0pt

Reference Manual

Generated by Doxygen 1.2.18

Tue Aug 26 00:00:04 2003

Contents

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Namespace Documentation

4.1 std Namespace Reference

5 Data Structure Documentation

5.1 Client_Acceptor Class Reference

```
#include <client_acceptor.h>
```

Public Types

- typedef Client_Acceptor_Base inherited
- enum concurrency_t { single_threaded_, thread_per_connection_, thread_pool_ }

Public Methods

- Client_Acceptor (Contest ∗myContest, concurrency_t concurrency=thread_pool_)
- Client_Acceptor (Contest ∗myContest, Thread_Pool &thread_pool)
- ~Client_Acceptor (void)
- int open (const ACE_INET_Addr &addr, ACE_Reactor ∗reactor, int pool_size=Thread_Pool::default_pool_size_)
- int close (void)
- int concurrency (void)
- Thread_Pool ∗ thread_pool (void)
- int thread_pool_is_private (void)
- int make_svc_handler (Client_Handler ∗&ch)

  Register the service handler to the acceptor.

- void setContest (Contest ∗curContest)

  Set parent contest.

- Contest ∗ getContest ()

  Get parent contest.

Protected Attributes

- int concurrency_
- Thread_Pool private_thread_pool_
- Thread_Pool & the_thread_pool_
- Contest ∗ parentContest

5.1.1 Member Typedef Documentation

5.1.1.1 typedef Client_Acceptor_Base Client_Acceptor::inherited

Definition at line 54 of file client_acceptor.h.

5.1.2 Member Enumeration Documentation

5.1.2.1 enum Client_Acceptor::concurrency_t

Now that we have more than two strategies, we need more than a boolean to tell us what we're using. A set of enums is a good choice because it allows us to use named values. Another option would be a set of static const integers.

Enumeration values:

single_threaded_

thread_per_connection_

thread_pool_

Definition at line 60 of file client_acceptor.h.

5.1.3 Constructor & Destructor Documentation

5.1.3.1 Client_Acceptor::Client_Acceptor (Contest ∗ *myContest*, concurrency_t *concurrency* = thread_pool_)

The default constructor allows the programmer to choose the concurrency strategy. Since we want to focus on thread-pool, that's what we'll use if nothing is specified.

Definition at line 7 of file client_acceptor.cpp.

5.1.3.2 Client_Acceptor::Client_Acceptor (Contest ∗ *myContest*, Thread_Pool & *thread_pool*)

Another option is to construct the object with an existing thread pool. The concurrency strategy is pretty obvious at that point.

Definition at line 16 of file client_acceptor.cpp.

5.1.3.3 Client_Acceptor::∼Client_Acceptor (void)

Our destructor will take care of shutting down the thread-pool if applicable.

Definition at line 26 of file client_acceptor.cpp.

References concurrency(), thread_pool(), thread_pool_, and thread_pool_is_private().

5.1.4 Member Function Documentation

5.1.4.1 int Client_Acceptor::close (void)

Close ourselves and our thread pool if applicable

Definition at line 51 of file client_acceptor.cpp.

References concurrency(), Thread_Pool::stop(), thread_pool(), thread_pool_, and thread_pool_is_private().

5.1.4.2 int Client_Acceptor::concurrency (void) `[inline]`

What is our concurrency strategy?

Definition at line 91 of file client_acceptor.h.

References concurrency_.

Referenced by close(), Client_Handler::concurrency(), open(), and ~Client_Acceptor().

5.1.4.3 Contest* Client_Acceptor::getContest () [inline]

Definition at line 123 of file client_acceptor.h.

References parentContest.

5.1.4.4 int Client_Acceptor::make_svc_handler (Client_Handler *& *ch*)

Registers the Client_Handler Reactor to the Acceptor's reactor, while also setting the parentContest object to this->getContest().

Definition at line 63 of file client_acceptor.cpp.

References Client_Handler::setContest().

5.1.4.5 int Client_Acceptor::open (const ACE_INET_Addr & *addr*, ACE_Reactor * *reactor*, int *pool_size* = Thread_Pool::default_pool_size_)

Open ourselves and register with the given reactor. The thread pool size can be specified here if you want to use that concurrency strategy.

Definition at line 38 of file client_acceptor.cpp.

References concurrency(), Thread_Pool::start(), thread_pool(), thread_pool_, and thread_pool_is_private().

Referenced by Contest::initThreads().

5.1.4.6 void Client_Acceptor::setContest (Contest * *curContest*) [inline]

Definition at line 120 of file client_acceptor.h.

References parentContest.

5.1.4.7 Thread_Pool* Client_Acceptor::thread_pool (void) [inline]

Give back a pointer to our thread pool. Our Client_Handler objects will need this so that their handle_input() methods can put themselves into the pool. Another alternative would be a globally accessible thread pool. ACE_Singleton<> is a way to achieve that.

Definition at line 101 of file client_acceptor.h.

References the_thread_pool_.

Referenced by close(), open(), Client_Handler::thread_pool(), and ~Client_Acceptor().

5.1.4.8 int Client_Acceptor::thread_pool_is_private (void) [inline]

Since we can be constructed with a Thread_Pool reference, there are times when we need to know if the thread pool we're using is ours or if we're just borrowing it from somebody else.

Definition at line 108 of file client_acceptor.h.

References private_thread_pool_, and the_thread_pool_.

Referenced by close(), open(), and ~Client_Acceptor().

5.1.5 Field Documentation

5.1.5.1 int Client_Acceptor::concurrency_ `[protected]`

Definition at line 127 of file client_acceptor.h.

Referenced by concurrency().

5.1.5.2 Contest∗ Client_Acceptor::parentContest `[protected]`

Definition at line 133 of file client_acceptor.h.

Referenced by getContest(), and setContest().

5.1.5.3 Thread_Pool Client_Acceptor::private_thread_pool_ `[protected]`

Definition at line 129 of file client_acceptor.h.

Referenced by thread_pool_is_private().

5.1.5.4 Thread_Pool& Client_Acceptor::the_thread_pool_ `[protected]`

Definition at line 131 of file client_acceptor.h.

Referenced by thread_pool(), and thread_pool_is_private().

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

- client_acceptor.h
- client_acceptor.cpp

5.2 Client_Handler Class Reference

`#include <client_handler.h>`

5.2.1 Detailed Description

Another feature of ACE_Svc_Handler is it's ability to present the ACE_Task<> interface as well. That's what the ACE_NULL_SYNCH parameter below is all about. That's beyond our scope here but we'll come back to it in the next tutorial when we start looking at concurrency options.

Definition at line 42 of file client_handler.h.

Public Types

- typedef ACE_Svc_Handler< ACE_SOCK_STREAM, ACE_NULL_SYNCH > inherited

Public Methods

- Client_Handler (void)

  Constructor...

- void destroy (void)
- int open (void ∗acceptor)

- int close (u_long flags=0)
- int handle_close (ACE_HANDLE handle, ACE_Reactor_Mask mask)
- int handle_input (ACE_HANDLE handle)
- void setContest (Contest ∗curContest)

  Sets the parentContest pointer with the given one.

- Contest ∗ getContest ()

  Returns the parentContest pointer.

Protected Methods

- int svc (void)
- int process (unsigned char ∗rdbuf, size_t rdbuf_len)
- ~Client_Handler (void)
- Client_Acceptor ∗ client_acceptor (void)
- void client_acceptor (Client_Acceptor ∗_client_acceptor)
- int concurrency (void)
- Thread_Pool ∗ thread_pool (void)

Protected Attributes

- ACE_Thread_Mutex mutex_

  A basic Thread_Mutex class that will ensure safe access to data.

- Contest ∗ parentContest

  Pointer to the parent Contest object;.

- unsigned char ∗ data

  Data to hold the connectionMsgBlock object.

- int srv_counter

  Counter to keep the active servers.

- Client_Acceptor ∗ client_acceptor_
- ACE_thread_t creator_

Private Types

- enum responseID { ID_CRC_OK = 1, ID_CRC_ERROR = -1 }

  Basic enum type for error checking.

5.2.2 Member Typedef Documentation

5.2.2.1 typedef ACE_Svc_Handler<ACE_SOCK_STREAM, ACE_NULL_SYNCH> Client_Handler::inherited

Definition at line 47 of file client_handler.h.

5.2.3 Member Enumeration Documentation

5.2.3.1 enum Client_Handler::responseID [private]

Enumeration values:

ID_CRC_OK

ID_CRC_ERROR

Definition at line 45 of file client_handler.h.

5.2.4 Constructor & Destructor Documentation

5.2.4.1 Client_Handler::Client_Handler (void)

Our constructor still doesn't really do anything. We simply initialize the acceptor pointer to "null" and get our current thread id. The static self() method of ACE_Thread will return you a thread id native to your platform.

Definition at line 21 of file client_handler.cpp.

References data, connectionMsgBlock::DATASIZE, and srv_counter.

5.2.4.2 Client_Handler::∼Client_Handler (void) [protected]

We don't really do anything in our destructor but we've declared it to be protected to prevent casual deletion of this object. As I said above, I really would prefer that everyone goes through the destroy() method to get rid of us.

Definition at line 29 of file client_handler.cpp.

References data.

5.2.5 Member Function Documentation

5.2.5.1 void Client_Handler::client_acceptor (Client_Acceptor ∗ *_client_acceptor*) [inline, protected]

And since you shouldn't access a member variable directly, neither should you set (mutate) it. Although it might seem silly to do it this way, you'll thank yourself for it later.

Definition at line 158 of file client_handler.h.

References client_acceptor_.

5.2.5.2 Client_Acceptor∗ Client_Handler::client_acceptor (void) [inline, protected]

When we get to the definition of Client_Handler we'll see that there are several places where we go back to the Client_Acceptor for information. It is generally a good idea to do that through an accesor rather than using the member variable directly.

Definition at line 150 of file client_handler.h.

References client_acceptor_.

Referenced by concurrency(), open(), and thread_pool().

5.2.5.3 int Client_Handler::close (u_long *flags* = 0)

When an ACE_Task<> object falls out of the svc() method, the framework will call the close() method. That's where we want to cleanup ourselves if we're running in either thread-per-connection or thread-pool mode.

Definition at line 89 of file client_handler.cpp.

References destroy().

5.2.5.4 int Client_Handler::concurrency (void) `[protected]`

The concurrency() accessor tells us the current concurrency strategy. It actually queries the Client_Acceptor for it but by having the accessor in place, we could change our implementation without affecting everything that needs to know.

Definition at line 39 of file client_handler.cpp.

References client_acceptor(), and Client_Acceptor::concurrency().

Referenced by handle_input(), and open().

5.2.5.5 void Client_Handler::destroy (void)

The destroy() method is our preferred method of destruction. We could have overloaded the delete operator but that is neither easy nor intuitive (at least to me). Instead, we provide a new method of destruction and we make our destructor protected so that only ourselves, our derivatives and our friends can delete us. It's a nice compromise.

Definition at line 80 of file client_handler.cpp.

References REMOVE_MASK.

Referenced by close().

5.2.5.6 Contest ∗ Client_Handler::getContest ()

Definition at line 250 of file client_handler.cpp.

References parentContest.

5.2.5.7 int Client_Handler::handle_close (ACE_HANDLE *handle*, ACE_Reactor_Mask *mask*)

When there is activity on a registered handler, the handle_input() method of the handler will be invoked. If that method returns an error code (eg – -1) then the reactor will invoke handle_close() to allow the object to clean itself up. Since an event handler can be registered for more than one type of callback, the callback mask is provided to inform handle_close() exactly which method failed. That way, you don't have to maintain state information between your handle_∗ method calls. The <handle> parameter is explained below... As a side-effect, the reactor will also invoke remove_handler() for the object on the mask that caused the -1 return. This means that we don't have to do that ourselves!

Definition at line 110 of file client_handler.cpp.

5.2.5.8 int Client_Handler::handle_input (ACE_HANDLE *handle*)

When we register with the reactor, we're going to tell it that we want to be notified of READ events. When the reactor sees that there is read activity for us, our handle_input() will be invoked. The _handleg provided is the handle (file descriptor in Unix) of the actual connection causing the activity. Since we're derived from ACE_Svc_Handler<> and it maintains it's own peer (ACE_SOCK_Stream) object, this is redundant for us. However, if we had been derived directly from ACE_Event_Handler, we may have chosen not to contain the peer. In that case, the <handle> would be important to us for reading the client's data.

Definition at line 135 of file client_handler.cpp.

References concurrency(), creator_, data, connectionMsgBlock::DATASIZE, Thread_Pool::enqueue(), process(), REGISTER_MASK, REMOVE_MASK, thread_pool(), and Client_Acceptor::thread_pool_.

5.2.5.9 int Client_Handler::open (void * *acceptor*)

Most ACE objects have an open() method. That's how you make them ready to do work. ACE_Event_Handler has a virtual open() method which allows us to create this overrride. ACE_Acceptor<> will invoke this method after creating a new Client_Handler when a client connects. Notice that the parameter to open() is a void*. It just so happens that the pointer points to the acceptor which created us. You would like for the parameter to be an ACE_Acceptor<>* but since ACE_Event_Handler is generic, that would tie it too closely to the ACE_Acceptor<> set of objects. In our definition of open() you'll see how we get around that.

Definition at line 54 of file client_handler.cpp.

References client_acceptor(), concurrency(), REGISTER_MASK, and Client_Acceptor::thread_per_connection_.

5.2.5.10 int Client_Handler::process (unsigned char * *rdbuf*, size_t *rdbuf_len*) [protected]

This has nothing at all to do with ACE. I've added this here as a worker function which I will call from handle_input(). That allows me to introduce concurrencly in later tutorials with a no changes to the worker function. You can think of process() as application-level code and everything elase as application-framework code.

Definition at line 200 of file client_handler.cpp.

References ID_CRC_ERROR, connectionMsgBlock::isValid(), Contest::logger, Logger::logMsg(), parentContest, Processor::process(), Contest::processor, and srv_counter.

Referenced by handle_input(), and svc().

5.2.5.11 void Client_Handler::setContest (Contest * *curContest*)

Definition at line 245 of file client_handler.cpp.

References parentContest.

Referenced by Client_Acceptor::make_svc_handler().

5.2.5.12 int Client_Handler::svc (void) [protected]

If the Client_Acceptor which created us has chosen a thread-per-connection strategy then our open() method will activate us into a dedicate thread. The svc() method will then execute in that thread performing some of the functions we used to leave up to the reactor.

Definition at line 191 of file client_handler.cpp.

References data, connectionMsgBlock::DATASIZE, and process().

5.2.5.13 Thread_Pool * Client_Handler::thread_pool (void) [protected]

Likewise for access to the Thread_Pool that we belong to.

Definition at line 46 of file client_handler.cpp.

References client_acceptor(), and Client_Acceptor::thread_pool().

Referenced by handle_input().

5.2.6 Field Documentation

5.2.6.1 Client_Acceptor* Client_Handler::client_acceptor_ [protected]

Definition at line 172 of file client_handler.h.

Referenced by client_acceptor().

5.2.6.2 ACE_thread_t Client_Handler::creator_ [protected]

For some reason I didn't create accessor/mutator methods for this. So much for consistency....

This variable is used to remember the thread in which we were created: the "creator" thread in other words. handle_input() needs to know if it is operating in the main reactor thread (which is the one that created us) or if it is operating in one of the thread pool threads. More on this when we get to handle_input().

Definition at line 182 of file client_handler.h.

Referenced by handle_input().

5.2.6.3 unsigned char* Client_Handler::data [protected]

Definition at line 119 of file client_handler.h.

Referenced by Client_Handler(), handle_input(), svc(), and ~Client_Handler().

5.2.6.4 ACE_Thread_Mutex Client_Handler::mutex_ [protected]

Definition at line 113 of file client_handler.h.

5.2.6.5 Contest* Client_Handler::parentContest [protected]

Definition at line 116 of file client_handler.h.

Referenced by getContest(), process(), and setContest().

5.2.6.6 int Client_Handler::srv_counter [protected]

Definition at line 122 of file client_handler.h.

Referenced by Client_Handler(), and process().

