Pessum Documentation

Release 2.0

Arden Rasmussen

Jul 17, 2017

CONTENTS:

Pessum is a base library for backend of programs. It provides a simple system for logging. These logs can be saved to a file, or handled in the program imediatly (*Logging*). Pessum also provides a simple system fo saving and loading basic variables from a external file (*Data*).

CHAPTER ONE

DATA

The data handling aspect of Pessum is primarily used to save data that is changed must be maintained external to the program, and when the program ends, the data can be saved to a different file.

1.1 Functions

1.1.1 Load

std::vector<*DataPoint*> **`Load`**(std::string *file*)

`file`	Path to file to read data from

Reads data from a specified file, and returns a vector of the data in *`DataPoint`*. The data will be converted into any basic types that it can be converted to (`int`, `double`, `bool`, `std::string`).

Return: Vector of *`DataPoint`* containing information read from `file`.

1.1.2 Save

void **`Save`**(std::string *file*, std::vector<*DataPoint*> *data*)

`file`	Path to file to save data to
`data`	Data to save to file

Saves information from `data` to `file`.

CHAPTER

TWO

DATA POINT

The DataPoint class is a simple class that can contain a value for `int`, `double`, `std::string`, or `bool`. This is used for when the type of the value is not known to the program. The type can then be determined through the use of `type`.

2.1 Enumerators

2.1.1 PessumDataType

enum PessumDataType

Used to define the type of a *DataPoint*.

PESSUM_NONE	0
PESSUM_INT	1
PESSUM_DOUBLE	2
PESSUM_STR	3
PESSUM_BOOL	4

2.2 Classes

2.2.1 DataPoint

class DataPoint

Class structure to contain data of one of several different base types. The data in a *DataPoint* class can be `int`, `double`, `std::string`, or `bool`.

```
class DataPoint{
 public:
  explicit DataPoint();
  explicit DataPoint(int value);
  explicit DataPoint(double value);
  explicit DataPoint(std::string value);
  explicit DataPoint(const char* value);
  explicit DataPoint(bool value);

  void operator=(int value);
  void operator=(double value);
  void operator=(std::string value);
```

```
    void operator=(const char* value);
    void operator=(bool value);

    operator int();
    operator double();
    operator std::string();
    operator bool();

    int int_value, type;
    double double_value;
    std::string string_value;
    bool bool_value;
};
```

Constructors

DataPoint(void)

`DataPoint::`**`DataPoint`**`()`

Default constructor, sets all values to default, and `type` to `PESSUM_NONE`.

DataPoint(int)

`DataPoint::`**`DataPoint`**`(`int *value*`)`

`value`	Integer value to use as set value

Constructor that sets the `type` to `PESSUM_INT`, and sets `int_value` to `value`.

DataPoint(double)

`DataPoint::`**`DataPoint`**`(`double *value*`)`

`value`	Double value to use as set value

Constructor that sets the `type` to `PESSUM_DOUBLE`, and sets `double_value` to `value`.

DataPoint(std::string)

`DataPoint::`**`DataPoint`**`(`std::string *value*`)`

`value`	String value to use as set value

Constructor that sets the `type` to `PESSUM_STR`, and sets `string_value` to `value`.

DataPoint(const char*)

`DataPoint::`**`DataPoint`**`(`**const** char **value*`)`

`value`	String value to use as set value

Constructor that sets the `type` to `PESSUM_STR`, and sets `string_value` to `value`.

DataPoint(bool)

`DataPoint::`**`DataPoint`**`(`bool *value*`)`

`value`	Boolian value to use as set value

Constructor that sets the `type` to `PESSUM_BOOL`, and sets `bool_value` to `value`.

Operators

operator=(int)

`DataPoint::`**`operator=`**`(`int *value*`)`

`value`	Double vlue to use as set value

Operator that sets the `type` to `PESSUM_INT`, and sets `int_value` to `value`.

operator=(double)

`DataPoint::`**`operator=`**`(`double *value*`)`

`value`	Double vlue to use as set value

Operator that sets the `type` to `PESSUM_DOUBLE`, and sets `double_value` to `value`.

operator=(std::string)

`DataPoint::`**`operator=`**`(`std::string *value*`)`

`value`	Double vlue to use as set value

Operator that sets the `type` to `PESSUM_STR`, and sets `string_value` to `value`.

operator=(const char*)

`DataPoint::`**`operator=`**`(`**const** char **value*`)`

`value`	Double vlue to use as set value

Operator that sets the `type` to `PESSUM_STR`, and sets `string_value` to `value`.

operator=(bool)

`DataPoint::`**`operator=`**`(`bool *value*`)`

`value`	Double vlue to use as set value

Operator that sets the `type` to `PESSUM_BOOL`, and sets `bool_value` to `value`.

operator int()

`DataPoint::`**`operator int`**`()`

Return: `int_value`

operator double()

`DataPoint::`**`operator double`**`()`

Return: `double_value`

operator std::string()

`DataPoint::`**`operator std::string`**`()`

Return: `string_value`

operator bool()

`DataPoint::`**`operator bool`**`()`

Return: `bool_value`

2.3 Functions

2.3.1 Make_DataPoint

DataPoint **`Make_DataPoint`**`(`std::string *str*`)`

`str`	String to convert to *`DataPoint`*

This function takes a string, and reads it. If the string can be converted into some other type (`int`, `double`, or `bool`), it is converted. Then everything is saved into a *`DataPoint`*.

