScores::CawalignSimpleScores (
    ConfigParser * settings )
```

Constructs a [CawalignSimpleScores](#) object using configuration settings.

This constructor reads configuration values from a [ConfigParser](#) to initialize the alphabet, scoring matrix, and gap penalties. Throws errors if the alphabet is missing or the scoring matrix dimensions are incorrect.

Parameters

<code>settings</code>	A pointer to a ConfigParser object containing configuration settings.
-----------------------	---

4.3.2 Member Function Documentation

4.3.2.1 _init_alphabet()

```
void CawalignSimpleScores::_init_alphabet (
    long not_found = -1 )
```

Initializes the character map for the scoring matrix.

This function populates the `char_map` array, mapping each character in the alphabet to its index in the scoring matrix. Characters not in the alphabet are assigned a value of `not_found`.

Parameters

<code>not_found</code>	The value to assign for characters not found in the alphabet.
------------------------	---

The documentation for this class was generated from the following files:

- `src/scoring.hpp`
- `src/scoring.cpp`

4.4 ConfigParser Class Reference

Public Member Functions

- [ConfigParser](#) (std::ifstream &configFile)
- template<typename T >
T **aConfig** (std::string section, std::string name, size_t pos=0)
- template<typename T >
std::vector< T > **aConfigVec** (std::string section, std::string name)
- template<> bool [aConfig](#) (std::string section, std::string configName, size_t pos)
- template<> std::vector< bool > [aConfigVec](#) (std::string section, std::string configName)
- template<> bool **aConfig** (std::string section, std::string name, size_t pos)
- template<> std::vector< bool > **aConfigVec** (std::string section, std::string name)

4.4.1 Constructor & Destructor Documentation

4.4.1.1 ConfigParser()

```
ConfigParser::ConfigParser (
    std::ifstream & configFile )
```

Constructor for [ConfigParser](#).

This function reads and parses a configuration file. It processes lines to strip whitespace, ignores comments, and splits key-value pairs using the = symbol. If the key is within a section (indicated by [. . .]), the key is prefixed with the section name. Multiple values for a key are stored in a vector. Parsed configurations are stored in a map with section and key names combined as the key.

Parameters

<i>configFile</i>	A reference to an <code>ifstream</code> representing the open configuration file.
-------------------	---

Exceptions

<code>std::runtime_error</code>	if a parsing error occurs in the configuration file.
---------------------------------	--

4.4.2 Member Function Documentation

4.4.2.1 aConfig()

```
template<>
bool ConfigParser::aConfig (
    std::string section,
    std::string configName,
    size_t pos )
```

Template specialization for retrieving boolean configuration values.

This function retrieves a specific boolean configuration value from a section and configuration name. The value is parsed as either `true` (for "true", "TRUE", or "1") or `false` (for "false", "FALSE", or "0"). If the value does not match any of these, the function defaults to `false`.

Parameters

<i>section</i>	The section name in the configuration file.
<i>configName</i>	The key within the section.
<i>pos</i>	The position of the value in the vector (if there are multiple values).

Returns

`true` if the configuration value is recognized as true, otherwise `false`.

4.4.2.2 aConfigVec()

```
template<>
std::vector< bool > ConfigParser::aConfigVec (
    std::string section,
    std::string configName )
```

Template specialization for retrieving a vector of boolean configuration values.

This function retrieves a vector of boolean values associated with a configuration key in a section. Each value is parsed as either `true` (for "true", "TRUE", or "1") or `false` (for "false", "FALSE", or "0"). If a value does not match any of these, it defaults to `false`.

Parameters

<i>section</i>	The section name in the configuration file.
<i>configName</i>	The key within the section.

Returns

A vector of boolean values parsed from the configuration.

The documentation for this class was generated from the following files:

- src/configparser.hpp
- src/configparser.cpp

4.5 sequence_gap_structure Struct Reference

Public Attributes

- long **first_nongap**
- long **last_nongap**
- long **resolved_start**
- long **resolved_end**

The documentation for this struct was generated from the following file:

- src/tn93_shared.h

4.6 StringBuffer Class Reference

Public Member Functions

- [StringBuffer](#) (void)
- [~StringBuffer](#) (void)
- char * **getString** (void) const
- void [appendChar](#) (const char)
- void [appendBuffer](#) (const char *, const long=-1)
- void [resetString](#) (void)
- void [swap](#) ([StringBuffer](#) &)
- unsigned long **length** (void) const
- void **reset_length** (unsigned long newL)
- char **setChar** (const long i, const char c)
- char **getChar** (const long i) const
- void [flip](#) (void)
- void **detach** (void)

Static Public Attributes

- static long **sbDefaultLength** = 16
- static long **sbDefaultBoost** = 16

4.6.1 Constructor & Destructor Documentation

4.6.1.1 StringBuffer()