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

- client_handler.h
- client_handler.cpp

5.3 connectionMsgBlock Class Reference

```
#include <connectionmsgblock.h>
```

5.3.1 Detailed Description

The information that is exchanged between the evalclient and the evalserver programs, is encapsulated in a connectionMsgBlock structure. This structure is of the following form:

- partiticipation code (type long long: 8 bytes)
- timestamp (type time_t: 4 bytes)
- MSISDN (string (not null terminated) : partDetails::MSISDNSIZE, 13 bytes)
- prize id (unsigned char: 1 byte)
- Checksum of the structure (CRC32 : 4 bytes)

Methods are provided to check for the validity of the structure, and the retrieval of the information inside the block.

Definition at line 38 of file connectionmsgblock.h.

Public Types

- enum { DATASIZE = 29, DATA_WO_CRC = 25, CRC_OFFSET = 25 }

  These enums give a more insightful view of the data positions inside the block.

Public Methods

- connectionMsgBlock (long long code, class partDetails &pd)

  This constructor takes the code and the partDetails object, and creates a message block.

- connectionMsgBlock (unsigned char ∗data)

  This constructor takes a buffer and creates a message block WITH the right CRC.

- ∼connectionMsgBlock ()

  Default destructor, just deallocates memory for the buffer data.

- int retrieveCRC ()

  Retrieve the CRC from the buffer.

- bool isValid ()

  Checks for the validity of the message block.

- bool operator== (connectionMsgBlock &cmb)

  Compares two message blocks.

- unsigned char ∗ getData ()

  Returns the raw buffer of the message block.

- pair< long long, partDetails > getParticipant ()

  Returns a pair containing the code (as a long long) and the corresponding partDetails.

- pair< string, partDetails > getParticipantStr ()

  Returns a pair containing the code (as a string) and the corresponding partDetails.

Private Attributes

- CRC_32 crc32

  The object used to calculate the CRC of the message block.

- unsigned char ∗ data

  Pointer to the data holding the message block (size: DATASIZE).

5.3.2 Member Enumeration Documentation

5.3.2.1 anonymous enum

Enumeration values:

DATASIZE

DATA_WO_CRC

CRC_OFFSET

Definition at line 46 of file connectionmsgblock.h.

5.3.3 Constructor & Destructor Documentation

5.3.3.1 connectionMsgBlock::connectionMsgBlock (long long *code*, class partDetails & *pd*)

This constructor constructs a message block, given the components separetely. It takes the code (long long type) and the partDetails object containing the rest of the data (must be created beforehand). It then puts the data into place and calculates the CRC for the block. The result is a valid connectionMsgBlock object.

Definition at line 20 of file connectionmsgblock.cpp.

References crc32, data, DATA_WO_CRC, DATASIZE, CRC_32::getCRC(), and partDetails::MSISDNSIZE.

5.3.3.2 connectionMsgBlock::connectionMsgBlock (unsigned char ∗ *newdata*)

This constructor constructs a message block from the raw data. It takes an existing buffer (without the CRC part) and appends the correct CRC.

Definition at line 53 of file connectionmsgblock.cpp.

References crc32, CRC_OFFSET, data, DATA_WO_CRC, DATASIZE, and CRC_32::getCRC().

5.3.3.3 connectionMsgBlock::∼connectionMsgBlock ()

The destructor has no other purpose than to deallocate the memory used by the buffer.

Definition at line 91 of file connectionmsgblock.cpp.

References data.

5.3.4 Member Function Documentation

5.3.4.1 unsigned char ∗ connectionMsgBlock::getData ()

Returns the buffer holding the data

Definition at line 82 of file connectionmsgblock.cpp.

References data.

Referenced by operator==().

5.3.4.2 pair< long long, partDetails > connectionMsgBlock::getParticipant ()

This method returns a pair object containing the code and respective partDetails object of the current connectionMsgBlock object. The code is of type long long.

Definition at line 129 of file connectionmsgblock.cpp.

References data, partDetails::MSISDNSIZE, partDetails::setGiftId(), partDetails::setMSISDN(), and partDetails::setTimestamp().

Referenced by getParticipantStr().

5.3.4.3 pair< string, partDetails > connectionMsgBlock::getParticipantStr ()

This is similar to getParticipant(). Likewise, it returns a pair object containing the code and respective partDetails object of the current connectionMsgBlock object. This time the code is of type string.

Definition at line 166 of file connectionmsgblock.cpp.

References getParticipant().

Referenced by Processor::process_i().

5.3.4.4 bool connectionMsgBlock::isValid ()

Validity of the object is checked by calculating the CRC of the data in the buffer and comparing it to the CRC _in_ the buffer. Returns true if these are equal.

Definition at line 113 of file connectionmsgblock.cpp.

References crc32, data, DATA_WO_CRC, CRC_32::getCRC(), and retrieveCRC().

Referenced by Client_Handler::process().

5.3.4.5 bool connectionMsgBlock::operator== (connectionMsgBlock & *cmb*)

Compares two connectionMsgBlocks (using the operator==) It compares the buffers AND the crcs of each object. Returns true if the objects are equal.

Definition at line 66 of file connectionmsgblock.cpp.

References data, DATASIZE, getData(), and retrieveCRC().

5.3.4.6 int connectionMsgBlock::retrieveCRC ()

Retrieves the CRC of the current object, as held in the buffer.

Definition at line 100 of file connectionmsgblock.cpp.

References CRC_OFFSET, and data.

Referenced by isValid(), and operator==().

5.3.5 Field Documentation

5.3.5.1 CRC_32 connectionMsgBlock::crc32 `[private]`

Definition at line 40 of file connectionmsgblock.h.

Referenced by connectionMsgBlock(), and isValid().

5.3.5.2 unsigned char∗ connectionMsgBlock::data `[private]`

Definition at line 43 of file connectionmsgblock.h.

Referenced by connectionMsgBlock(), getData(), getParticipant(), isValid(), operator==(), retrieveCRC(), and ∼connectionMsgBlock().

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

- connectionmsgblock.h
- connectionmsgblock.cpp

5.4 Contest Class Reference

`#include <contest.h>`

Public Types

- enum { PORT_NUM = 10101 }

  The default port number on which the evalserver will listen on.

- enum { ID_SOC = 0, ID_EOD = 1, ID_EOW = 2, ID_EOM = 4, ID_EOC = -1 }
- enum { MINUTEPRIZES = 1440, HOURLYPRIZES = 24, DAILYPRIZES = 1, WEEKLYPRIZES = 1, MONTHLYPRIZES = 1 }

Public Methods

- Contest (char ∗hostname)

  The Constructor. it will take as arguments the hostname and the port number to bind to.

- ∼Contest ()

  The destructor. Clean up everything and write the final reports.

- void init ()

  The init() method is responsible for some setup that cannot happen in the constructor.

- void shutdown ()

  Shutdown cleans up the stuff from init().

- void preparetoShutdown ()

  Sets the varible ReadyToShutdown to true.

- virtual int handle_timeout (const ACE_Time_Value &tv, const void ∗arg)

  The method to handle timeouts.

- int handle_signal (int signum, siginfo_t ∗, ucontext_t ∗)

  Catch and handle signals.

- bool initTimers ()

  Set up and schedule all timers, and their respective actions.

- bool initThreads (ACE_INET_Addr ∗iaddr)

  Set up the thread creation engine.

- bool initSignal ()

  Set up the signal handler to catch signals.

- bool setupPrizes ()

  Allocate and initialize the Prizes map.

- bool initDB ()

  Initialize Database Connection.

- bool initLogger ()

  Initialize the logger object.

- bool initProcessor ()

  Initialize the Processor object.

- void start ()

  Basically a wrapper to handle_events().

- Day ∗ getCurrentDay ()

  Return current day.

- Day ∗ getLastDay ()

  Return last day.

- vector< class Day > & getContestDays ()

  Return ContestDays vector.

- bool giftIsGiven (TimePeriod tp, u_int gid)

  Return true if gift gid is given in period tp.

- void logMsg (string msg)

  Wrapper around logger object's logMsg.

Data Fields

- TimeStamp SOC

  Start of Contest Timestamp.

- vector< TimeStamp > EOD

End-of-day Timer. There will be apprx. 90 of them (Oct 1, 2002 - Dec 31, 2002).

- vector< TimeStamp > EOW

    End-of-week Timer. Apprx 12 of them.

- vector< TimeStamp > EOM

    End-of-month Timer.

- TimeStamp EOC

    End-of-contest Timer. Basically it will shutdown the program.

- u_int current_day

    Index variables to keep count of where we are.

- u_int current_week

    Index variables to keep count of where we are.

- u_int current_month

    Index variables to keep count of where we are.

- SQLTable< string, partDetails > Participants

    The Participants map; basically it will hold a log of every participation.

- SQLTable< string, partDetails > Winners

    Winners map is for search purposes only,.

- SQLTable< string, Count > Subscribers

    Subscribers is used to hold the unique MSISDNS and a counter for each.

- SQLTable< string, giftDetails > Prizes

    A map of the gifts of the contest.

- vector< string > PrizeNames

    This one holds the names of the prizes.

- SQLTable< string, Day > Counters

    We also keep an SQL TABLE called COUNTERS for faster statistics.

- Logger * logger

    Logger object.

- Processor * processor

    Processor object.

- SQLiteConnection * dbconnection

    Database Connection Object.

Private Attributes

- ACE_Thread_Mutex mutex_

  A basic Thread_Mutex class that will ensure safe access to data.

- ACE_Reactor reactor

  Reactor.

- Client_Acceptor ∗ peer_acceptor

  The thread pool acceptor.

- ACE_INET_Addr ∗ iaddr

  The internet address to bind to.

- vector< class Day > ContestDays

  Day objects vector.

- bool ReadyToShutdown

  The program will operate while this is false.

- pid_t basethreadpid

  We need to know the parent thread.

- bool SetupPrizes

  If we are restarting (from a crash?) there is no need to setup the prizes again.

5.4.1 Member Enumeration Documentation

5.4.1.1 anonymous enum

Enumeration values:

PORT_NUM

Definition at line 83 of file contest.h.

5.4.1.2 anonymous enum

handle_timeout() receives one of ID_{SOC,EOC,EOD,EOW,EOM} to signify the end of the respective timer. All these are set as arguments in schedule_timer() in initTimers().

Enumeration values:

ID_SOC

ID_EOD

ID_EOW

ID_EOM

ID_EOC

Definition at line 88 of file contest.h.

5.4.1.3 anonymous enum

The quantity of the prizes in their respective periods. The minute/hour are rather specified as daily prizes with the quantity set accordingly.

Enumeration values:

MINUTEPRIZES

HOURLYPRIZES

DAILYPRIZES

WEEKLYPRIZES

MONTHLYPRIZES

Definition at line 94 of file contest.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Contest::Contest (char $*$ *hostname*)

The constructor. It sets up the internet address to listen to, Initializes the random number generator with a seed, and sets up the timers of the Contest. The rest of the initialization is done in init().

Definition at line 18 of file contest.cpp.

References basethreadpid, iaddr, initLogger(), initProcessor(), initTimers(), logMsg(), PORT_NUM, ReadyToShutdown, and SetupPrizes.

5.4.2.2 Contest::~Contest ()

The destructor. Just a dummy, all the important stuff is done in shutdown()

Definition at line 46 of file contest.cpp.

5.4.3 Member Function Documentation

5.4.3.1 vector< class Day > & Contest::getContestDays ()

Definition at line 565 of file contest.cpp.

References ContestDays.

Referenced by Day::calculateEstNoMsgs().

5.4.3.2 Day $*$ Contest::getCurrentDay ()

Returns a pointer to the current day of the contest

Definition at line 544 of file contest.cpp.

References ContestDays, and current_day.

Referenced by Processor::process_i().

5.4.3.3 Day $*$ Contest::getLastDay ()

Returns a pointer to the last day of the contest or NULL if the contest has just started (so that it has no last day)

Definition at line 556 of file contest.cpp.

References ContestDays, and current_day.

5.4.3.4 bool Contest::giftIsGiven (TimePeriod *tp*, u_int *gid*)

Given a TimePeriod object and a gift ID, it returns true if the gift is given in that period, false otherwise

Definition at line 573 of file contest.cpp.

References TimePeriod::getBeginTS(), logMsg(), Prizes, and SQLTable< string, giftDetails >::selectObjects().

Referenced by Day::executeDraw().

5.4.3.5 int Contest::handle_signal (int *signum*, siginfo_t *, ucontext_t *)

This one is responsible for handling the signal INT, TERM and QUIT. The only reason we want these caught and handled is to know when the software is terminated on a user's command. The action is logged in evalserver.log. The only thing we do is set ReadyToShutdown to true.

Definition at line 237 of file contest.cpp.

References basethreadpid, logMsg(), and preparetoShutdown().

5.4.3.6 int Contest::handle_timeout (const ACE_Time_Value & *tv*, const void * *argument*) `[virtual]`

handle_timeout() is called every time a timer is triggered. Because timers are given an argument, which in this case is an ID number regarding the kind of the timer (Start of Contest, End of Contest, End of Day, End of Week, End of Month), handle_timeout can prepare the necessary structures to create the next Day object, or start/end the contest. Because this works in a multithreaded environment, we have included everything in an ACE_Guard scope. At the beginning of the timeout handling all transactions are disabled and are re-enabled at the end.

Definition at line 148 of file contest.cpp.

References SQLiteConnection::changeLogFilename(), ContestDays, current_day, current_month, current_week, dbconnection, SQLiteConnection::disableTransactions(), SQLiteConnection::enableTransactions(), EOD, EOM, EOW, TimePeriod::getEndTS(), ID_EOC, ID_EOD, ID_EOM, ID_EOW, ID_SOC, init(), logMsg(), mutex_, TimeStamp::nextDay(), preparetoShutdown(), SOC, and SQLiteConnection::transactionsEnabled().

5.4.3.7 void Contest::init ()

Contest initialization method. Some of the setup cannot happen in the constructor, or actually should not happen. The constructor initializes the object and the necessary parts of the Contest object, but does *not* actually start the Contest. This happens when the SOC (Start Of Contest) timer is triggered. Then the sockets are created, the thread engine is initialized, the database connection is setup, and the prize pool is created. Also, the counters current_day, current_week, current_month are set to zero.

Definition at line 60 of file contest.cpp.

References current_day, current_month, current_week, iaddr, initDB(), initSignal(), initThreads(), logMsg(), setupPrizes(), and SetupPrizes.

Referenced by handle_timeout().

5.4.3.8 bool Contest::initDB ()

Here we open the database connection and create the tables if we have to. When we create a table, we use two vector<string> objects. fieldnames and fieldtypes for the names and types of the fields of the SQL table respectively. Then we initialize the SQLTable using the appropriate templates and we also pass the logger object as an argument, since we want to have one global logger.

Definition at line 631 of file contest.cpp.

References Counters, dbconnection, DBNAME, SQLiteConnection::enableTransactions(), SQLiteConnection::existsTable(), SQLTable< string, giftDetails >::isReady(), SQLTable< string, Day >::isReady(), SQLTable< string, partDetails >::isReady(), SQLiteConnection::isReady(), logger, Participants, Prizes, SetupPrizes, and Winners.

Referenced by init().

5.4.3.9 bool Contest::initLogger (void)

initLogger creates and opens the logger object. The logger object is a thread that runs asynchronously to the other threads and does all the logging (that is all I/O that is relevant to logging).

Definition at line 603 of file contest.cpp.

References LOGFILENAME, logger, Logger::open(), and OUTPUT_BOTH.

Referenced by Contest().

5.4.3.10 bool Contest::initProcessor (void)

initProcessor creates and opens the Processor object. This is the object that the processing, that is the draw is done.

Definition at line 615 of file contest.cpp.

References Processor::open(), and processor.

Referenced by Contest().

5.4.3.11 bool Contest::initSignal ()

Set up the signal handler to catch signals. The program should catch SIGINT (Ctrl-C), SIGQUIT (Ctrl-\) and if possible SIGKILL and log the action before quiting.

Definition at line 405 of file contest.cpp.

Referenced by init().

5.4.3.12 bool Contest::initThreads (ACE_INET_Addr ∗ *iaddr*)

Creates the acceptor object which uses the Strategy feature of ACE to create a thread per socket connection and runs svc() in that thread to process the connection.

Definition at line 387 of file contest.cpp.

References iaddr, Client_Acceptor::open(), and peer_acceptor.

Referenced by init().