Return: *`DataPoint`* containing the reducd type of the string data.

CHAPTER

THREE

LOGGING

The logging functionality of Pessum, is very simple. It permits logs entries to be added to a set of global log entries for that occurance of the program. These log entries can then be handled by provided functions when they are added, or they can be retreaved later with one of several log retreval functions. The entire list of log entries can also be saved to an external file for review after program termination.

3.1 Enumerators

3.1.1 LogOptions

enum `LogOptions`

Used to specify a logging option to set using *`SetLogOption()`*.

TIME_STAMP	0
DATE_STAMP	1

3.1.2 LogType

enum `LogType`

Used to define the type/importance of the log call.

ERROR	0
WARNING	1
TRACE	2
DEBUG	3
SUCCESS	4
INFO	5
DATA	6
NONE	7

3.2 Functions

3.2.1 Log

void **`Log`** (int *type*, std::string *msg*, std::string *func*, . . .)

`type`	Type of log entry from *LogType*
`msg`	Format string of log entry
`func`	The name of the function creating the log entry
`...`	Additional formating args for `msg`

Core function for all logging output, `msg` is a format string with additional arguments as needed from `...`. Formated string and log type are saved to *global_logs*.

3.2.2 GetLogSize

int **GetLogSize**()

Gets the length of the *global_logs*.

Return: Length of *global_logs* as an integer.

3.2.3 ClearLogs

void **ClearLogs**()

Clears all log entries from *global_logs*.

3.2.4 GetLog

GetLog

std::string **GetLog**(int *type*)

`type`	The type of log entry to find and retrieve

Gets last log entry of specified type with formated string.

Return: Formated string of log entry.

FGetLog

std::pair<int, std::string> **FGetLog**(int *type*)

`type`	The type of log entry to find and retrieve

Gets last log entry of specified type with log type and formated string.

Return: Pair of log type and formated string of log entry.

IGetLog

std::string **IGetLog** (int *index*)

`index`	The index of the log entry from *global_logs*

Gets log entry of specified index with formated string.

Return: Formated string of log entry.

IFGetLog

std::string **IFGetLog** (int *index*)

`index`	The index of the log entry from *global_logs*

Gets log entry of specified index with log type formated string.

Return: Pair of log type and formated string of log entry.

VGetLog

std::vector<std::string> **VGetLog** (int *start*, int *end*)

`start`	The first index value from *global_logs*
`end`	The last index value from *global_logs*

Get a set of log entries between (inclusive) specified start and end index with formated string.

Return: Vector of strings of log entries.

VFGetLog

std::vector<std::string> **VFGetLog** (int *start*, int *end*)

`start`	The first index value from *global_logs*
`end`	The last index value from *global_logs*

Get a set of log entries between (inclusive) specified start and end index with log type and formated string.

Return: Vector of pairs of log type and formated stirng of log entry.

3.2.5 Set Log Options

SetLogHandle[1/2]

void **SetLogHandle** (void (**handle*)) std::pair<int, std::string>

`handle`	Pointer to function with return of void and args of a pair of int and string

Sets *log_handle_full* to given pointer.

SetLogHandle[2/2]

void **SetLogHandle** (void (**handle*)) std::string

`handle`	Pointer to function with return of void and args of a string

Sets *log_handle* to given pointer.

SetLogOption

void **SetLogOption** (int *option*, int *setting*)

`option`	Value for option from *LogOptions*
`setting`	Value to set for `option`

Sets `option` of *options* to `setting`.

3.2.6 GetTypeStr

std::string **GetTypeStr** (int *type*)

`type`	Type from *LogType* to convert to string

Determines that string corisponding to `type` value.

Return: String corisponding to `type` value.

3.2.7 SaveLog

void **SaveLog** (std::string *file*)

`file`	Path to file save log into

Saves the log entries from *global_logs* to specified file.

3.3 Variables

3.3.1 options

std::array<int, 2> **`options`**

Array storing values for the different log options set in *`SetLogOption()`*.

Note: Variable is private.

3.3.2 global_logs

std::vector<std::pair<int, std::string>> **`global_logs`**

All log calls are saved to this vector, and can be retrieved later with any form of the *`GetLog()`* functions.

Note: Variable is private.

3.3.3 log_handle_full

void (***`log_handle_full`**) (std::pair<int, std::string>)

Pointer to function for handling log calls with full log information. This function is called with every log entry added through *`Log()`*.

Note: Variable is private.

3.3.4 log_handle

void (***`log_handle`**) (std::string)

Pointer to function for handling logs with only formated string This funtion is called with every log entry added through *`Log()`*.

Note: Variable is private.

CHAPTER FOUR

DOWNLOADS

4.1 Documentation

4.1.1 PDF

4.2 Project

4.2.1 ZIP

Pessum (master).zip

INDEX