```
StringBuffer::StringBuffer (  
    void )
```

Initializes an empty [StringBuffer](#) with a default initial capacity. This buffer dynamically grows as new characters are appended.

4.6.1.2 ~StringBuffer()

```
StringBuffer::~~StringBuffer (  
    void )
```

Frees the memory allocated for the string buffer.

4.6.2 Member Function Documentation

4.6.2.1 appendBuffer()

```
void StringBuffer::appendBuffer (  
    const char * buffer,  
    const long length = -1 )
```

Appends a string or a buffer of specified length to the [StringBuffer](#).

Parameters

<i>buffer</i>	The string or character buffer to append.
<i>length</i>	The length of the buffer, if known. If not, the length is inferred using <code>strlen</code> .

4.6.2.2 appendChar()

```
void StringBuffer::appendChar (
    const char c )
```

Appends a single character to the end of the buffer, growing the buffer if needed.

Parameters

<i>c</i>	The character to append.
----------	--------------------------

4.6.2.3 flip()

```
void StringBuffer::flip (
    void )
```

Reverses the content of the [StringBuffer](#).

4.6.2.4 resetString()

```
void StringBuffer::resetString (
    void )
```

Resets the [StringBuffer](#) to an empty state.

Clears the buffer content by resetting its length, but keeps the allocated memory.

4.6.2.5 swap()

```
void StringBuffer::swap (
    StringBuffer & src )
```

Swaps the contents of this [StringBuffer](#) with another [StringBuffer](#).

This function exchanges the data, length, and capacity of two [StringBuffer](#) objects.

Parameters

<i>src</i>	The StringBuffer object to swap with.
------------	---

The documentation for this class was generated from the following files:

- src/stringBuffer.h
- src/stringBuffer.cc

4.7 Vector Class Reference

Public Member Functions

- [Vector](#) (void)
- [~Vector](#) (void)
- void [appendValue](#) (const long)
- void [appendVector](#) (const [Vector](#) &)
- long [extractMin](#) ([VectorFP](#) &)
- void [resetVector](#) (void)
- void [remove](#) (const unsigned long)
- void [storeValue](#) (const long, const unsigned long)
- void [storeVector](#) (const [Vector](#) &, const unsigned long)
- void [sort](#) (void)
- void [swap](#) ([Vector](#) &)
- long [value](#) (const long idx) const
- unsigned long [length](#) (void) const

Static Public Attributes

- static long [vDefaultLength](#) = 16
- static long [vDefaultBoost](#) = 16

4.7.1 Constructor & Destructor Documentation

4.7.1.1 Vector()

```
Vector::Vector (  
    void )
```

Initializes an empty vector of long values with a default initial capacity.

4.7.1.2 ~Vector()

```
Vector::~~Vector (  
    void )
```

Frees the memory allocated for the vector data.

4.7.2 Member Function Documentation

4.7.2.1 appendValue()

```
void Vector::appendValue (  
    const long l )
```

Appends a value to the [Vector](#).

Adds a long value to the end of the vector, growing the vector if needed.

Parameters

<i>l</i>	The long value to append.
----------	---------------------------

4.7.2.2 appendVector()

```
void Vector::appendVector (
    const Vector & v )
```

Appends the contents of another [Vector](#) to this [Vector](#).

Parameters

<i>v</i>	The source Vector whose contents are to be appended.
----------	--

4.7.2.3 extractMin()

```
long Vector::extractMin (
    VectorFP & values )
```

Extracts the minimum value from the vector based on a [VectorFP](#) of floating-point values.

Parameters

<i>values</i>	The VectorFP of floating-point values to compare.
---------------	---

Returns

The index of the minimum value, or -1 if the vector is empty.

4.7.2.4 remove()

```
void Vector::remove (
    const unsigned long l )
```

Removes the element at the given index and shifts the remaining elements to fill the gap.

Parameters

<i>l</i>	The index of the element to remove.
----------	-------------------------------------

4.7.2.5 resetVector()

```
void Vector::resetVector (
    void )
```

Resets the [Vector](#) to an empty state.

Clears the vector by resetting its length, but keeps the allocated memory.

4.7.2.6 `sort()`

```
void Vector::sort (
    void )
```

Sorts the [Vector](#) in ascending order.

4.7.2.7 `storeValue()`