5.4.3.13 bool Contest::initTimers ()

This method initializes the EOD, EOW, EOM, SOC, EOC TimeStamp objects (or vectors of them) and registers these with the reactor instance. We also have to make sure that no timers overlap. That is, if a day timer overlaps with a week or month timer, we exclude the day timer from the registered timers in ACE, but NOT from the EOD vector.

Definition at line 275 of file contest.cpp.

References EOC, EOD, EOM, EOW, TimeStamp::getTimeStamp(), ID_EOC, ID_EOD, ID_EOM, ID_EOW, ID_SOC, logMsg(), TimeStamp::nextWeek(), TimeStamp::setTimeStamp(), SOC, start(), and TimeStamp::toString().

Referenced by Contest().

5.4.3.14 void Contest::logMsg (string *msg*)

Instead of using directly the logmsg method from the logger object, we wrap it with another one, so that if it (the logger) is not available (at the start or the end of execution) we will still have a logging mechanism available via cout.

Definition at line 736 of file contest.cpp.

References logger, and Logger::logMsg().

Referenced by Contest(), giftIsGiven(), handle_signal(), handle_timeout(), init(), initTimers(), setupPrizes(), and shutdown().

5.4.3.15 void Contest::preparetoShutdown ()

Just sets the ReadyToShutdown flag to true.

Definition at line 90 of file contest.cpp.

References ReadyToShutdown.

Referenced by handle_signal(), and handle_timeout().

5.4.3.16 bool Contest::setupPrizes (void)

Set up the prizes for the contest, give them identifiers and register them in the Prizes table (SQL) of the database.

Definition at line 425 of file contest.cpp.

References DAILYPRIZES, EOC, EOD, EOM, EOW, giftDetails::ID_DAILYPRIZES, giftDetails::ID_MINUTEPRIZES, giftDetails::ID_MONTHLYPRIZES, giftDetails::ID_WEEKLYPRIZES, SQLTable< string, giftDetails >::insertObject(), giftDetails::insertString(), logMsg(), MINUTEPRIZES, MONTHLYPRIZES, PrizeNames, Prizes, SQLTable< string, giftDetails >::sumColumn(), and WEEKLYPRIZES.

Referenced by init().

5.4.3.17 void Contest::shutdown ()

The deallocation of resources happens here. It first shuts down the database connection, cancels all the (remaining) timers and deactivates the reactor.

Definition at line 98 of file contest.cpp.

References dbconnection, EOC, TimeStamp::getTimeStamp(), iaddr, logMsg(), and SOC.

Referenced by start().

5.4.3.18 void Contest::start ()

Basically calls the handle_events() of the reactor which is responsible for handling all the events (socket connection, timer triggering)

Definition at line 373 of file contest.cpp.

References ReadyToShutdown, and shutdown().

Referenced by initTimers(), and main().

5.4.4 Field Documentation

5.4.4.1 pid_t Contest::basethreadpid [private]

Definition at line 76 of file contest.h.

Referenced by Contest(), and handle_signal().

5.4.4.2 vector<class Day> Contest::ContestDays [private]

Definition at line 70 of file contest.h.

Referenced by getContestDays(), getCurrentDay(), getLastDay(), and handle_timeout().

5.4.4.3 SQLTable<string, Day> Contest::Counters

Definition at line 142 of file contest.h.

Referenced by Day::Day(), initDB(), and Processor::process_i().

5.4.4.4 u_int Contest::current_day

Definition at line 109 of file contest.h.

Referenced by getCurrentDay(), getLastDay(), handle_timeout(), and init().

5.4.4.5 u_int Contest::current_month

Definition at line 109 of file contest.h.

Referenced by Day::assignPrizes(), Day::executeDraw(), handle_timeout(), and init().

5.4.4.6 u_int Contest::current_week

Definition at line 109 of file contest.h.

Referenced by Day::assignPrizes(), Day::executeDraw(), handle_timeout(), and init().

5.4.4.7 SQLiteConnection* Contest::dbconnection

Definition at line 196 of file contest.h.

Referenced by handle_timeout(), initDB(), Processor::process_i(), and shutdown().

5.4.4.8 TimeStamp Contest::EOC

Definition at line 106 of file contest.h.

Referenced by initTimers(), setupPrizes(), and shutdown().

5.4.4.9 vector<TimeStamp> Contest::EOD

Definition at line 100 of file contest.h.

Referenced by handle_timeout(), initTimers(), and setupPrizes().

5.4.4.10 vector<TimeStamp> Contest::EOM

Definition at line 104 of file contest.h.

Referenced by Day::assignPrizes(), Day::executeDraw(), handle_timeout(), initTimers(), and setupPrizes().

5.4.4.11 vector<TimeStamp> Contest::EOW

Definition at line 102 of file contest.h.

Referenced by Day::assignPrizes(), Day::executeDraw(), handle_timeout(), initTimers(), and setupPrizes().

5.4.4.12 ACE_INET_Addr* Contest::iaddr `[private]`

Definition at line 67 of file contest.h.

Referenced by Contest(), init(), initThreads(), and shutdown().

5.4.4.13 Logger* Contest::logger

Definition at line 190 of file contest.h.

Referenced by Day::assignPrizes(), Day::calculateEstNoMsgs(), Day::Day(), initDB(), initLogger(), logMsg(), Client_Handler::process(), and Processor::process_i().

5.4.4.14 ACE_Thread_Mutex Contest::mutex_ `[private]`

Definition at line 58 of file contest.h.

Referenced by handle_timeout().

5.4.4.15 SQLTable<string, partDetails> Contest::Participants

Definition at line 127 of file contest.h.

Referenced by initDB(), and Processor::process_i().

5.4.4.16 Client_Acceptor* Contest::peer_acceptor `[private]`

Definition at line 64 of file contest.h.

Referenced by initThreads().

5.4.4.17 vector<string> Contest::PrizeNames

Definition at line 139 of file contest.h.

Referenced by setupPrizes().

5.4.4.18 SQLTable<string, giftDetails> Contest::Prizes

Definition at line 136 of file contest.h.

Referenced by Day::assignPrizes(), giftIsGiven(), initDB(), Processor::process_i(), and setupPrizes().

5.4.4.19 Processor* Contest::processor

Definition at line 193 of file contest.h.

Referenced by initProcessor(), and Client_Handler::process().

5.4.4.20 ACE_Reactor Contest::reactor `[private]`

Definition at line 61 of file contest.h.

5.4.4.21 bool Contest::ReadyToShutdown `[private]`

Definition at line 73 of file contest.h.

Referenced by Contest(), preparetoShutdown(), and start().

5.4.4.22 bool Contest::SetupPrizes `[private]`

Definition at line 79 of file contest.h.

Referenced by Contest(), init(), and initDB().

5.4.4.23 TimeStamp Contest::SOC

Definition at line 98 of file contest.h.

Referenced by handle_timeout(), initTimers(), and shutdown().

5.4.4.24 SQLTable<string, Count> Contest::Subscribers

Definition at line 133 of file contest.h.

5.4.4.25 SQLTable<string, partDetails> Contest::Winners

Definition at line 130 of file contest.h.

Referenced by initDB(), and Processor::process_i().

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

- contest.h
- contest.cpp

5.5 Count Class Reference

```
#include <Count.h>
```

Public Methods

- Count ()
- Count (counter_t &counter)
- Count (string str)
- void operator++ (int)
- void operator– (int)
- unsigned int val ()
- void set (unsigned int val)
- string insertString ()
- string updateString ()

Protected Attributes

- counter_t counter_

5.5.1 Constructor & Destructor Documentation

5.5.1.1 Count::Count () `[inline]`

A simple initializing constructor, just sets counter_ to 0.

Definition at line 47 of file Count.h.

References counter_.

5.5.1.2 Count::Count (counter_t & *counter*) `[inline]`

The copy constructor.

Definition at line 51 of file Count.h.

References counter_, and counter_t.

5.5.1.3 Count::Count (string *str*) `[inline]`

A constructor that takes a number in a string and converts to an integer.

Definition at line 56 of file Count.h.

References counter_.

5.5.2 Member Function Documentation

5.5.2.1 string Count::insertString () `[inline]`

This method exists so that Count object can be used as a templated class for the insert∗ methods in SQLTable. What it basically does is return a string representation of its contents.

Definition at line 81 of file Count.h.

References counter_.

5.5.2.2 void Count::operator++ (int) `[inline]`

Overloaded ++ operator, increases counter_ by 1.

Definition at line 62 of file Count.h.

References counter_.

5.5.2.3 void Count::operator– (int) `[inline]`

Overloaded – operator, decreases counter_ by 1.

Definition at line 66 of file Count.h.

References counter_.

5.5.2.4 void Count::set (unsigned int *val*) `[inline]`

Sets value of counter_ to given val.

Definition at line 74 of file Count.h.

References counter_, and val().

Referenced by Day::Day(), and Day::setCounter().

5.5.2.5 string Count::updateString () `[inline]`

This method exists so that Count object can be used as a templated class for the update∗ methods in SQLTable. Like the insertString() method it returns a string that should contain the part of the command to update the counter value in the SQL table.

Definition at line 94 of file Count.h.

References counter_.

5.5.2.6 unsigned int Count::val () `[inline]`

Retrieves current value of counter_.

Definition at line 70 of file Count.h.

References counter_.

Referenced by Day::executeDraw(), Day::getCounter(), Day::getUniqueCounter(), Day::insertString(), set(), Day::setCounter(), and Day::updateString().

5.5.3 Field Documentation

5.5.3.1 counter_t Count::counter_ `[protected]`

The actual counter object. It is of type ACE_Atomic_Op using templates ACE_Mutex and unsigned int

Definition at line 43 of file Count.h.

Referenced by Count(), insertString(), operator++(), operator–(), set(), updateString(), and val().

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

- Count.h

5.6 Counter_Guard Class Reference

Public Methods

- Counter_Guard (Thread_Pool::counter_t &counter)
- ~Counter_Guard (void)

Protected Attributes

- Thread_Pool::counter_t & counter_

5.6.1 Constructor & Destructor Documentation

5.6.1.1 Counter_Guard::Counter_Guard (Thread_Pool::counter_t & *counter*) `[inline]`

Definition at line 122 of file thread_pool.cpp.

References counter_, and Thread_Pool::counter_t.

5.6.1.2 Counter_Guard::~Counter_Guard (void) `[inline]`

Definition at line 128 of file thread_pool.cpp.

References counter_.

5.6.2 Field Documentation

5.6.2.1 Thread_Pool::counter_t& Counter_Guard::counter_ `[protected]`

Definition at line 134 of file thread_pool.cpp.

Referenced by Counter_Guard(), and ~Counter_Guard().

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

- thread_pool.cpp

5.7 CRC_32 Class Reference

```
#include <crc_32.h>
```

5.7.1 Detailed Description

It is used by connectionMsgBlock class to ensure that data is correctly transferred. Although the chances of data sent incorrectly are quite minimal, especially in this particular case, where the server runs on the same machine as the client, it still is good practice. The usage is simple. After initialization of an object, one can use it to retrieve the CRC of a given buffer with getCRC().

Definition at line 42 of file crc_32.h.

Public Methods

- CRC_32 ()

Basic Constructor, it initializes the crc32_table.

- int getCRC (unsigned char ∗data, size_t size)

  Creates a CRC from a data buffer.

Protected Methods

- u_long reflect (u_long ref, unsigned char ch)

  Reflects CRC bits in the lookup table.

Protected Attributes

- u_long crc32_table [256]

  Lookup table array.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 CRC_32::CRC_32 ()

The constructor. It basically creates the crc32 table used for calculating the checksums.

Definition at line 17 of file crc_32.cpp.

References crc32_table, and reflect().

5.7.3 Member Function Documentation

5.7.3.1 int CRC_32::getCRC (unsigned char ∗ *data*, size_t *size*)

The only useful method for the user. getCRC() is given a buffer called data for which it calculates and returns the CRC. The buffer MUST be of type unsigned char ∗ (casting is ok).

Definition at line 56 of file crc_32.cpp.

References crc32_table.

Referenced by connectionMsgBlock::connectionMsgBlock(), and connectionMsgBlock::isValid().

5.7.3.2 u_long CRC_32::reflect (u_long *ref*, unsigned char *ch*) `[protected]`

Helper method used by the constructor. It swaps the bit orientation in ch, based on ref.

Definition at line 36 of file crc_32.cpp.

Referenced by CRC_32().

5.7.4 Field Documentation

5.7.4.1 u_long CRC_32::crc32_table[256] `[protected]`

Definition at line 45 of file crc_32.h.

Referenced by CRC_32(), and getCRC().

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

- crc_32.h
- crc_32.cpp

5.8 Day Class Reference

```
#include <day.h>
```

5.8.1 Detailed Description

The day class is used for the following reasons:

- Hold statistical information for the participations daily.
- Hold the array of prizes to be given daily.
- execute the actual draw of the competition, that is decide whether this participation wins or not, and what.

  There are three possible ways to create a Day object, using three possible constructors. The first Day() is just a dummy constructor that just creates an empty shell to be populated later. It is not usable. The second takes a string representation of a Day's details, actually the result from an SQL SELECT command in the COUNTERS table, so that it can construct the Day again. It's only used in statistics. The third constructor, takes the time period, a flag to signify whether we are at the start of a new week or month, and performs the following actions:
- Estimate the number of the participations for the particular day.
- Assign the prizes in fixed positions in an array that holds the messages.
- provide a method to be called from the Processor object, to execute the actual draw of the competition.

  The class Day is also used as a template class for SQLTable<>. For this reason, we have created two methods insertString() and updateString(), that return the daily counters as strings representations, to be used in the insert∗ and update∗ SQLTable methods respectively. About the DayMessages vector: This one has a ”current index” variable, a cursor, that moves forward one position for each participation. At the start of the day, this vector is cleared and resized to hold the estimated messages for this day. For each participation, it picks the value at the current position and returns it to the participant. The value is either 0 (for no prize), or the id of the prize itself.

Definition at line 63 of file day.h.

Public Methods

- Day (class Contest ∗myContest, TimeStamp &ts)

  The class constructor.

- Day (string daystr)

  Class constructor for SQLTable<> support.

- Day ()

  dummy constructor that just creates an empty structure

- TimePeriod getTimePeriod ()

return the current day TimePeriod object

- void setTimePeriod (TimePeriod &tp)

  Sets the time period.

- void assignPrizes ()

  Assign the prizes to the DayMessages.

- u_int calculateEstNoMsgs ()

  Calculate estimated Number of Messages for this day.

- u_int getEstNoMsgs ()

  Return estimated Number of Messages for this day.

- bool executeDraw (partDetails &pd)

  Execute draw for the given partDetails object.

- void setCounter (u_int &val)

  Set counter.

- u_int getCounter ()

  return counter

- u_int getUniqueCounter ()

  return unique counter

- u_int getNoPrizes ()

  return no of prizes

- string insertString ()

  This is for the SQLTable<> template class.

- string updateString ()

  This is for the SQLTable<> template class.

- bool isEmpty ()

  boolean method to be used in SQLTable<> methods

Private Types

- enum { default_avg_threshold = 5 }

  An enumerator object for the calculation of the estimated number of messages.

- enum { default_estNoMsgs = 10000, min_estNoMsgs = 1600 }

  Enumeration objects for the default and minimum number of est. number of messages.

Private Attributes

- Contest ∗ parentContest

  A pointer to the Contest part of which is this.

- vector< u_int > DayMessages

  The vector that holds the Prizes that are given to the participants.

- Count counter

  Counter of the current number of participants.

- Count unique_counter

  The unique players for today.

- u_int estNoMsgs

  The estimated number of messages for this day.

- u_int prizes

  The number of prizes won so far in the current day.

- TimePeriod day_period

  The current day period.

- bool empty

  boolean for SQLTable<> template class

Friends

- class Processor

  The Processor is a friend class and can access the private members of Day.

5.8.2 Member Enumeration Documentation

5.8.2.1 anonymous enum `[private]`

Enumeration values:

default_avg_threshold

Definition at line 66 of file day.h.

5.8.2.2 anonymous enum `[private]`

Enumeration values:

default_estNoMsgs

min_estNoMsgs

Definition at line 69 of file day.h.

5.8.3 Constructor & Destructor Documentation

5.8.3.1 Day::Day (class Contest ∗ *myContest*, TimeStamp & *start_ts*)

The Day constructor. It takes the Contest object pointer, the starting TimeStamp. It estimates the number of messages for the specific time period and resizes the DayMessages vector. Firstly, it checks the COUNTERS table to see if a record for the particular day already exists, and if so, retrieves the counters from the table. Then the system will continue from the previous state. Otherwise it will try to estimate the number of messages for this day using calculateEstNoMsgs(). Whatever the case it will have a valid value for estNoMsgs and will resize DayMessages vector with that size. Then it will assign the available prizes (taking into account the COUNTERS variable prizes) into random positions in DayMessages.

Definition at line 29 of file day.cpp.

References assignPrizes(), calculateEstNoMsgs(), counter, Contest::Counters, day_period, DayMessages, estNoMsgs, TimePeriod::getBeginTS(), getCounter(), getEstNoMsgs(), getNoPrizes(), TimeStamp::getTimeStamp(), getUniqueCounter(), SQLTable< string, Day >::insertObject(), isEmpty(), Contest::logger, Logger::logMsg(), TimeStamp::nextDay(), parentContest, prizes, Count::set(), TimeStamp::toString(), TimePeriod::toString(), and unique_counter.

5.8.3.2 Day::Day (string *daystr*)

Take a string representation of the variables in the COUNTERS table, counter, unique_counter, estNoMsgs and prizes, and resets the day, setting the empty boolean to true as well.

Definition at line 75 of file day.cpp.

References counter, empty, estNoMsgs, prizes, Count::set(), strTokenizer(), and unique_counter.

5.8.3.3 Day::Day () `[inline]`

Definition at line 105 of file day.h.

References empty.

5.8.4 Member Function Documentation

5.8.4.1 void Day::assignPrizes ()

assignPrizes() is responsible for attaching a gift to a position in the DayMessages vector. Basically, DayMessages is a vector whose current position will give the next given prize. Since probabilities are floating point numbers and we have to correspond these to gift ids, we have to iterate over all available presents for this day and assign them to the DayMessages vector. Espessially for the weekly and monthly gifts, we don't force them to be assigned in the first n-1 days of their respective periods but we do force them to be assigned in the last day.

Definition at line 103 of file day.cpp.

References Contest::current_month, Contest::current_week, TimePeriod::dateInPeriod(), day_period, DayMessages, Contest::EOM, Contest::EOW, TimePeriod::getBeginTS(), giftDetails::ID_MONTHLYPRIZES, giftDetails::ID_NOGIFT, giftDetails::ID_WEEKLYPRIZES, Contest::logger, Logger::logMsg(), parentContest, TimeStamp::previousDay(), Contest::Prizes, and SQLTable< string, giftDetails >::selectObjects().

Referenced by Day().

5.8.4.2 u_int Day::calculateEstNoMsgs ()

Calculates an average of the past {default_avg_threshold} days' counters, and returns the result. If the Contest has just started then it returns {default_estNoMsgs}, and if the result of 0.8 ∗ average is less than {min_estNoMsgs} it returns {min_estNoMsgs} instead.

Definition at line 194 of file day.cpp.

References default_avg_threshold, default_estNoMsgs, Contest::getContestDays(), Contest::logger, Logger::logMsg(), min_estNoMsgs, and parentContest.

Referenced by Day().

5.8.4.3 bool Day::executeDraw (partDetails & *pd*)

The actual draw engine is here. Given a partDetails object which holds information on the code, the MSISDN, and the timestamp, fill the appropriate gift id in this object. We take the gift id from the current value in DayMessages vector in the position pointed to by counter, if of course, counter is less than the size of DayMessages. If it is bigger, then the prize won is nothing. At the end a check is made to confirm that we don't give extra prizes.

Definition at line 243 of file day.cpp.

References counter, Contest::current_month, Contest::current_week, day_period, DayMessages, Contest::EOM, Contest::EOW, partDetails::getGiftId(), Contest::giftIsGiven(), giftDetails::ID_DAILYPRIZES, giftDetails::ID_HOURLYPRIZES, giftDetails::ID_MONTHLYPRIZES, giftDetails::ID_NOGIFT, giftDetails::ID_WEEKLYPRIZES, parentContest, partDetails::setGiftId(), and Count::val().

Referenced by Processor::process_i().

5.8.4.4 u_int Day::getCounter ()

Returns the current value of the counter

Definition at line 296 of file day.cpp.

References counter, and Count::val().

Referenced by Day().

5.8.4.5 u_int Day::getEstNoMsgs ()

Definition at line 183 of file day.cpp.

References estNoMsgs.

Referenced by Day().

5.8.4.6 u_int Day::getNoPrizes ()

Returns the current value

Definition at line 319 of file day.cpp.

References prizes.

Referenced by Day().

5.8.4.7 TimePeriod Day::getTimePeriod ()

Definition at line 173 of file day.cpp.

References day_period.

Referenced by Processor::process_i().

5.8.4.8 u_int Day::getUniqueCounter ()

Returns the current value of the counter

Definition at line 312 of file day.cpp.

References unique_counter, and Count::val().

Referenced by Day().

5.8.4.9 string Day::insertString ()

returns a string suitable to be used in insert∗ methods of SQLTable.

Definition at line 328 of file day.cpp.

References counter, estNoMsgs, prizes, unique_counter, and Count::val().

5.8.4.10 bool Day::isEmpty ()

This is to check if a Day object has been created with the dummy constructor Day().

Definition at line 356 of file day.cpp.

References empty.

Referenced by Day().

5.8.4.11 void Day::setCounter (u_int & *val*)

Sets the value of the counter. Used for recovery purposes.

Definition at line 303 of file day.cpp.

References counter, Count::set(), and Count::val().

5.8.4.12 void Day::setTimePeriod (TimePeriod & *tp*)

Definition at line 178 of file day.cpp.

References day_period.

5.8.4.13 string Day::updateString ()

returns a string suitable to be used in update∗ methods of SQLTable.

Definition at line 342 of file day.cpp.

References counter, estNoMsgs, prizes, unique_counter, and Count::val().

Referenced by Processor::process_i().

5.8.5 Friends And Related Function Documentation

5.8.5.1 friend class Processor `[friend]`

Definition at line 78 of file day.h.

5.8.6 Field Documentation

5.8.6.1 Count Day::counter `[private]`

Definition at line 81 of file day.h.

Referenced by Day(), executeDraw(), getCounter(), insertString(), Processor::process_i(), setCounter(), and updateString().

5.8.6.2 TimePeriod Day::day_period `[private]`

Definition at line 93 of file day.h.

Referenced by assignPrizes(), Day(), executeDraw(), getTimePeriod(), and setTimePeriod().

5.8.6.3 vector<u_int> Day::DayMessages `[private]`

Definition at line 75 of file day.h.

Referenced by assignPrizes(), Day(), and executeDraw().

5.8.6.4 bool Day::empty `[private]`

Definition at line 96 of file day.h.

Referenced by Day(), and isEmpty().

5.8.6.5 u_int Day::estNoMsgs `[private]`

Definition at line 87 of file day.h.

Referenced by Day(), getEstNoMsgs(), insertString(), and updateString().

5.8.6.6 class Contest* Day::parentContest `[private]`

Definition at line 72 of file day.h.

Referenced by assignPrizes(), calculateEstNoMsgs(), Day(), and executeDraw().

5.8.6.7 u_int Day::prizes `[private]`

Definition at line 90 of file day.h.

Referenced by Day(), getNoPrizes(), insertString(), Processor::process_i(), and updateString().

5.8.6.8 Count Day::unique_counter `[private]`

Definition at line 84 of file day.h.

Referenced by Day(), getUniqueCounter(), insertString(), Processor::process_i(), and updateString().

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

- day.h
- day.cpp

5.9 giftDetails Class Reference

`#include <giftdetails.h>`

5.9.1 Detailed Description

This class provides an easy way to hold prize information, and is been used for the value part of the SQL table PRIZES that is been used, and provides access to info, such as the prize id, its name, the initial quantity, remaining and the actual period for which it is available.

Definition at line 33 of file giftdetails.h.

Public Types

- enum { ID_NOGIFT = 0, ID_MINUTEPRIZES = 1, ID_HOURLYPRIZES = 2, ID_DAILYPRIZES = 3, ID_WEEKLYPRIZES = 4, ID_MONTHLYPRIZES = 5, ID_USEDCODE = 10 }

  These enums are used to distinguish the types of the prizes.

Public Methods

- giftDetails ()

  dummy constructor that just creates an empty structure.

- giftDetails (int gid, string newgiftName, TimePeriod tp, int initQ, int rem)

  Default constructor.

- giftDetails (string giftstr)

  Constructor that uses a string representation of the parameters to create the object.

- ~giftDetails ()

  Dummy destructor.

- u_int getGiftId ()

  Returns the id of the prize, usually one of the enums ID_∗.

- string getName ()

  Returns the name of prize as a string.

- size_t getInitialQuantity ()

  Returns the initial quantity of the current prize.

- size_t getCurrentQuantity ()

  Returns the current quantity (available prizes).

- void setCurrentQuantity (int curQ)

  Sets current quantity to the given number.

- void operator– (int)

  The operator– decreases the current quantity of the prize by 1.

- void setName (string name)

  Sets the name to the given string.

- void setInitialQuantity (int initQ)

  Sets the initial quantity to the given number.

- string insertString ()

  This is for the SQLTable<> template class.

- string updateString ()

  This is for the SQLTable<> template class.

- bool isEmpty ()

  boolean method to be used in SQLTable<> methods

- u_int getGiftDuration ()

  Return duration of specific gift.

Private Attributes

- u_int giftid

  The prize id.

- string giftName

  The prize name.

- TimePeriod period

  A TimePeriod during which the prizes will be available.

- int initial

  Initial quantity of the prizes.

- int remaining

  Current quantity of the prizes.

- bool empty

  boolean for SQLTable<> template class

5.9.2 Member Enumeration Documentation

5.9.2.1 anonymous enum

Enumeration values:

ID_NOGIFT

ID_MINUTEPRIZES

ID_HOURLYPRIZES

ID_DAILYPRIZES

ID_WEEKLYPRIZES

ID_MONTHLYPRIZES

ID_USEDCODE

Definition at line 54 of file giftdetails.h.

5.9.3 Constructor & Destructor Documentation

5.9.3.1 giftDetails::giftDetails ()

Dummy constructor. Just creates an empty object.

Definition at line 17 of file giftdetails.cpp.

References empty.

5.9.3.2 giftDetails::giftDetails (int *gid*, string *newgiftName*, TimePeriod *tp*, int *initQ*, int *rem*)

The default constructor creates a giftDetails object given the parameters. Apart from initializing the variables, it also sets empty to false.

Definition at line 28 of file giftdetails.cpp.

References empty.

5.9.3.3 giftDetails::giftDetails (string *gstr*)

This constructor creates a giftDetails object, given a string representation of the parameters. These must comma-separated and with the following order:

- giftid
- name
- start of period
- end of period
- initial quantity
- current quantity

Definition at line 46 of file giftdetails.cpp.

References empty, giftid, period, setCurrentQuantity(), setInitialQuantity(), setName(), and strTokenizer().

5.9.3.4 giftDetails::∼giftDetails ()

Dummy destructor

Definition at line 65 of file giftdetails.cpp.

5.9.4 Member Function Documentation

5.9.4.1 size_t giftDetails::getCurrentQuantity ()

Return the current quantity of the current prize.

Definition at line 96 of file giftdetails.cpp.

References remaining.

5.9.4.2 u_int giftDetails::getGiftDuration ()

Returns the duration that a specific gift is available in days.

Definition at line 138 of file giftdetails.cpp.

References ID_DAILYPRIZES, ID_HOURLYPRIZES, ID_MINUTEPRIZES, ID_MONTHLYPRIZES, and ID_WEEKLYPRIZES.

5.9.4.3 u_int giftDetails::getGiftId ()

Return the prize id

Definition at line 72 of file giftdetails.cpp.

References giftid.

5.9.4.4 size_t giftDetails::getInitialQuantity ()

Return the initial quantity of the current prize.

Definition at line 88 of file giftdetails.cpp.

References initial.

5.9.4.5 string giftDetails::getName ()

Return the name of the prize as a string object

Definition at line 80 of file giftdetails.cpp.

References giftName.

5.9.4.6 string giftDetails::insertString ()

returns a string suitable to be used in insert∗ methods of SQLTable.

Definition at line 162 of file giftdetails.cpp.

References TimePeriod::getBeginTS(), TimePeriod::getEndTS(), TimeStamp::getTimeStamp(), giftName, initial, period, and remaining.

Referenced by Contest::setupPrizes().

5.9.4.7 bool giftDetails::isEmpty ()

This is to check if a giftDetails object has been created with the dummy constructor giftDetails().

Definition at line 194 of file giftdetails.cpp.

References empty.

5.9.4.8 void giftDetails::operator– (int)

Overloads the – operator for this class. It decreases the current quantity of this prize by 1.

Definition at line 113 of file giftdetails.cpp.

References remaining.

5.9.4.9 void giftDetails::setCurrentQuantity (int *curQ*)

Sets the current quantity of the current prize to the given number curQ

Definition at line 104 of file giftdetails.cpp.

References remaining.

Referenced by giftDetails().

5.9.4.10 void giftDetails::setInitialQuantity (int *initQ*)

Sets the initial quantity of the current prize to the given number initQ

Definition at line 122 of file giftdetails.cpp.

References initial.

Referenced by giftDetails().

5.9.4.11 void giftDetails::setName (string *name*)

Sets the name of the prize to the given name

Definition at line 130 of file giftdetails.cpp.

References giftName.

Referenced by giftDetails().

5.9.4.12 string giftDetails::updateString ()

returns a string suitable to be used in update∗ methods of SQLTable.

Definition at line 178 of file giftdetails.cpp.

References TimePeriod::getBeginTS(), TimePeriod::getEndTS(), TimeStamp::getTimeStamp(), giftName, initial, period, and remaining.

5.9.5 Field Documentation

5.9.5.1 bool giftDetails::empty `[private]`

Definition at line 50 of file giftdetails.h.

Referenced by giftDetails(), and isEmpty().

5.9.5.2 u_int giftDetails::giftid `[private]`

Definition at line 35 of file giftdetails.h.

Referenced by getGiftId(), and giftDetails().

5.9.5.3 string giftDetails::giftName `[private]`

Definition at line 38 of file giftdetails.h.

Referenced by getName(), insertString(), setName(), and updateString().

5.9.5.4 int giftDetails::initial `[private]`

Definition at line 44 of file giftdetails.h.

Referenced by getInitialQuantity(), insertString(), setInitialQuantity(), and updateString().

5.9.5.5 TimePeriod giftDetails::period `[private]`

Definition at line 41 of file giftdetails.h.

Referenced by giftDetails(), insertString(), and updateString().

5.9.5.6 int giftDetails::remaining `[private]`

Definition at line 47 of file giftdetails.h.

Referenced by getCurrentQuantity(), insertString(), operator–(), setCurrentQuantity(), and updateString().

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

- giftdetails.h
- giftdetails.cpp

5.10 Logger Class Reference

`#include <logger.h>`

5.10.1 Detailed Description

This class offers a totally independent way to keep logs in a program. It allows asynchronous logging to the stdout and/or a file (timestamped). It runs in a separate thread and uses ACE's queueing mechanisms to avoid thrashing of concurrently written messages. Its use is simple, just call the logmsg() method with a string or a char ∗ object.

Definition at line 44 of file logger.h.

Public Methods

- Logger (output_t out=OUTPUT_BOTH, string logfilename="")

  The standard constructor, can be called with no arguments.

- virtual ∼Logger ()

  Typical destructor.

- virtual int open (void ∗)

  Virtual open, starts the thread and opens the file.

- virtual int close (u_long flags=0)

  Virtual close, stops the thread and closes the file.

- virtual int svc (void)

This method handles the dequeueing of the messages.

- void setLogFileName (string filename)

  Changes the filename of the logfile.

- string getLogFileName ()

  Returns the filename of the logfile.

- ACE_Future< u_long > logMsg (string msg)

  Log a message.

- u_long logMsg_i (string msg)

  Actual implementation of the Logger.

Private Attributes

- output_t out_

  Specify the output method.

- string logfilename_

  The filename to write the logs into.

- ofstream logFile

  The standard C++ ofstream of the logfile.

- ACE_Thread_Mutex mutex_

  The mutex mechanism, we use ACE's Guard.