```
void Vector::storeValue (
    const long v,
    const unsigned long l )
```

Stores a value at a specific index in the [Vector](#).

If the index is beyond the current capacity, the vector grows to accommodate the value.

Parameters

<code>v</code>	The long value to store.
<code>l</code>	The index at which to store the value.

4.7.2.8 `storeVector()`

```
void Vector::storeVector (
    const Vector & v,
    const unsigned long l )
```

Stores a pointer to a [Vector](#) at a specified index, growing the vector if necessary.

Parameters

<code>v</code>	The Vector object to store.
<code>l</code>	The index at which to store the vector.

4.7.2.9 `swap()`

```
void Vector::swap (
    Vector & src )
```

Swaps the contents of this [Vector](#) with another [Vector](#).

Exchanges the data, length, and capacity of two [Vector](#) objects.

Parameters

<i>src</i>	The Vector object to swap with.
------------	---

4.7.2.10 value()

```
long Vector::value (
    const long idx ) const
```

Retrieves a value from the [Vector](#) at the specified index.

Parameters

<i>idx</i>	The index from which to retrieve the value.
------------	---

Returns

The value stored at the specified index.

The documentation for this class was generated from the following files:

- src/stringBuffer.h
- src/stringBuffer.cc

4.8 VectorFP Class Reference

Public Member Functions

- [VectorFP](#) (void)
- [~VectorFP](#) (void)
- void [appendValue](#) (const cawalign_fp)
- void [appendValues](#) (const cawalign_fp *, long)
- void [storeValue](#) (const cawalign_fp, const unsigned long)
- cawalign_fp [value](#) (const long *idx*)
- unsigned long [length](#) (void) const
- const cawalign_fp * [values](#) (void)
- cawalign_fp * [rvalues](#) (void)

Static Public Attributes

- static long [vDefaultLength](#) = 16
- static long [vDefaultBoost](#) = 16

4.8.1 Constructor & Destructor Documentation

4.8.1.1 VectorFP()

```
VectorFP::VectorFP (  
    void )
```

Initializes an empty vector of floating-point values with a default initial capacity.

4.8.1.2 ~VectorFP()

```
VectorFP::~~VectorFP (  
    void )
```

Frees the memory allocated for the floating-point vector data.

4.8.2 Member Function Documentation

4.8.2.1 appendValue()

```
void VectorFP::appendValue (  
    const cawlign_fp l )
```

Appends a floating-point value to the [VectorFP](#).

Adds a floating-point value to the end of the vector, growing the vector if needed.

Parameters

<i>l</i>	The floating-point value to append.
----------	-------------------------------------

4.8.2.2 appendValues()

```
void VectorFP::appendValues (  
    const cawlign_fp * l,  
    long N )
```

Appends multiple floating-point values to the [VectorFP](#).

Parameters

<i>l</i>	The array of floating-point values to append.
<i>N</i>	The number of values to append.

4.8.2.3 storeValue()

```
void VectorFP::storeValue (
    const cawign_fp v,
    const unsigned long l )
```

Stores a floating-point value at a specific index in the [VectorFP](#).

If the index is beyond the current capacity, the vector grows to accommodate the value.

Parameters

<i>v</i>	The floating-point value to store.
<i>l</i>	The index at which to store the value.

The documentation for this class was generated from the following files:

- src/stringBuffer.h
- src/stringBuffer.cc

Chapter 5

File Documentation

5.1 alignment.h

```
00001 /*
00002
00003 HyPhy - Hypothesis Testing Using Phylogenies.
00004
00005 Copyright (C) 1997-now
00006 Core Developers:
00007 Sergei L Kosakovsky Pond (spond@ucsd.edu)
00008 Art FY Poon (apoon42@uwo.ca)
00009 Steven Weaver (sweaver@ucsd.edu)
00010
00011 Module Developers:
00012 Lance Hepler (nlhepler@gmail.com)
00013 Martin Smith (martin.audacis@gmail.com)
00014
00015 Significant contributions from:
00016 Spencer V Muse (muse@stat.ncsu.edu)
00017 Simon DW Frost (sdf22@cam.ac.uk)
00018
00019 Permission is hereby granted, free of charge, to any person obtaining a
00020 copy of this software and associated documentation files (the
00021 "Software"), to deal in the Software without restriction, including
00022 without limitation the rights to use, copy, modify, merge, publish,
00023 distribute, sublicense, and/or sell copies of the Software, and to
00024 permit persons to whom the Software is furnished to do so, subject to
00025 the following conditions:
00026
00027 The above copyright notice and this permission notice shall be included
00028 in all copies or substantial portions of the Software.
00029
00030 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
00031 OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
00032 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
00033 IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
00034 CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
00035 TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
00036 SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00037
00038 */
00039
00040 #ifndef __ALIGNMENT_HEADER_FILE__
00041
00042 #define __ALIGNMENT_HEADER_FILE__
00043
00044 typedef float cawalign_fp;
00045
00046 cawalign_fp AlignStrings( char const * r_str
00047 , char const * q_str
00048 , const long _r_len
00049 , const long _q_len
00050 , char * & r_res
00051 , char * & q_res
00052 , long * char_map
00053 , const cawalign_fp * cost_matrix
00054 , const long cost_stride
00055 , const char gap
00056 , cawalign_fp open_insertion
00057 , cawalign_fp extend_insertion
00058 , cawalign_fp open_deletion
```

```

00059         , cawalign_fp extend_deletion
00060         , cawalign_fp miscall_cost
00061         , const bool do_local
00062         , const bool do_affine
00063         , const bool do_codon
00064         , const long char_count
00065         , const cawalign_fp * codon3x5
00066         , const cawalign_fp * codon3x4
00067         , const cawalign_fp * codon3x2
00068         , const cawalign_fp * codon3x1
00069         , const bool do_true_local = false
00070         , const bool report_ref_insertions = true
00071         , cawalign_fp* score_matrix_cache = nullptr
00072         , cawalign_fp* insertion_matrix_cache = nullptr
00073         , cawalign_fp* deletion_matrix_cache = nullptr
00074     );
00075
00076 cawalign_fp LinearSpaceAlign(    const char * s1          // first string
00077                                , const char * s2          // second string
00078                                , const long s1L
00079                                , const long s2L
00080                                , long* cmap              // char -> position in scoring matrix mapper
00081                                , const cawalign_fp * ccost // NxN matrix of edit distances on
                                characters
00082                                , const long costD
00083                                , cawalign_fp gopen       // the cost of opening a gap in sequence 1
00084                                , cawalign_fp gextend      // the cost of extending a gap in sequence 1
                                (ignored unless doAffine == true)
00085                                , cawalign_fp gopen2      // the cost of opening a gap in sequence 2
00086                                , cawalign_fp gextend2    // the cost of opening a gap in sequence 2
                                (ignored unless doAffine == true)
00087                                , bool doLocal            // ignore prefix and suffix gaps
00088                                , bool doAffine           // use affine gap penalties
00089                                , long * ops              // edit operations for the optimal alignment
00090                                , cawalign_fp scoreCheck // check the score of the alignment
00091                                , long from1
00092                                , long to1
00093                                , long from2
00094                                , long to2
00095                                , cawalign_fp ** buffer  // matrix storage,
00096                                , char parentGapLink
00097                                , char * ha
00098                                );
00099
00100 #endif

```

5.2 argparse.hpp

```

00001
00002 #ifndef ARGPARSE_H
00003 #define ARGPARSE_H
00004
00005 #include <stdio.h>
00006 #include <configparser.hpp>
00007 // argument defaults
00008
00009 #define PROGNAME          "cawalign"
00010 #define DEFAULT_DATA_TYPE nucleotide
00011 #define DEFAULT_REFERENCE "HXB2_pol"
00012 #define DEFAULT_SCORING  "Nucleotide-BLAST"
00013 #define DEFAULT_SPACE    quadratic
00014 #define DEFAULT_LOCAL_TYPE trim
00015 #define DEFAULT_OUTPUT_FORMAT refmap
00016 #define DEFAULT_RC_TYPE  none
00017
00018 #ifndef VERSION_NUMBER
00019     #define VERSION_NUMBER "0.0.1"
00020 #endif
00021
00022 #ifndef LIBRARY_PATH
00023     #define LIBRARY_PATH  "/usr/local/shares/cawalign/"
00024 #endif
00025
00026 #define SCORES_SUBPATH "scoring"
00027 #define REF_SUBPATH   "references"
00028
00029 namespace argparse
00030 {
00031
00032     enum data_t {
00033         nucleotide,
00034         codon,
00035         protein
00036     };

```