- ACE_Activation_Queue activation_queue_

  The queue to keep the method objects.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Logger::Logger (output_t *out* = OUTPUT_BOTH, string *logfilename* = "")

The constructor. It's simple in form and its only important function, other than to set the member variables of the object, is to open the logfile for writing (if it is specified in out_).

Definition at line 19 of file logger.cpp.

References logFile, logfilename_, out_, OUTPUT_BOTH, OUTPUT_FILEONLY, and output_t.

5.10.2.2 Logger::∼Logger () `[virtual]`

The destructor just calls the method close().

Definition at line 30 of file logger.cpp.

References close().

5.10.3 Member Function Documentation

5.10.3.1 int Logger::close (u_long *flags* = 0) [virtual]

Called when the active object is destroyed. Just closes the file if it is open.

Definition at line 47 of file logger.cpp.

References logFile, out_, OUTPUT_BOTH, and OUTPUT_FILEONLY.

Referenced by ∼Logger().

5.10.3.2 string Logger::getLogFileName ()

Definition at line 93 of file logger.cpp.

References logfilename_.

5.10.3.3 ACE_Future< u_long > Logger::logMsg (string *msg*)

This method is called asynchronously. It logs the message. It actually creates a future object that will hold the result of the action and puts the method object (of type logMsg_MO) to the activation_queue_ of the Logger object. This in turn is handled by svc() and the actual logging method logMsg_i() is called to do the logging.

Definition at line 106 of file logger.cpp.

References activation_queue_.

Referenced by Day::assignPrizes(), Day::calculateEstNoMsgs(), SQLiteConnection::commitTransaction(), Day::Day(), SQLTable< key, data >::logMsg(), Contest::logMsg(), SQLiteConnection::logTransaction(), Client_Handler::process(), Processor::process_i(), SQLiteConnection::reconnect(), and SQLiteConnection::SQLiteConnection().

5.10.3.4 u_long Logger::logMsg_i (string *msg*)

The actual method to do the logging. It uses ACE_DEBUG to write the content to stdout, and timestamped C++ stream I/O for the file.

Definition at line 119 of file logger.cpp.

References logFile, mutex_, out_, OUTPUT_BOTH, OUTPUT_FILEONLY, and OUTPUT_STDOUT.

5.10.3.5 int Logger::open (void ∗) [virtual]

The open() method where the active object is activated Create a detached thread to handle the logging.

Definition at line 38 of file logger.cpp.

Referenced by Contest::initLogger(), and SQLiteConnection::SQLiteConnection().

5.10.3.6 void Logger::setLogFileName (string *filename*)

Set the log filename to the given one. Also, close and reopen the logfile with the new filename.

Definition at line 83 of file logger.cpp.

References logFile, logfilename_, out_, OUTPUT_BOTH, and OUTPUT_FILEONLY.

Referenced by SQLiteConnection::changeLogFilename().

5.10.3.7 int Logger::svc (void) `[virtual]`

The svc() method is the one that does all the work. The thread created will run in an infinite loop waiting for method objects to be enqueued on the private activation queue. Once a method object is inserted in the queue the thread wakes up dequeues the object and then invokes the call() method on the object it just dequeued. If there are no method objects on the activation queue the task blocks and falls asleep.

Definition at line 63 of file logger.cpp.

5.10.4 Field Documentation

5.10.4.1 ACE_Activation_Queue Logger::activation_queue_ `[private]`

Definition at line 86 of file logger.h.

Referenced by logMsg().

5.10.4.2 ofstream Logger::logFile `[private]`

Definition at line 52 of file logger.h.

Referenced by close(), Logger(), logMsg_i(), and setLogFileName().

5.10.4.3 string Logger::logfilename_ `[private]`

Definition at line 49 of file logger.h.

Referenced by getLogFileName(), Logger(), and setLogFileName().

5.10.4.4 ACE_Thread_Mutex Logger::mutex_ `[private]`

Definition at line 83 of file logger.h.

Referenced by logMsg_i().

5.10.4.5 output_t Logger::out_ `[private]`

Definition at line 46 of file logger.h.

Referenced by close(), Logger(), logMsg_i(), and setLogFileName().

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

- logger.h
- logger.cpp

5.11 logMsg_MO Class Reference

```
#include <logmsg_mo.h>
```

5.11.1 Detailed Description

The philosophy behind Method Objects is described in ACE Tutorial and the C++ NP. In practice, a method object returns a future object containing the result object of the actual implementation method that is called in call().

The logMsg_MO class is the method object that is queued by the logger. When ready, the logger object (or actually its svc() method) calls the call() method in the MO object which in turn calls the logMsg_i() method of the Logger.

Definition at line 32 of file logmsg_mo.h.

Public Methods

- logMsg_MO (Logger ∗logger, string msg, ACE_Future< u_long > &future_result)

  The default Constructor, takes a pointer to the logger, the msg string and a future object.

- virtual ~logMsg_MO ()

  Destructor.

- virtual int call (void)

  The call() method will be called by the svc() of the Logger Active Object.

Private Attributes

- Logger ∗ logger_

  Pointer to the Logger object.

- string msg_

  The message to be logged.

- ACE_Future< u_long > future_result_

  The ACE Future object that is returned.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 logMsg_MO::logMsg_MO (Logger ∗ *logger*, string *msg*, ACE_Future< u_long > & *future_result*)

The default constructor. It just initializes the member variables of the object (logger_, msg_ and future_result_).

Definition at line 17 of file logmsg_mo.cpp.

5.11.2.2 logMsg_MO::~logMsg_MO () `[virtual]`

Dummy destructor, we don't allocate anything dynamically.

Definition at line 26 of file logmsg_mo.cpp.

5.11.3 Member Function Documentation

5.11.3.1 int logMsg_MO::call (void) `[virtual]`

We don't have a lot to do. We just create a future object containing the result of logMsg_i(). What this means is that the logMsg_MO will finish immediately, but the result (the return value of call()) will remain uninitialized until the Logger object MO queue actually reaches this object.

Definition at line 37 of file logmsg_mo.cpp.

References future_result_.

5.11.4 Field Documentation

5.11.4.1 ACE_Future<u_long> logMsg_MO::future_result_ [private]

Definition at line 50 of file logmsg_mo.h.

Referenced by call().

5.11.4.2 Logger* logMsg_MO::logger_ [private]

Definition at line 44 of file logmsg_mo.h.

5.11.4.3 string logMsg_MO::msg_ [private]

Definition at line 47 of file logmsg_mo.h.

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

- logmsg_mo.h
- logmsg_mo.cpp

5.12 Message_Block_Guard Class Reference

Public Methods

- Message_Block_Guard (ACE_Message_Block *&mb)
- ~Message_Block_Guard (void)

Protected Attributes

- ACE_Message_Block *& mb_

5.12.1 Constructor & Destructor Documentation

5.12.1.1 Message_Block_Guard::Message_Block_Guard (ACE_Message_Block *& *mb*) [inline]

Definition at line 145 of file thread_pool.cpp.

References mb_.

5.12.1.2 Message_Block_Guard::~Message_Block_Guard (void) [inline]

Definition at line 150 of file thread_pool.cpp.

References mb_.

5.12.2 Field Documentation

5.12.2.1 ACE_Message_Block∗& Message_Block_Guard::mb_ `[protected]`

Definition at line 156 of file thread_pool.cpp.

Referenced by Message_Block_Guard(), and ∼Message_Block_Guard().

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

- thread_pool.cpp

5.13 partDetails Class Reference

```
#include <partdetails.h>
```

5.13.1 Detailed Description

This class provides a way to simplify the access and processing of participation data. Along with the other classes giftDetails and Day it provides an interface to the SQLTable class, and we use this facility to populate the SQL tables CODES and WINNINGCODES.

Definition at line 34 of file partdetails.h.

Public Types

- enum { MSISDNSIZE = 13 }

  The size of the string holding the MSISDN is an enum.

Public Methods

- partDetails ()

  dummy constructor that just creates an empty structure.

- partDetails (string newmsisdn, time_t ts, int gid)

  The default constructor, takes the MSISDN as string, the timestamp and the prize id.

- partDetails (string pdstr)

  This constructor builds a partDetails object from a string representation of the parameters.

- ∼partDetails ()

  Dummy destructor, we don't allocate anything dynamically.

- void setMSISDN (string newmsisdn)

  Set the MSISDN to the given string.

- char ∗ getMSISDN ()

  Returns the MSISDN as a C string (null terminated).

- string getMSISDNStr ()

Returns the MSISDN as a C++ string object.

- void setTimestamp (time_t ts)

  Sets the timestamp to the given timestamp.

- time_t getTimestamp ()

  Returns the timestamp as a time_t object.

- void setGiftId (int gid)

  Sets the Prize Id to the given one.

- unsigned char getGiftId ()

  Returns the Prize id of the participation.

- string insertString ()

  This is for the SQLTable<> template class' insert∗ methods.

- string updateString ()

  This is for the SQLTable<> template class' update∗ methods.

- string toHTMLString ()

  This is for the SQLTable<> template class, to create a HTML table.

- bool isEmpty ()

  boolean method to be used in SQLTable<> methods

Private Attributes

- char msisdn [MSISDNSIZE+1]

  The MSISDN is kept as a C string.

- time_t timestamp

  The timestamp of the participation.

- unsigned char giftid

  The id of the prize won (see giftdetails.h for details).

- bool empty

  boolean for SQLTable<> template class

Friends

- ostream & operator<< (ostream &out, partDetails &pd)

  We overload the operator<< to allow a partDetails to be output to a C++ stream.

5.13.2 Member Enumeration Documentation

5.13.2.1 anonymous enum

Enumeration values:
MSISDNSIZE

Definition at line 37 of file partdetails.h.

5.13.3 Constructor & Destructor Documentation

5.13.3.1 partDetails::partDetails ()

Dummy constructor. Just creates an empty object.

Definition at line 17 of file partdetails.cpp.

References empty, and setMSISDN().

5.13.3.2 partDetails::partDetails (string *newmsisdn*, time_t *ts*, int *gid*)

The default constructor. Creates the partDetails object from the given parameters (msisdn, timestamp and prize id). It also sets the boolean empty to true.

Definition at line 28 of file partdetails.cpp.

References empty, and setMSISDN().

5.13.3.3 partDetails::partDetails (string *pdstr*)

This constructor creates a partDetails object, given a string representation of the parameters. These must comma-separated and with the following order:

- giftid
- MSISDN
- timestamp (as type time_t)

Definition at line 43 of file partdetails.cpp.

References empty, setGiftId(), setMSISDN(), setTimestamp(), and strTokenizer().

5.13.3.4 partDetails::∼partDetails ()

Dummy destructor. We don't allocate anything dynamically.

Definition at line 59 of file partdetails.cpp.

5.13.4 Member Function Documentation

5.13.4.1 unsigned char partDetails::getGiftId ()

Return the prize id the partdetails object.

Definition at line 127 of file partdetails.cpp.

References giftid.

Referenced by Day::executeDraw(), insertString(), operator<<(), toHTMLString(), and updateString().

5.13.4.2 char ∗ partDetails::getMSISDN ()

Return the MSISDN as a C string (null terminated).

Definition at line 87 of file partdetails.cpp.

References msisdn.

Referenced by insertString(), operator<<(), toHTMLString(), and updateString().

5.13.4.3 string partDetails::getMSISDNStr ()

Return the MSISDN as a string object.

Definition at line 95 of file partdetails.cpp.

References msisdn.

5.13.4.4 time_t partDetails::getTimestamp ()

Return the timestamp of the partdetails object.

Definition at line 111 of file partdetails.cpp.

References timestamp.

Referenced by operator<<(), and toHTMLString().

5.13.4.5 string partDetails::insertString ()

Returns a string suitable to be used in insert∗ methods of SQLTable.

Definition at line 136 of file partdetails.cpp.

References getGiftId(), getMSISDN(), and timestamp.

5.13.4.6 bool partDetails::isEmpty ()

This is to check if a giftDetails object has been created with the dummy constructor giftDetails().

Definition at line 180 of file partdetails.cpp.

References empty.

Referenced by Processor::process_i().

5.13.4.7 void partDetails::setGiftId (int *gid*)

Set the prize id of the partdetails object to the given value.

Definition at line 119 of file partdetails.cpp.

References giftid.

Referenced by Day::executeDraw(), connectionMsgBlock::getParticipant(), and partDetails().

5.13.4.8 void partDetails::setMSISDN (string *newmsisdn*)

Set the MSISDN of the partdetails object to the given string

Definition at line 79 of file partdetails.cpp.

References msisdn.

Referenced by connectionMsgBlock::getParticipant(), and partDetails().

5.13.4.9 void partDetails::setTimestamp (time_t *ts*)

Set the timestamp of the partdetails object to the given value.

Definition at line 103 of file partdetails.cpp.

References timestamp.

Referenced by connectionMsgBlock::getParticipant(), and partDetails().

5.13.4.10 string partDetails::toHTMLString ()

Returns a string suitable to be used in select∗ methods of SQLTable class to create a HTML table.

Definition at line 162 of file partdetails.cpp.

References getGiftId(), getMSISDN(), and getTimestamp().

5.13.4.11 string partDetails::updateString ()

Returns a string suitable to be used in update∗ methods of SQLTable.

Definition at line 148 of file partdetails.cpp.

References getGiftId(), getMSISDN(), and timestamp.

5.13.5 Friends And Related Function Documentation

5.13.5.1 ostream& operator<< (ostream & *out*, partDetails & *pd*) `[friend]`

We overload the operator<< to allow a partDetails object to be output to a C++ stream. The output will be of the form: MSISDN\tGID\tDATESTRING

Definition at line 68 of file partdetails.cpp.

5.13.6 Field Documentation

5.13.6.1 bool partDetails::empty `[private]`

Definition at line 50 of file partdetails.h.

Referenced by isEmpty(), and partDetails().

5.13.6.2 unsigned char partDetails::giftid `[private]`

Definition at line 47 of file partdetails.h.

Referenced by getGiftId(), and setGiftId().

5.13.6.3 char partDetails::msisdn[MSISDNSIZE+1] `[private]`

Definition at line 41 of file partdetails.h.

Referenced by getMSISDN(), getMSISDNStr(), and setMSISDN().

5.13.6.4 time_t partDetails::timestamp `[private]`

Definition at line 44 of file partdetails.h.

Referenced by getTimestamp(), insertString(), setTimestamp(), and updateString().

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

- partdetails.h
- partdetails.cpp

5.14 Processor Class Reference

`#include <processor.h>`

5.14.1 Detailed Description

This class works in exactly the same way as Logger. Its use is slightly different, in that it is called from Client_Handler to process the messages that are sent to the server, calls Day::executeDraw() on the object and returns the result to the handler, which in turn returns it to evalclient and closes the connection.

Definition at line 37 of file processor.h.

Public Methods

- Processor (Contest ∗myContest)

  The standard constructor is passed a pointer to the parent Contest object.

- virtual ∼Processor ()

  Typical destructor.

- virtual int open (void ∗)

  Virtual open, starts the thread in which the processing takes place.

- virtual int close (u_long flags=0)

  Closes the thread.

- virtual int svc (void)

  This method handles the dequeueing of the messages.

- ACE_Future< int > process (connectionMsgBlock ∗cmb)

  Process the message, or rather queue it for processing.

- int process_i (connectionMsgBlock ∗cmb)

  Actual implementation of the processing method.

Private Attributes

- Contest ∗ parentContest

  Pointer to the parent Contest object.

- ACE_Thread_Mutex mutex_

  The mutex mechanism, we use ACE's Guard.

- ACE_Activation_Queue activation_queue_

  The queue to keep the method objects.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 Processor::Processor (Contest ∗ *contest*)

The constructor just initializes the Contest pointer.

Definition at line 18 of file processor.cpp.

5.14.2.2 Processor::∼Processor () `[virtual]`

Dummy destructor just calls close().

Definition at line 25 of file processor.cpp.

References close().

5.14.3 Member Function Documentation

5.14.3.1 int Processor::close (u_long *flags* = 0) `[virtual]`

Called when the active object is destroyed. A no-op actually.

Definition at line 43 of file processor.cpp.

Referenced by ∼Processor().

5.14.3.2 int Processor::open (void ∗) `[virtual]`

The open() method where the active object is activated Create a detached thread to handle the processing.

Definition at line 34 of file processor.cpp.

Referenced by Contest::initProcessor().

5.14.3.3 ACE_Future< int > Processor::process (connectionMsgBlock ∗ *cmb*)

This method is called asynchronously. It processes the message It actually creates a future object that will hold the result of the action and puts the method object (of type processor_MO) to the activation_queue_ of the Processor object. This in turn is handled by svc() and the actual processing method process_i() is called to do the processing.

Definition at line 80 of file processor.cpp.

References activation_queue_.

Referenced by Client_Handler::process().

5.14.3.4 int Processor::process_i (connectionMsgBlock ∗ *cmb*)

The actual method to do the processing. It keeps everything inside an ACE Guard scope for safety reasons. This one is actually a very important method so we'll do a more thorough analysis:

- First the message is converted into a string and is checked in the Participants table if it exists. If so ID_USEDCODE is returned.
- If we proceed, we have to know the current day, so we use getCurrentDay() from Contest object.
- Increase counter by 1.
- We check if this MSISDN has been used in this day already and if so we increase unique_counter as well.
- Now that we have a valid partDetails object we call Day::executeDraw on this and we have the result of the draw in the same partDetails object.
- If it is a prize, we disable transactions, insert the record in the CODES and WINNINGCODES tables, update the PRIZES table and re-enable the transactions.
- We also update the COUNTERS table, do some logging and return the id of the prize to the calling function/method (Client_Handler::svc() actually).

Definition at line 108 of file processor.cpp.

References SQLiteConnection::commitTransaction(), Day::counter, Contest::Counters, Contest::dbconnection, Day::executeDraw(), TimePeriod::getBeginTS(), Contest::getCurrentDay(), connectionMsgBlock::getParticipantStr(), Day::getTimePeriod(), giftDetails::ID_USEDCODE, SQLTable< string, partDetails >::insertObject(), partDetails::isEmpty(), Contest::logger, Logger::logMsg(), parentContest, Contest::Participants, Day::prizes, Contest::Prizes, SQLTable< string, partDetails >::selectObject(), SQLTable< string, partDetails >::size(), TimeStamp::toString(), Day::unique_counter, SQLTable< string, Day >::updateObject(), SQLTable< string, giftDetails >::updateObject(), Day::updateString(), and Contest::Winners.

5.14.3.5 int Processor::svc (void) [virtual]

The svc() method is the one that does all the work. The thread created will run in an infinite loop waiting for method objects to be enqueued on the private activation queue. Once a method object is inserted in the queue the thread wakes up dequeues the object and then invokes the call() method on the object it just dequeued. If there are no method objects on the activation queue the task blocks and falls asleep.

Definition at line 56 of file processor.cpp.

5.14.4 Field Documentation

5.14.4.1 ACE_Activation_Queue Processor::activation_queue_ [private]

Definition at line 68 of file processor.h.

Referenced by process().

5.14.4.2 ACE_Thread_Mutex Processor::mutex_ [private]

Definition at line 65 of file processor.h.

5.14.4.3 Contest∗ Processor::parentContest [private]

Definition at line 62 of file processor.h.

Referenced by process_i().

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

- processor.h
- processor.cpp

5.15 processor_MO Class Reference

`#include <processor_mo.h>`

5.15.1 Detailed Description

The philosophy behind Method Objects is described in ACE Tutorial and the C++ NP. In practice, a method object returns a future object containing the result object of the actual implementation method that is called in call().

The processor_MO class is the method object that is queued by the Processor. When ready, the processor object (or actually its svc() method) calls the call() method in the MO object which in turn calls the process_i() method of the Processor.

Definition at line 33 of file processor_mo.h.

Public Methods

- processor_MO (Processor *processor, connectionMsgBlock *cmb, ACE_Future< int > &future_result)

  The default Constructor, takes a pointer to the Processor, the message block and a future object.

- virtual ~processor_MO ()

  Dummy destructor.

- virtual int call (void)

  The call() method will be called by the svc() of the Logger Active Object.

Private Attributes

- Processor * processor_

  Pointer to the Processor object.

- connectionMsgBlock * cmb_

  Pointer to the connectionMsgBlock.

- ACE_Future< int > future_result_

  The ACE Future object that is returned.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 processor_MO::processor_MO (Processor ∗ *processor*, connectionMsgBlock ∗ *cmb*, ACE_Future< int > & *future_result*)

The default constructor. It just initializes the member variables of the object (processor_, cmb_ and future_result_).

Definition at line 18 of file processor_mo.cpp.

5.15.2.2 processor_MO::∼processor_MO () `[virtual]`

Dummy destructor, we don't allocate anything dynamically.

Definition at line 28 of file processor_mo.cpp.

5.15.3 Member Function Documentation

5.15.3.1 int processor_MO::call (void) `[virtual]`

We don't have a lot to do. We just create a future object containing the result of process_i(). What this means is that the processor_MO will finish immediately, but the result (the return value of call()) will remain uninitialized until the Processor object MO queue actually reaches this object.

Definition at line 39 of file processor_mo.cpp.

References future_result_.

5.15.4 Field Documentation

5.15.4.1 connectionMsgBlock∗ processor_MO::cmb_ `[private]`

Definition at line 48 of file processor_mo.h.

5.15.4.2 ACE_Future<int> processor_MO::future_result_ `[private]`

Definition at line 51 of file processor_mo.h.

Referenced by call().

5.15.4.3 Processor∗ processor_MO::processor_ `[private]`

Definition at line 45 of file processor_mo.h.

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

- processor_mo.h
- processor_mo.cpp

5.16 SQLiteConnection Class Reference

```
#include <sqliteconnection.h>
```

5.16.1 Detailed Description

SQLite does not provide a C++ API, only a C API, so I had to write one for C++. It is basically a wrapper that encapsulates the sqlitedb pointer and provides a simple way to access the db and execute transactions. There are a few things that have to be pointed out about the use of this class. After you create the class, transactions have to be enabled with beginTransaction() and disabled with disableTransaction(). A transaction is commited with commitTransaction(). This class also provides a logging facility. All SQL commands are also recorded into date-named files in directory TRANSLOGPATH. In the event of an error, it possible to reconstruct the db from these files with the reconstructdb command.

Definition at line 52 of file sqliteconnection.h.

Public Methods

- SQLiteConnection (string dbname, Logger ∗mylogger, bool usetranslog=false)

  Constructor that uses a string as input.

- SQLiteConnection (char ∗dbname, Logger ∗mylogger, bool usetranslog=false)

  Constructor that uses a normal C string as input.

- ∼SQLiteConnection ()

  Default destructor.

- sqlite ∗ getdb ()

  Returns the pointer to the sqlite handler.

- bool isReady ()

  Is the db ready? That is, are we open?

- void reconnect ()

  Close and Reopen the database.

- bool beginTransaction ()

  Begins the transaction.

- bool commitTransaction ()

  Commit the transaction.

- void enableTransactions ()

  Enable the transactions.

- void disableTransactions (bool commit=doCommit)

  Disable (and optionally commit) the current transaction.

- bool transactionsEnabled ()

  Returns true if transactions are enabled.

- void logTransaction (string cmdstr)

  Logs the current transaction to the date-named file.

- bool existsTable (string tablename)

Checks for the existence of an SQL table.

- bool existsIndex (string index, string tablename)

  Checks for the existence of an index for a table.

- size_t getTransactions ()

  Returns the number of SQL commands in the current transaction.

- string pickFilename ()

  Automatically picks a filename for the SQL log files.

- void changeLogFilename ()

  Change the current log filename.

Private Types

- enum { transactions_threshold = 100, max_conflicts = 10 }

  some enumerations that keep some default values

- enum { doCommit = true, dontCommit = false }

  Instead of using true/false, we use the doCommit/dontCommit enums.

Private Attributes

- sqlite ∗ sqlitedb

  The pointer to the sqlite handler.

- char ∗ zErrMsg

  This string holds the actuall SQLite message in the event of an error.

- Logger ∗ logger

  Pointer to the original Logger object and the private one.

- Logger ∗ transactions_log

  Pointer to the original Logger object and the private one.

- string DBname

  The name of the DB file.

- size_t transactions

  The number of transactions held and the conflicts.

- size_t conflicts

  The number of transactions held and the conflicts.

- stringstream transactioncmd

  The complete SQL transaction that is going to be executed.

- bool transactions_enabled

  Booleans that decide if transactions are going to be used and logged.

- bool use_transactions_log

  Booleans that decide if transactions are going to be used and logged.

Friends

- class SQLTable

  Declare SQLTable class as a friend so that it can access private members.

5.16.2 Member Enumeration Documentation

5.16.2.1 anonymous enum `[private]`

Enumeration values:

transactions_threshold

max_conflicts

Definition at line 54 of file sqliteconnection.h.

5.16.2.2 anonymous enum `[private]`

Enumeration values:

doCommit

dontCommit

Definition at line 57 of file sqliteconnection.h.

5.16.3 Constructor & Destructor Documentation

5.16.3.1 SQLiteConnection::SQLiteConnection (string *dbname*, Logger $*$ *mylogger*, bool *usetranslog* = false)

This constructor allows the use of string name for the dbname, instead of just plain C strings.

Definition at line 17 of file sqliteconnection.cpp.

5.16.3.2 SQLiteConnection::SQLiteConnection (char $*$ *dbname*, Logger $*$ *mylogger*, bool *usetranslog* = false)

The default constructor. It opens the SQLite DB (residing in the file dbname), Initializes the transactions logger object and the transactions logfiles. It also resets the counters transactions and conflicts. Transactions are disabled by default.

Definition at line 28 of file sqliteconnection.cpp.

References conflicts, logger, Logger::logMsg(), Logger::open(), OUTPUT_FILEONLY, pickFilename(), sqlitedb, transactions, transactions_enabled, transactions_log, and zErrMsg.

5.16.3.3 SQLiteConnection::∼SQLiteConnection ()

Closes the DB connection.

Definition at line 52 of file sqliteconnection.cpp.

References sqlitedb.

5.16.4 Member Function Documentation

5.16.4.1 bool SQLiteConnection::beginTransaction ()

Begins the current transaction. Basically insert 'BEGIN TRANSACTION' in the transaction string.

Definition at line 169 of file sqliteconnection.cpp.

References transactioncmd, and transactions.

Referenced by SQLTable< key, data >::insertObject(), and SQLTable< key, data >::updateObject().

5.16.4.2 void SQLiteConnection::changeLogFilename ()

If transaction logging is enabled then set the log filename using the method in pickFilename().

Definition at line 84 of file sqliteconnection.cpp.

References pickFilename(), Logger::setLogFileName(), and transactions_log.

Referenced by Contest::handle_timeout().

5.16.4.3 bool SQLiteConnection::commitTransaction ()

Commits the current transaction. Error handling: if there are no available file descriptors or there is an error with the execution of the command executed (SQLITE_MISUSE or SQLITE_CANTOPEN) then the DB is reopened with reconnect(). Then the command is re-executed after a period of one second. If the number of conflicts exceeds max_conflicts then the program gives up with executing this transaction. The number of conflicts is reset after each successfull transaction.

Definition at line 189 of file sqliteconnection.cpp.

References conflicts, logger, Logger::logMsg(), logTransaction(), max_conflicts, reconnect(), sqlitedb, transactioncmd, transactions, and zErrMsg.

Referenced by disableTransactions(), SQLTable< key, data >::insertObject(), Processor::process_i(), and SQLTable< key, data >::updateObject().

5.16.4.4 void SQLiteConnection::disableTransactions (bool *commit* = doCommit)

Disables transactions. Optionally, commits the current transaction Before disabling.

Definition at line 147 of file sqliteconnection.cpp.

References commitTransaction(), transactioncmd, transactions, and transactions_enabled.

Referenced by Contest::handle_timeout().

5.16.4.5 void SQLiteConnection::enableTransactions ()

Enables the transactions. Resets counters and sets transactions_enabled to true.

Definition at line 135 of file sqliteconnection.cpp.

References transactioncmd, transactions, and transactions_enabled.

Referenced by Contest::handle_timeout(), and Contest::initDB().

5.16.4.6 bool SQLiteConnection::existsIndex (string *index*, string *tablename*)

Returns true if the specified index for the table exists in the database.

Definition at line 300 of file sqliteconnection.cpp.

References sqlitedb.

Referenced by SQLTable< key, data >::createIndices().

5.16.4.7 bool SQLiteConnection::existsTable (string *tablename*)

Returns true if the specified table exists in the database.

Definition at line 274 of file sqliteconnection.cpp.

References sqlitedb.

Referenced by Contest::initDB().

5.16.4.8 sqlite * SQLiteConnection::getdb ()

Returns the pointer to the SQLite connection.

Definition at line 93 of file sqliteconnection.cpp.

References sqlitedb.

Referenced by SQLTable< key, data >::createIndices(), SQLTable< key, data >::deleteObject(), SQLTable< key, data >::deleteObjects(), SQLTable< key, data >::drop(), SQLTable< key, data >::insertObject(), SQLTable< key, data >::selectAllObjects(), SQLTable< key, data >::selectDistinctObjects(), SQLTable< key, data >::selectDistinctObjectsMap(), SQLTable< key, data >::selectObject(), SQLTable< key, data >::selectObjects(), SQLTable< key, data >::size(), SQLTable< key, data >::sizeofDistinctObjects(), SQLTable< key, data >::sumColumn(), and SQLTable< key, data >::updateObject().

5.16.4.9 size_t SQLiteConnection::getTransactions ()

Returns the number of SQL commands in the transaction string.

Definition at line 257 of file sqliteconnection.cpp.

References transactions.

5.16.4.10 bool SQLiteConnection::isReady ()

Returns true if the connection is successful.

Definition at line 101 of file sqliteconnection.cpp.

Referenced by Contest::initDB().

5.16.4.11 void SQLiteConnection::logTransaction (string *cmdstr*)

Logs the current transaction to the transaction logfile.

Definition at line 265 of file sqliteconnection.cpp.

References Logger::logMsg(), and transactions_log.

Referenced by commitTransaction(), SQLTable< key, data >::createIndices(), SQLTable< key, data >::deleteObject(), SQLTable< key, data >::deleteObjects(), SQLTable< key, data >::drop(), SQLTable< key, data >::insertObject(), and SQLTable< key, data >::updateObject().

5.16.4.12 string SQLiteConnection::pickFilename ()

Returns a filename to be used for logging. Its path will be on directory TRANSLOGPATH, and the basename will be the current date (YYYYMMDD format). There will be no overwriting of existing files, instead a counter appended to the filename will be used. the suffix will be TRANSLOGSUFFIX (default .sql).

Definition at line 66 of file sqliteconnection.cpp.

References TRANSLOGPATH, and TRANSLOGSUFFIX.

Referenced by changeLogFilename(), and SQLiteConnection().

5.16.4.13 void SQLiteConnection::reconnect ()

Closes and reopens the database. Should be used if a problem occurs with the execution of some SQL commands. Also resets the counters.

Definition at line 115 of file sqliteconnection.cpp.

References DBname, logger, Logger::logMsg(), sqlitedb, transactioncmd, transactions, and zErrMsg.

Referenced by commitTransaction(), SQLTable< key, data >::deleteObject(), SQLTable< key, data >::insertObject(), and SQLTable< key, data >::updateObject().

5.16.4.14 bool SQLiteConnection::transactionsEnabled ()

Returns true if transactions are enabled.

Definition at line 160 of file sqliteconnection.cpp.

References transactions_enabled.

Referenced by Contest::handle_timeout().

5.16.5 Friends And Related Function Documentation

5.16.5.1 friend class SQLTable [friend]

Definition at line 81 of file sqliteconnection.h.

5.16.6 Field Documentation

5.16.6.1 size_t SQLiteConnection::conflicts [private]

Definition at line 72 of file sqliteconnection.h.

Referenced by commitTransaction(), SQLTable< key, data >::insertObject(), SQLiteConnection(), and SQLTable< key, data >::updateObject().

5.16.6.2 string SQLiteConnection::DBname [private]

Definition at line 69 of file sqliteconnection.h.

Referenced by reconnect().

5.16.6.3 Logger* SQLiteConnection::logger [private]

Definition at line 66 of file sqliteconnection.h.

Referenced by commitTransaction(), SQLTable< key, data >::logMsg(), reconnect(), and SQLiteConnection().

5.16.6.4 sqlite* SQLiteConnection::sqlitedb [private]

Definition at line 60 of file sqliteconnection.h.