```

00036     };
00037
00038     enum local_t {
00039         trim,
00040         global,
00041         local
00042     };
00043
00044     enum space_t {
00045         quadratic,
00046         linear
00047     };
00048
00049     enum out_format_t {
00050         refmap,
00051         realign,
00052         pairwise
00053     };
00054
00055     enum rc_t {
00056         none,
00057         silent,
00058         annotated
00059     };
00060
00061     class args_t {
00062     public:
00063
00064         FILE          * output,
00065                     * reference,
00066                     * input;
00067
00068         ConfigParser * scores;
00069
00070         data_t        data_type;
00071         local_t       local_option;
00072         space_t       space_type;
00073         out_format_t out_format;
00074         rc_t          reverse_complement;
00075
00076         bool          quiet;
00077         bool          affine;
00078         bool          include_reference;
00079
00080
00081         args_t( int, const char ** );
00082         ~args_t();
00083
00084     private:
00085         void parse_input      ( const char * );
00086         void parse_reference  ( const char * );
00087         void parse_output     ( const char * );
00088         void parse_scores     ( const char * );
00089         void parse_quiet      ( void );
00090         void parse_affine     ( void );
00091         void parse_include_ref( void );
00092         void parse_rc         ( const char * );
00093         void parse_space_t    ( const char * );
00094         void parse_data_t     ( const char * );
00095         void parse_local_t    ( const char * );
00096         void parse_out_format_t( const char * );
00097
00098     };
00099
00100     void ERROR_NO_USAGE ( const char * msg, ... );
00101 }
00102
00103 #endif // ARGPARSE_H

```

5.3 configparser.hpp

```

00001 // Copyright (c) 2018 Daniel Zilles
00002 //
00003 // Permission is hereby granted, free of charge, to any person obtaining a copy
00004 // of this software and associated documentation files (the "Software"), to deal
00005 // in the Software without restriction, including without limitation the rights
00006 // to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00007 // copies of the Software, and to permit persons to whom the Software is
00008 // furnished to do so, subject to the following conditions:
00009 //
00010 // The above copyright notice and this permission notice shall be included in all
00011 // copies or substantial portions of the Software.
00012 //

```

```

00013 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00014 // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00015 // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00016 // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00017 // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00018 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00019 // SOFTWARE.
00020
00021 #ifndef CONFIGPARSER_HPP
00022 #define CONFIGPARSER_HPP
00023
00024 #include <string>
00025 #include <vector>
00026 #include <map>
00027 #include <iostream>
00028 #include <sstream>
00029
00030 typedef std::map<std::string, std::vector<std::string> > configList;
00031
00032 class ConfigParser {
00033
00034 public:
00035     ConfigParser(std::ifstream& configFile);
00036
00037     template<typename T>
00038     T aConfig(std::string section, std::string name, size_t pos = 0);
00039     template<typename T>
00040     std::vector<T> aConfigVec(std::string section, std::string name);
00041
00042 private:
00043     static void handleMissingKey (std::string);
00044     configList mConfigurations;
00045 };
00046
00047 };
00048
00049 template <>
00050 bool ConfigParser::aConfig<bool>(std::string section, std::string name, size_t pos);
00051
00052 template <typename T>
00053 T ConfigParser::aConfig(std::string section, std::string configName, size_t pos) {
00054
00055     T tmp;
00056
00057     const auto& mConfigRef = mConfigurations;
00058     auto search = mConfigRef.find(section + " - " + configName);
00059
00060     if (search == mConfigRef.end()) {
00061         handleMissingKey (std::string("Could not find required configuration section ") + section +
00062             std::string(" key ") + configName);
00063     }
00064
00065     std::istringstream iss(search->second[0]);
00066
00067     if (search->second[0].find( "0x" ) != std::string::npos)
00068         iss >> std::hex >> tmp;
00069     else
00070         iss >> std::dec >> tmp;
00071
00072     return tmp;
00073 }
00074
00075 template <>
00076 std::vector<bool> ConfigParser::aConfigVec<bool>(std::string section, std::string name);
00077
00078 template <typename T>
00079 std::vector<T> ConfigParser::aConfigVec(std::string section, std::string configName) {
00080
00081     const auto& mConfigRef = mConfigurations;
00082     auto search = mConfigRef.find(section + " - " + configName);
00083
00084     if (search == mConfigRef.end()) {
00085         handleMissingKey (std::string("Could not find required configuration section ") + section +
00086             std::string(" key ") + configName);
00087     }
00088
00089     std::vector<T> tmp(search->second.size());
00090     for (unsigned i = 0; i < search->second.size(); i++) {
00091
00092         std::istringstream iss(search->second[i]);
00093
00094         if (search->second[i].find( "0x" ) != std::string::npos)
00095             iss >> std::hex >> tmp[i];
00096         else
00097             iss >> std::dec >> tmp[i];
00098     }
00099 }

```