Referenced by commitTransaction(), existsIndex(), existsTable(), getdb(), reconnect(), SQLiteConnection(), and ~SQLiteConnection().

5.16.6.5 stringstream SQLiteConnection::transactioncmd [private]

Definition at line 75 of file sqliteconnection.h.

Referenced by beginTransaction(), commitTransaction(), disableTransactions(), enableTransactions(), SQLTable< key, data >::insertObject(), reconnect(), and SQLTable< key, data >::updateObject().

5.16.6.6 size_t SQLiteConnection::transactions [private]

Definition at line 72 of file sqliteconnection.h.

Referenced by beginTransaction(), commitTransaction(), disableTransactions(), enableTransactions(), getTransactions(), SQLTable< key, data >::insertObject(), reconnect(), SQLiteConnection(), and SQLTable< key, data >::updateObject().

5.16.6.7 bool SQLiteConnection::transactions_enabled [private]

Definition at line 78 of file sqliteconnection.h.

Referenced by disableTransactions(), enableTransactions(), SQLTable< key, data >::insertObject(), SQLiteConnection(), transactionsEnabled(), and SQLTable< key, data >::updateObject().

5.16.6.8 Logger * SQLiteConnection::transactions_log [private]

Definition at line 66 of file sqliteconnection.h.

Referenced by changeLogFilename(), logTransaction(), and SQLiteConnection().

5.16.6.9 bool SQLiteConnection::use_transactions_log [private]

Definition at line 78 of file sqliteconnection.h.

5.16.6.10 char* SQLiteConnection::zErrMsg [private]

Definition at line 63 of file sqliteconnection.h.

Referenced by commitTransaction(), reconnect(), and SQLiteConnection().

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

- sqliteconnection.h
- sqliteconnection.cpp

5.17 SQLTable< key, data > Class Template Reference

```
#include <sqltable.h>
```

5.17.1 Detailed Description

template<class key, class data> class SQLTable< key, data >

This class is one of the most important of the program. It allows a uniform and more or less consistent way of accessing the database independently of the actual information in the tables. It deals with most of the important SQL commands in a consistent way. These include INSERT, UPDATE, DELETE, SELECT. It offers a number of methods to assist programming in most of the usual ways one may call the SQL commands. It is designed to complement the SQLiteConnection class and right now it cannot be used with another adapter.

The class is a templated class. It uses two abstract classes, namely a key class and a data class. The key class is the one that is used as the index of the SQL table. It can be a PRIMARY KEY or not, this is declared on the construction of the object (and the creation of the respective table). The only requirement for the key class is to provide a c_str() method that returns a pointer to a C string (that is null-terminated). As for the data class there are a few requirements as well:

- It must provide a method insertString(), which returns the part of the SQL INSERT command, without the key. For example, if the INSERT command would be:

INSERT INTO CODES VALUES('76138768176', 1, '309641768762', 1231444534);

then the insertString() method should return:

1, '309641768762', 1231444534

This should then be used to form the complete INSERT command to be executed.

- Likewise, the data class must provide an updateString() to be used when calling UPDATE on the data.
- For information reasons only, it may provide a toHTMLString() method, that displays a table row of its data.
- Equally important is the existance of two constructors, a dummy that sets a boolean variable "empty" to true, and a constructor that takes as an argument the string representation of the data. The data is first read using a SELECT command, is output in a string as comma-separated values and passed as a parameter to the data class constructor. The constructor should then tokenize the string and use these values to initialize itself. Of course it has to set the empty boolean to false afterwards. It is a generic way and can be used for pretty much any object that can be put in a table.

Definition at line 69 of file sqltable.h.

Public Methods

- SQLTable ()

  Dummy constructor.

- SQLTable (SQLiteConnection ∗db, const string &tname, const string &iname, const string &ts, vector< string > &fnames, vector< string > &ftypes, vector< string > &indexnames, bool primarykey)

  The default constructor.

- SQLTable (const SQLTable &source)

  Copy constructor, copies a SQLTable object to another.

- SQLTable & operator= (const SQLTable &source)

  We overload the assignment operator. Basically the same procedure as the copy constructor.

- ∼SQLTable ()

  The default destructor.

- bool createIndices ()

  Creates the indices described in the vector indices.

- bool drop ()

  Drops the current table from the database.

- pair< key, data > selectObject (size_t index)

  Returns a key, data pair of the record that exists in the position index.

- data selectObject (key val, const string &iname)

  Returns the data object of the record where the field iname = val.

- map< key, data > ∗ selectObjects (const key &from, const key &to, const string &iname)

  Returns a map of <key, data> objects where iname field is in the range (from, to).

- map< key, data > ∗ selectObjects (TimePeriod &tp)

  Returns a map of <key, data> objects that have timestamps in the period tp.

- map< key, data > ∗ selectObjects (TimeStamp &ts, const string &tpprefix, const string &qtyname)

  Returns a map of <key, data> objects that their timeperiods include timestamp ts and qtyname != 0.

- map< key, data > ∗ selectObjects (vector< key > &objs, const string &iname)

  Return a map of <key, data> objects where the field iname, has values from the vector objs.

- map< key, data > ∗ selectAllObjects ()

  Return a map of <key, data> of all objects.

- map< key, data > ∗ selectDistinctObjects (TimePeriod &tp, const string &iname)

  Return a map of the UNIQUE objects (using field iname) in the timeperiod tp.

- map< key, data > ∗ selectDistinctObjects (const string &iname)

  Return a map of all UNIQUE objects (using field iname).

- void selectDistinctObjectsMap (ostream &out, TimePeriod &tp, const string &iname)

  Write the UNIQUE objects (using field iname) in the timeperiod tp in the stream out as HTML.

- void selectDistinctObjectsMap (ostream &out, const string &iname)

  Write all UNIQUE objects (using field iname) in the stream out as HTML.

- void insertObject (key val, data &obj)

  Inserts the object pair (val, obj) in the table.

- void insertObjects (map< key, data > &objs)

  Inserts the given map of objects in the table.

- void updateObject (const key &val, data &obj, const string &iname)

  Updates the object where field iname has the value val according to the values of data obj.

- void updateObjects (map< key, data > &objs, const string &iname)

  Updates the map of objects, using the index field iname.

- void deleteObject (key &val, const string &iname)

  Deletes the object where field iname has value val.

- void deleteObjects (const key &from, const key &to, const string &iname)

  Deletes the objects where field iname has values in the range from-to.

- void deleteObjects (vector< key > &objs, const string &iname)

  Delete the objects where the field iname takes values from the vector objs.

- size_t size (key val, const string &iname)

  Returns the number of records where field iname has value val.

- size_t size (key from, key to, const string &iname)

  Returns the number of records where field iname has value in the range from-to.

- size_t size (TimePeriod &tp)

  Returns the number of records where the field tsname has values in the timeperiod tp.

- size_t size (const TimeStamp &ts, const string &tpprefix, const string &qtyname)

  Returns the number of records where their timeperiods include timestamp ts and qtyname != 0.

- size_t size ()

  Return the size of the table.

- size_t sizeofDistinctObjects (TimePeriod &tp, const string &iname)

  Return the size of UNIQUE objects (using index iname) in the timeperiod tp.

- size_t sizeofDistinctObjects (const string &iname)

  Return the size of UNIQUE objects (using index iname).

- string sumColumn (key from, key to, const string &iname, const string &colname)

  Return the sum of field colname of records where field iname is in the range from-to.

- string sumColumn (TimePeriod &tp, key val, const string &iname, const string &colname)

  Return the sum of field colname of records where field iname is equal to val and in the timeperiod tp.

- string sumColumn (const TimeStamp &ts, const string &tpprefix, key val, const string &iname, const string &colname)

  Return the sum of field colname of records where field iname is equal to val and their timeperiods include timestamp ts and qtyname != 0.

- string sumColumn (key val, const string &iname, const string &colname)

  Return the sum of field colname of records where field iname is equal to val.

- data operator[ ] (key x)

  Overload the operator[ ] to access the object where indexname is equal to x.

- pair< key, data > operator[ ] (size_t ind)

  Overload the operator[ ] to access the object in the position ind.

- bool isReady ()

  Return true if the table is opened and operational.

- void logMsg (string msg)

  Wrapper around logger object's logMsg.

Private Types

- typedef map< key, vector< data > > sqlmap

  We typedef the map<key, vector<data> > type to sqlmap.

Private Attributes

- bool ready

  This boolean declares whether a table is ready to be used.

- SQLiteConnection ∗ dbcon

  Pointer to the SQLiteConnection object.

- string tablename

  The name of the table.

- string indexname

  The name of the key field of the table.

- string tsname

  The name of the timestamp field.

- vector< string > fieldnames

  A vector that holds the names of all the fields.

- vector< string > fieldtypes

This vector holds the names of the _types)_ of the fields.

- vector< string > indices

  This vector holds the names of the fields to be indexed.

- bool hasPrimaryKey

  If true then the index is also a PRIMARY KEY.

5.17.2 Member Typedef Documentation

5.17.2.1 template<class key, class data> typedef map<key, vector<data> > SQLTable< key, data >::sqlmap `[private]`

Definition at line 98 of file sqltable.h.

Referenced by SQLTable< key, data >::selectDistinctObjectsMap().

5.17.3 Constructor & Destructor Documentation

5.17.3.1 template<class key, class data> SQLTable< key, data >::SQLTable () `[inline]`

Definition at line 102 of file sqltable.h.

5.17.3.2 template<class key, class data> SQLTable< key, data >::SQLTable< key, data > (SQLiteConnection ∗ *db*, const string & *tname*, const string & *iname*, const string & *ts*, vector< string > & *fnames*, vector< string > & *ftypes*, vector< string > & *indexnames*, bool *primarykey*)

This is the constructor. Apart from initializing the member variables dbcon, tablename, indexname, tsname, fieldnames, fieldtypes and hasPrimaryKey, it is responsible to check for the existense of the table and create it if needed. The creation of the table uses the following information:

- The key of the table (the first field) is indexname
- It is declared as PRIMARY KEY if hasPrimaryKey is set to true
- All the fields are created using the names and types in fieldnames and types vectors Upon successfull creation the ready flag is set to true.

5.17.3.3 template<class key, class data> SQLTable< key, data >::SQLTable (const SQLTable< key, data > & *source*)

This is the copy constructor, copies the contents of a source SQLTable to another

Definition at line 237 of file sqltable.h.

5.17.3.4 template<class key, class data> SQLTable< key, data >::∼SQLTable () `[inline]`

Definition at line 117 of file sqltable.h.

5.17.4 Member Function Documentation

5.17.4.1 template<class key, class data> bool SQLTable< key, data >::createIndices ()

This method creates the necessary indices for the current table, if needed. First it checks for the existence of the indices with existsIndex(). If it exists it does a CREATE INDEX on the table, using a UNIQUE index if the field is a PRIMARY KEY. Upon successfull creation the ready flag is set to true. The indices are VERY important to the performance of the database, esp. with very large datasets.

Definition at line 353 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::existsIndex(), SQLiteConnection::getdb(), SQLTable< key, data >::hasPrimaryKey, SQLTable< key, data >::indexname, SQLTable< key, data >::indices, SQLTable< key, data >::logMsg(), SQLiteConnection::logTransaction(), and SQLTable< key, data >::tablename.

5.17.4.2 template<class key, class data> void SQLTable< key, data >::deleteObject (key & *val*, const string & *iname*)

Deletes the object where field has value val from the table. The key class is required to provide a c_str() method.

Definition at line 1105 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), SQLiteConnection::logTransaction(), SQLiteConnection::reconnect(), and SQLTable< key, data >::tablename.

Referenced by SQLTable< key, data >::deleteObjects().

5.17.4.3 template<class key, class data> void SQLTable< key, data >::deleteObjects (vector< key > & *objs*, const string & *iname*)

Deletes the objects included in the given vector objs from the table. Calls deleteObject() for each object.

Definition at line 1201 of file sqltable.h.

References SQLTable< key, data >::deleteObject(), and SQLTable< key, data >::indexname.

5.17.4.4 template<class key, class data> void SQLTable< key, data >::deleteObjects (const key & *from*, const key & *to*, const string & *iname*)

Deletes the object where field has value val in the range (from,to). The key class is required to provide a c_str() method.

Definition at line 1156 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), SQLiteConnection::logTransaction(), and SQLTable< key, data >::tablename.

5.17.4.5 template<class key, class data> bool SQLTable< key, data >::drop ()

Drops the table, NOT a destuctor. The destructor should not delete the table from the database.

Definition at line 413 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), SQLiteConnection::logTransaction(), and SQLTable< key, data >::tablename.

5.17.4.6 template<class key, class data> void SQLTable< key, data >::insertObject (key *val*, data & *obj*)

Inserts the object (val, obj.insertString()) to the table. The data class is required to provide an insertString() method. The key class is required to provide a c_str() method. The same procedure is followed as in SQLiteConnection::commitTransaction().

Definition at line 1018 of file sqltable.h.

References SQLiteConnection::beginTransaction(), SQLiteConnection::commitTransaction(), SQLiteConnection::conflicts, SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), SQLiteConnection::logTransaction(), SQLiteConnection::max_conflicts, SQLiteConnection::reconnect(), SQLTable< key, data >::tablename, SQLiteConnection::transactioncmd, SQLiteConnection::transactions, SQLiteConnection::transactions_enabled, and SQLiteConnection::transactions_threshold.

Referenced by SQLTable< key, data >::insertObjects().

5.17.4.7 template<class key, class data> void SQLTable< key, data >::insertObjects (map< key, data > & *objs*)

Inserts the objects (val, obj.insertString()) included in the given map to the table. Calls insertObject() for each object.

Definition at line 1090 of file sqltable.h.

References SQLTable< key, data >::insertObject().

5.17.4.8 template<class key, class data> bool SQLTable< key, data >::isReady () `[inline]`

Definition at line 225 of file sqltable.h.

5.17.4.9 template<class key, class data> void SQLTable< key, data >::logMsg (string *msg*)

Instead of using directly the logmsg method from the logger object, we wrap it with another one, so that if it (the logger) is not available (at the start or the end of execution) we will still have a logging mechanism available via cout.

Definition at line 1666 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::logger, and Logger::logMsg().

Referenced by SQLTable< key, data >::createIndices(), SQLTable< key, data >::deleteObject(), SQLTable< key, data >::deleteObjects(), SQLTable< key, data >::drop(), SQLTable< key, data >::insertObject(), SQLTable< key, data >::selectAllObjects(), SQLTable< key, data >::selectDistinctObjects(), SQLTable< key, data >::selectDistinctObjectsMap(), SQLTable< key, data >::selectObject(), SQLTable< key, data >::selectObjects(), SQLTable< key, data >::size(), SQLTable< key, data >::sizeofDistinctObjects(), SQLTable< key, data >::sumColumn(), and SQLTable< key, data >::updateObject().

5.17.4.10 template<class key, class data> SQLTable< key, data > & SQLTable< key, data >::operator= (const SQLTable< key, data > & *source*)

This is the assignment operator=, works just like the copy constructor

Definition at line 253 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLTable< key, data >::fieldnames, SQLTable< key, data >::fieldtypes, SQLTable< key, data >::hasPrimaryKey, SQLTable< key, data >::indexname, SQLTable< key, data >::indices, SQLTable< key, data >::ready, SQLTable< key, data >::tablename, and SQLTable< key, data >::tsname.

5.17.4.11] template<class key, class data> pair<key, data> SQLTable< key, data >::operator[ ] (size_t *ind*) [inline]

Definition at line 220 of file sqltable.h.

5.17.4.12] template<class key, class data> data SQLTable< key, data >::operator[ ] (key *x*) [inline]

Definition at line 215 of file sqltable.h.

5.17.4.13 template<class key, class data> map< key, data > ∗ SQLTable< key, data >::selectAllObjects ()

Returns the map of objects <key, data> of all the records. This one should be used with care, esp. with large databases.

Definition at line 713 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.14 template<class key, class data> map< key, data > ∗ SQLTable< key, data >::selectDistinctObjects (const string & *iname*)

Returns the map of objects <key, data> of the UNIQUE records using the field iname.

Definition at line 820 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.15 template<class key, class data> map< key, data > ∗ SQLTable< key, data >::selectDistinctObjects (TimePeriod & *tp*, const string & *iname*)

Returns the map of objects <key, data> of the UNIQUE records where the field tsname has values inside the period tp. We check for uniqueness using the field iname.