```

00098     }
00099     return tmp;
00100 }
00101 #endif

```

5.4 scoring.hpp

```

00001 #ifndef SCORING_H
00002 #define SCORING_H
00003
00004 #include <iostream>
00005 #include "alignment.h"
00006 #include "argparse.hpp"
00007 #include "tn93_shared.h"
00008
00009 using namespace std;
00010 using namespace argparse;
00011
00012
00013 class CawalignSimpleScores {
00014     public:
00015         CawalignSimpleScores (
00016             const char * _alphabet,
00017             const cawalign_fp * _scoring_matrix,
00018             const cawalign_fp _open_gap_reference,
00019             const cawalign_fp _open_gap_query,
00020             const cawalign_fp _extend_gap_reference,
00021             const cawalign_fp _extend_gap_query
00022         );
00023
00024         CawalignSimpleScores (ConfigParser*);
00025         CawalignSimpleScores (void) {D=0;};
00026         virtual ~CawalignSimpleScores (void) {};
00027
00028         StringBuffer          alphabet;
00029         /*
00030          ordered characters that are included in the scoring matrix
00031          */
00032         unsigned int          D;
00033         /* the number of characters in the string
00034          */
00035         long                  char_map [255];
00036         /*
00037          for each ASCII character, this will map the character to the corrspeing entry the scoring
00038          matrix
00039          all characters NOT in `alphabet` get mapped to index D (the 'not defined' character)
00040          */
00041         VectorFP              scoring_matrix;
00042         /*
00043          A (D+1)x(D+1) scoring matrix where element (i,j) gives the score of matching (or
00044          mis-matching)
00045          the D-th row/column is for matchign a character NOT in the alphabet
00046          While generally symmetric, an asymmetric matrix can also be meaningful if there is
00047          some reason to have
00048          substitutions in reference/query weighted differently
00049          */
00049         cawalign_fp           open_gap_reference,
00050                             open_gap_query,
00051                             extend_gap_query,
00052                             extend_gap_reference;
00053
00054         char                  gap_char;
00055         /* gap open and extend character*/
00056
00057         void                  _init_alphabet (long not_found = -1);
00058
00059 };
00060
00061 class CawalignCodonScores : public CawalignSimpleScores {
00062     public:
00063
00064         CawalignCodonScores (ConfigParser*);
00065         virtual ~CawalignCodonScores (void) {};
00066
00067         /* compute how many nucleotides are different between the two codons encoded as 0-63 integers
00068          */
00068         static int nucleotide_diff (long, long);
00069
00070         Vector          translation_table;
00071         /* codon (0-63 index) to single letter amino-acid code translation table
00072          */
00072
00073         // partial score tables

```

```

00074     VectorFP          s3x1,
00075                     s3x2,
00076                     s3x4,
00077                     s3x5;
00078
00079     // the cost of introducing frameshifts
00080     cawalign_fp        frameshift_cost,
00081     // the penalty for synonymous substitutions, per nucleotide change
00082                     synonymous_penalty;
00083
00084     // ordered amino-acid scoring tables
00085     StringBuffer      amino_acids;
00086
00087     int                stop_codon_index;
00088     int                mismatch_index;
00089
00090
00091
00092 };
00093
00094
00095 extern const char    kNucleotideAlphabet[];
00096 extern const cawalign_fp kNucScoring[];
00097
00098 #endif

```

5.5 stringBuffer.h

```

00001 #ifndef __STRINGBUFFER__
00002 #define __STRINGBUFFER__
00003
00004 #include "alignment.h"
00005
00006 // _____
00007
00008 class StringBuffer {
00009
00010     char *sData;
00011     unsigned long sLength, saLength;
00012
00013 public:
00014     StringBuffer(void);
00015     ~StringBuffer(void);
00016
00017     char *getString(void) const { return sData; }
00018     void appendChar(const char);
00019     void appendBuffer(const char *, const long = -1);
00020     void resetString(void);
00021     void swap(StringBuffer &);
00022     unsigned long length(void) const { return sLength; }
00023     void reset_length(unsigned long newL) {
00024         if (newL < sLength) {
00025             sLength = newL;
00026         }
00027     }
00028
00029     char setChar(const long i, const char c) {
00030         char oc = sData[i];
00031         sData[i] = c;
00032         return oc;
00033     }
00034
00035     char getChar(const long i) const { return sData[i]; }
00036     void flip (void);
00037     void detach (void) { sData = nullptr; }
00038
00039     static long sbDefaultLength, sbDefaultBoost;
00040 };
00041
00042 // _____
00043
00044 class VectorFP {
00045
00046     cawalign_fp *vData;
00047
00048     unsigned long vLength, vaLength;
00049
00050 public:
00051     VectorFP(void);
00052     ~VectorFP(void);
00053
00054     void appendValue(const cawalign_fp);
00055     void appendValues(const cawalign_fp*, long);

```



```

00056 void storeValue(const cawlign_fp, const unsigned long);
00057 cawlign_fp value(const long idx) { return vData[idx]; }
00058 unsigned long length(void) const { return vLength; }
00059 const cawlign_fp * values (void) {return vData;}
00060 cawlign_fp * rvalues (void) {return vData;}
00061
00062 static long vDefaultLength, vDefaultBoost;
00063 };
00064
00065 // _____
00066
00067 class Vector {
00068     long *vData;
00069
00070     unsigned long vLength, vaLength;
00071
00072 public:
00073     Vector(void);
00074     ~Vector(void);
00075
00076     void appendValue(const long);
00077     void appendVector(const Vector &);
00078     long extractMin(VectorFP &);
00079     void resetVector(void);
00080     void remove(const unsigned long);
00081     void storeValue(const long, const unsigned long);
00082     void storeVector(const Vector &, const unsigned long);
00083     void sort(void);
00084     void swap(Vector &);
00085     long value(const long idx) const;
00086     unsigned long length(void) const { return vLength; }
00087
00088     static long vDefaultLength, vDefaultBoost;
00089 };
00090
00091 #endif
00092 #endif

```

5.6 tn93_shared.h

```

00001 #ifndef __TN93SHARED__
00002 #define __TN93SHARED__
00003
00004 #include <iostream>
00005 #include <cstdlib>
00006 #include <cstdio>
00007 #include <iomanip>
00008 #include <math.h>
00009 #include <string.h>
00010 #include <unistd.h>
00011 #include <climits>
00012 #include "stringBuffer.h"
00013
00014 using namespace std;
00015
00016 #define RESOLVE_A 0x01
00017 #define RESOLVE_C 0x02
00018 #define RESOLVE_G 0x04
00019 #define RESOLVE_T 0x08
00020
00021
00022 #define RESOLVE 0
00023 #define AVERAGE 1
00024 #define SKIP 2
00025 #define GAPMM 3
00026 #define SUBSET 4
00027 #define MISMATCH 5
00028 #define INFORMATIVE 6
00029
00030 #define RAND_RANGE 0xffffffffUL /* Maximum value returned by genrand_int32 */
00031
00032 #define MIN(a,b) (a) < (b) ? (a) : (b)
00033 #define MAX(a,b) (a) > (b) ? (a) : (b)
00034
00035 struct sequence_gap_structure {
00036     long first_nongap,
00037         last_nongap,
00038         resolved_start,
00039         resolved_end;
00040
00041     sequence_gap_structure (void) {
00042         first_nongap = LONG_MAX;

```

```

00044     last_nongap      = 0L;
00045     resolved_start = 0L;
00046     resolved_end   = 0L;
00047 }
00048
00049 };
00050
00051 void init_genrand(unsigned long s);
00052 unsigned long genrand_int32(void);
00053 double computeTN93 (const char * s1, const char *s2, const unsigned long L, const char
matchMode, const long * randomize, const long min_overlap, unsigned long* = NULL, const double = 0.0,
const unsigned long cnt = 0, const long count1 = 1, const long count2 = 1, const
sequence_gap_structure * = NULL, const sequence_gap_structure * = NULL);
00054
00055 long computeDifferences (const char * s1,
00056                        const char *s2,
00057                        const unsigned long L,
00058                        const char matchMode,
00059                        Vector& storage,
00060                        const sequence_gap_structure * = NULL,
00061                        const sequence_gap_structure * = NULL);
00062
00063
00064 long stringLength (Vector& lengths, unsigned long index);
00065 char* stringText (const StringBuffer& strings, const Vector& lengths, unsigned long index);
00066 void addASequenceToList (StringBuffer& sequences, Vector& seqLengths, long &firstSequenceLength,
StringBuffer& names, Vector& nameLengths);
00067 int readFASTA (FILE* F, char& automatonState, StringBuffer &names, StringBuffer& sequences, Vector
&nameLengths, Vector &seqLengths, long& firstSequenceLength, bool oneByOne = false, Vector*
sequenceInstances = NULL, char sep = ':', double include_prob = 1.0, bool show_progress = false);
00068 void dump_sequence_fasta (unsigned long index, FILE* output, long firstSequenceLength, double * d =
NULL, bool = false, unsigned long from = 0L, unsigned long to = 0L);
00069 void initAlphabets(bool = false, char * = NULL, bool id_map = false);
00070 void merge_two_sequences (const char* source, char* target, const long sequence_length);
00071 long perfect_match (const char* source, char* target, const long sequence_length);
00072 void dump_fasta (const char* source, const long sequence_length, FILE* output, bool newln = true, bool
= false, unsigned long from = 0L, unsigned long to = 0L);
00073
00074 int reverseComplement (StringBuffer& sequence, unsigned long from, unsigned long to);
00075 struct sequence_gap_structure describe_sequence (const char* source, const unsigned long
sequence_length, const unsigned long char_count = 4UL);
00076
00077 const long * resolve_char (unsigned char, bool = false, bool = true);
00078 const double resolution_count (unsigned char, bool = false);
00079 const char unmap_char (unsigned char, bool = false);
00080 inline void unpack_difference (long diff, long& location, unsigned& alt) {
00081     location = diff >> 8;
00082     alt = diff & 0xff;
00083 }
00084
00085
00086 extern StringBuffer names,
00087     sequences;
00088
00089 extern unsigned char * resolveTheseAmbigs;
00090
00091 extern double resolve_fraction;
00092
00093 extern Vector nameLengths,
00094     seqLengths,
00095     workingNodes,
00096     nodeParents;
00097
00098 extern VectorFP distanceEstimates;
00099 extern const double resolutionsCount [];
00100 extern char validFlags[];
00101
00102 #endif

```

Index

- [_init_alphabet](#)
 - [CawalignSimpleScores, 11](#)
- [~StringBuffer](#)
 - [StringBuffer, 14](#)
- [~Vector](#)
 - [Vector, 16](#)
- [~VectorFP](#)
 - [VectorFP, 20](#)
- [~args_t](#)
 - [argparse::args_t, 8](#)
- [aConfig](#)
 - [ConfigParser, 12](#)
- [aConfigVec](#)
 - [ConfigParser, 13](#)
- [appendBuffer](#)
 - [StringBuffer, 14](#)
- [appendChar](#)
 - [StringBuffer, 15](#)
- [appendValue](#)
 - [Vector, 16](#)
 - [VectorFP, 20](#)
- [appendValues](#)
 - [VectorFP, 20](#)
- [appendVector](#)
 - [Vector, 17](#)
- [argparse::args_t, 7](#)
 - [~args_t, 8](#)
 - [args_t, 7](#)
- [args_t](#)
 - [argparse::args_t, 7](#)
- [CawalignCodonScores, 8](#)
 - [CawalignCodonScores, 9](#)
 - [nucleotide_diff, 9](#)
- [CawalignSimpleScores, 10](#)
 - [_init_alphabet, 11](#)
 - [CawalignSimpleScores, 10, 11](#)
- [ConfigParser, 12](#)
 - [aConfig, 12](#)
 - [aConfigVec, 13](#)
 - [ConfigParser, 12](#)
- [extractMin](#)
 - [Vector, 17](#)
- [flip](#)
 - [StringBuffer, 15](#)
- [nucleotide_diff](#)
 - [CawalignCodonScores, 9](#)
- [remove](#)
 - [Vector, 17](#)
- [resetString](#)
 - [StringBuffer, 15](#)
- [resetVector](#)
 - [Vector, 17](#)
- [sequence_gap_structure, 13](#)
- [sort](#)
 - [Vector, 18](#)
- [src/alignment.h, 23](#)
- [src/argparse.hpp, 24](#)
- [src/configparser.hpp, 25](#)
- [src/scoring.hpp, 27](#)
- [src/stringBuffer.h, 28](#)
- [src/tn93_shared.h, 29](#)
- [storeValue](#)
 - [Vector, 18](#)
 - [VectorFP, 20](#)
- [storeVector](#)
 - [Vector, 18](#)
- [StringBuffer, 14](#)
 - [~StringBuffer, 14](#)
 - [appendBuffer, 14](#)
 - [appendChar, 15](#)
 - [flip, 15](#)
 - [resetString, 15](#)
 - [StringBuffer, 14](#)
 - [swap, 15](#)
- [swap](#)
 - [StringBuffer, 15](#)
 - [Vector, 18](#)
- [value](#)
 - [Vector, 19](#)
- [Vector, 16](#)
 - [~Vector, 16](#)
 - [appendValue, 16](#)
 - [appendVector, 17](#)
 - [extractMin, 17](#)
 - [remove, 17](#)
 - [resetVector, 17](#)
 - [sort, 18](#)
 - [storeValue, 18](#)
 - [storeVector, 18](#)
 - [swap, 18](#)
 - [value, 19](#)
 - [Vector, 16](#)
- [VectorFP, 19](#)
 - [~VectorFP, 20](#)

[appendValue](#), [20](#)
[appendValues](#), [20](#)
[storeValue](#), [20](#)
[VectorFP](#), [20](#)