Definition at line 766 of file sqltable.h.

References SQLTable< key, data >::dbcon, TimePeriod::getBeginTS(), SQLiteConnection::getdb(), TimePeriod::getEndTS(), TimeStamp::getTimeStamp(), SQLTable< key, data >::logMsg(), SQLTable< key, data >::tablename, and SQLTable< key, data >::tsname.

5.17.4.16 template<class key, class data> void SQLTable< key, data >::selectDistinctObjectsMap (ostream & *out*, const string & *iname*)

This does not return anything. Instead we provide a filestream descriptor and it writes an HTML table of the UNIQUE records (using the field iname). The class data is required to provide a toHTMLString() method. The performance is much greater than creating a map of the objects in RAM, and then writing it to disk.

Definition at line 948 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), SQLTable< key, data >::sqlmap, and SQLTable< key, data >::tablename.

5.17.4.17 template<class key, class data> void SQLTable< key, data >::selectDistinctObjectsMap (ostream & *out*, TimePeriod & *tp*, const string & *iname*)

This does not return anything. Instead we provide a filestream descriptor and it writes an HTML table of the UNIQUE records (using the field iname) where their tsname field is inside the timeperiod tp. The class data is required to provide a toHTMLString() method. The performance is much greater than creating a map of the objects in RAM, and then writing it to disk.

Definition at line 878 of file sqltable.h.

References SQLTable< key, data >::dbcon, TimePeriod::getBeginTS(), SQLiteConnection::getdb(), TimePeriod::getEndTS(), TimeStamp::getTimeStamp(), SQLTable< key, data >::logMsg(), SQLTable< key, data >::tablename, and SQLTable< key, data >::tsname.

5.17.4.18 template<class key, class data> data SQLTable< key, data >::selectObject (key *val*, const string & *iname*)

Returns the pair <key, data> of the record where field iname has value val.

Definition at line 487 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.19 template<class key, class data> pair< key, data > SQLTable< key, data >::selectObject (size_t *index*)

Returns the pair <key, data> of the record which is in the position index of the TABLE. Basically it uses the OFFSET parameter in SELECT.

Definition at line 446 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

Referenced by SQLTable< string, partDetails >::operator[ ](), and SQLTable< key, data >::selectObjects().

5.17.4.20 template<class key, class data> map< key, data > * SQLTable< key, data >::selectObjects (vector< key > & *objs*, const string & *iname*)

Returns the map of objects <key, data> of the records where the field iname takes values given in the vector<key> objs.

Definition at line 692 of file sqltable.h.

References SQLTable< key, data >::indexname, and SQLTable< key, data >::selectObject().

5.17.4.21 template<class key, class data> map< key, data > * SQLTable< key, data >::selectObjects (TimeStamp & *ts*, const string & *tpprefix*, const string & *qtyname*)

Returns the pair <key, data> of the record where the given ts is inside the values of the fields {tpprefix}_begin and {tpprefix}_end and where the value of the field qtyname is != 0.

Definition at line 635 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), TimeStamp::getTimeStamp(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.22 template<class key, class data> map< key, data > ∗ SQLTable< key, data >::selectObjects (TimePeriod & *tp*)

Returns the pair <key, data> of the record where field tsname (which is used to hold timestamp information) is inside the timeperiod tp.

Definition at line 581 of file sqltable.h.

References SQLTable< key, data >::dbcon, TimePeriod::getBeginTS(), SQLiteConnection::getdb(), TimePeriod::getEndTS(), TimeStamp::getTimeStamp(), SQLTable< key, data >::logMsg(), SQLTable< key, data >::tablename, and SQLTable< key, data >::tsname.

5.17.4.23 template<class key, class data> map< key, data > ∗ SQLTable< key, data >::selectObjects (const key & *from*, const key & *to*, const string & *iname*)

Returns the pair <key, data> of the record where field iname has value in the range (from, to).

Definition at line 528 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.24 template<class key, class data> size_t SQLTable< key, data >::size ()

Returns the number of all records.

Definition at line 1495 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.25 template<class key, class data> size_t SQLTable< key, data >::size (const TimeStamp & *ts*, const string & *tpprefix*, const string & *qtyname*)

Returns the number of the records where the given ts is inside the values of the fields {tpprefix}_begin and {tpprefix}_end and where the value of the field qtyname is != 0.

Definition at line 1460 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), TimeStamp::getTimeStamp(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.26 template<class key, class data> size_t SQLTable< key, data >::size (TimePeriod & *tp*)

Returns the number of all the records whom the field tsname (which holds timestamp information) is inside the timeperiod tp.

Definition at line 1361 of file sqltable.h.

References SQLTable< key, data >::dbcon, TimePeriod::getBeginTS(), SQLiteConnection::getdb(), TimePeriod::getEndTS(), TimeStamp::getTimeStamp(), SQLTable< key, data >::logMsg(), SQLTable< key, data >::tablename, and SQLTable< key, data >::tsname.

5.17.4.27 template<class key, class data> size_t SQLTable< key, data >::size (key *from*, key *to*, const string & *iname*)

Returns the number of all the records whom the field iname has value in the range (from, to).

Definition at line 1331 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.28 template<class key, class data> size_t SQLTable< key, data >::size (key *val*, const string & *iname*)

Returns the number of all the records whom the field iname has value equal to val.

Definition at line 1301 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.29 template<class key, class data> size_t SQLTable< key, data >::sizeofDistinctObjects (const string & *iname*)

Returns the number of all the unique records (using the index iname).

Definition at line 1426 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.30 template<class key, class data> size_t SQLTable< key, data >::sizeofDistinctObjects (TimePeriod & *tp*, const string & *iname*)

Returns the number of all the unique records (using the index iname) whom the field tsname (whichholds timestamp information) is inside the timeperiod tp.

Definition at line 1393 of file sqltable.h.

References SQLTable< key, data >::dbcon, TimePeriod::getBeginTS(), SQLiteConnection::getdb(), TimePeriod::getEndTS(), TimeStamp::getTimeStamp(), SQLTable< key, data >::logMsg(), SQLTable< key, data >::tablename, and SQLTable< key, data >::tsname.

5.17.4.31 template<class key, class data> string SQLTable< key, data >::sumColumn (key *val*, const string & *iname*, const string & *colname*)

Returns the sum of the field colname, of all the records for which the file iname is equal to val.

Definition at line 1634 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.32 template<class key, class data> string SQLTable< key, data >::sumColumn (const TimeStamp & *ts*, const string & *tpprefix*, key *val*, const string & *iname*, const string & *colname*)

Returns the sum of the field colname, of all the records for which the given ts is inside the values of the fields {tpprefix}_begin and {tpprefix}_end and where the value of the field qtyname is != 0.

Definition at line 1592 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), TimeStamp::getTimeStamp(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.33 template<class key, class data> string SQLTable< key, data >::sumColumn (TimePeriod & *tp*, key *val*, const string & *iname*, const string & *colname*)

Returns the sum of the field colname, of all the records for which the field iname is equal to val and the field tsname is inside the timeperiod tp.

Definition at line 1555 of file sqltable.h.

References SQLTable< key, data >::dbcon, TimePeriod::getBeginTS(), SQLiteConnection::getdb(), TimePeriod::getEndTS(), TimeStamp::getTimeStamp(), SQLTable< key, data >::logMsg(), SQLTable< key, data >::tablename, and SQLTable< key, data >::tsname.

5.17.4.34 template<class key, class data> string SQLTable< key, data >::sumColumn (key *from*, key *to*, const string & *iname*, const string & *colname*)

Returns the sum of the field colname, of all the records for which the field iname has value in the range (from, to).

Definition at line 1523 of file sqltable.h.

References SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::logMsg(), and SQLTable< key, data >::tablename.

5.17.4.35 template<class key, class data> void SQLTable< key, data >::updateObject (const key & *val*, data & *obj*, const string & *iname*)

Updates the object where field iname has value iname, using the values in the object data. The data class is required to provide for an updateString() method. The same procedure is followed as in SQLiteConnection::commitTransaction().

Definition at line 1218 of file sqltable.h.

References SQLiteConnection::beginTransaction(), SQLiteConnection::commitTransaction(), SQLiteConnection::conflicts, SQLTable< key, data >::dbcon, SQLiteConnection::getdb(), SQLTable< key, data >::indexname, SQLTable< key, data >::logMsg(), SQLiteConnection::logTransaction(), SQLiteConnection::max_conflicts, SQLiteConnection::reconnect(), SQLTable< key, data >::tablename, SQLiteConnection::transactioncmd, SQLiteConnection::transactions, SQLiteConnection::transactions_enabled, and SQLiteConnection::transactions_threshold.

Referenced by SQLTable< key, data >::updateObjects().

5.17.4.36 template<class key, class data> void SQLTable< key, data >::updateObjects (map< key, data > & *objs*, const string & *iname*)

Updates the objects (val, obj.updateString()) included in the given map to the table. Calls updateObject() for each object.

Definition at line 1287 of file sqltable.h.

References SQLTable< key, data >::indexname, and SQLTable< key, data >::updateObject().

5.17.5 Field Documentation

5.17.5.1 template<class key, class data> SQLiteConnection∗ SQLTable< key, data >::dbcon `[private]`

Definition at line 74 of file sqltable.h.

Referenced by SQLTable< key, data >::createIndices(), SQLTable< key, data >::deleteObject(), SQLTable< key, data >::deleteObjects(), SQLTable< key, data >::drop(), SQLTable< key, data >::insertObject(), SQLTable< key, data >::logMsg(), SQLTable< key, data >::operator=(), SQLTable< key, data

>::selectAllObjects(), SQLTable< key, data >::selectDistinctObjects(), SQLTable< key, data >::selectDistinctObjectsMap(), SQLTable< key, data >::selectObject(), SQLTable< key, data >::selectObjects(), SQLTable< key, data >::size(), SQLTable< key, data >::sizeofDistinctObjects(), SQLTable< key, data >::sumColumn(), and SQLTable< key, data >::updateObject().

5.17.5.2 template<class key, class data> vector<string> SQLTable< key, data >::fieldnames `[private]`

Definition at line 86 of file sqltable.h.

Referenced by SQLTable< key, data >::operator=().

5.17.5.3 template<class key, class data> vector<string> SQLTable< key, data >::fieldtypes `[private]`

Definition at line 89 of file sqltable.h.

Referenced by SQLTable< key, data >::operator=().

5.17.5.4 template<class key, class data> bool SQLTable< key, data >::hasPrimaryKey `[private]`

Definition at line 95 of file sqltable.h.

Referenced by SQLTable< key, data >::createIndices(), and SQLTable< key, data >::operator=().

5.17.5.5 template<class key, class data> string SQLTable< key, data >::indexname `[private]`

Definition at line 80 of file sqltable.h.

Referenced by SQLTable< key, data >::createIndices(), SQLTable< key, data >::deleteObjects(), SQLTable< key, data >::operator=(), SQLTable< string, partDetails >::operator[ ](), SQLTable< key, data >::selectObjects(), SQLTable< key, data >::updateObject(), and SQLTable< key, data >::updateObjects().

5.17.5.6 template<class key, class data> vector<string> SQLTable< key, data >::indices `[private]`

Definition at line 92 of file sqltable.h.

Referenced by SQLTable< key, data >::createIndices(), and SQLTable< key, data >::operator=().

5.17.5.7 template<class key, class data> bool SQLTable< key, data >::ready `[private]`

Definition at line 71 of file sqltable.h.

Referenced by SQLTable< string, partDetails >::isReady(), SQLTable< key, data >::operator=(), and SQLTable< string, partDetails >::SQLTable().

5.17.5.8 template<class key, class data> string SQLTable< key, data >::tablename `[private]`

Definition at line 77 of file sqltable.h.

Referenced by SQLTable< key, data >::createIndices(), SQLTable< key, data >::deleteObject(), SQLTable< key, data >::deleteObjects(), SQLTable< key, data >::drop(), SQLTable< key, data >::insertObject(), SQLTable< key, data >::operator=(), SQLTable< key, data >::selectAllObjects(), SQLTable< key, data >::selectDistinctObjects(), SQLTable< key, data >::selectDistinctObjectsMap(), SQLTable< key, data >::selectObject(), SQLTable< key, data >::selectObjects(), SQLTable< key, data >::size(), SQLTable< key, data >::sizeofDistinctObjects(), SQLTable< key, data >::sumColumn(), and SQLTable< key, data >::updateObject().

5.17.5.9 template<class key, class data> string SQLTable< key, data >::tsname `[private]`

Definition at line 83 of file sqltable.h.

Referenced by SQLTable< key, data >::operator=(), SQLTable< key, data >::selectDistinctObjects(), SQLTable< key, data >::selectDistinctObjectsMap(), SQLTable< key, data >::selectObjects(), SQLTable< key, data >::size(), SQLTable< key, data >::sizeofDistinctObjects(), and SQLTable< key, data >::sumColumn().

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

- sqltable.h

5.18 Thread_Pool Class Reference

`#include <thread_pool.h>`

5.18.1 Detailed Description

This object creates a collection of thread objects that will do the processing from a queue of event handlers. The base thread object is ACE_Task<>.

Definition at line 51 of file thread_pool.h.

Public Types

- typedef ACE_Task< ACE_MT_SYNCH > inherited
- typedef ACE_Atomic_Op< ACE_Mutex, int > counter_t
- enum size_t { default_pool_size_ = 10 }

Public Methods

- Thread_Pool (void)

  Basic constructor.

- int start (int pool_size=default_pool_size_)
- virtual int stop (void)

  Shut down the thread pool.

- int enqueue (ACE_Event_Handler ∗handler)

Protected Methods

- int svc (void)

Protected Attributes

- counter_t active_threads_

5.18.2 Member Typedef Documentation

5.18.2.1 typedef ACE_Atomic_Op<ACE_Mutex, int> Thread_Pool::counter_t

Another handy ACE template is ACE_Atomic_Op<>. When parameterized, this allows is to have a thread-safe counting object. The typical arithmetic operators are all internally thread-safe so that you can share it across threads without worrying about any contention issues.

Definition at line 84 of file thread_pool.h.

Referenced by Counter_Guard::Counter_Guard().

5.18.2.2 typedef ACE_Task<ACE_MT_SYNCH> Thread_Pool::inherited

Definition at line 54 of file thread_pool.h.

5.18.3 Member Enumeration Documentation

5.18.3.1 enum Thread_Pool::size_t

Provide an enumeration for the default pool size. By doing this, other objects can use the value when they want a default.

Enumeration values:
default_pool_size_

Definition at line 58 of file thread_pool.h.

5.18.4 Constructor & Destructor Documentation

5.18.4.1 Thread_Pool::Thread_Pool (void)

All we do here is initialize our active thread counter.

Definition at line 10 of file thread_pool.cpp.

5.18.5 Member Function Documentation

5.18.5.1 int Thread_Pool::enqueue (ACE_Event_Handler ∗ *handler*)

To use the thread pool, you have to put some unit of work into it. Since we're dealing with event handlers (or at least their derivatives), I've chosen to provide an enqueue() method that takes a pointer to an ACE_Event_Handler. The handler's handle_input() method will be called, so your object has to know when it is being called by the thread pool.

Definition at line 50 of file thread_pool.cpp.

Referenced by Client_Handler::handle_input(), and stop().

5.18.5.2 int Thread_Pool::start (int *pool_size* = default_pool_size_)

Starting the thread pool causes one or more threads to be activated. When activated, they all execute the svc() method declared below.

Definition at line 19 of file thread_pool.cpp.

Referenced by Client_Acceptor::open().

5.18.5.3 int Thread_Pool::stop (void) `[virtual]`

Definition at line 27 of file thread_pool.cpp.

References active_threads_, and enqueue().

Referenced by Client_Acceptor::close().

5.18.5.4 int Thread_Pool::svc (void) `[protected]`

Our svc() method will dequeue the enqueued event handler objects and invoke the handle_input() method on each. Since we're likely running in more than one thread, idle threads can take work from the queue while other threads are busy executing handle_input() on some object.

Definition at line 162 of file thread_pool.cpp.

References active_threads_.

5.18.6 Field Documentation

5.18.6.1 counter_t Thread_Pool::active_threads_ `[protected]`

We use the atomic op to keep a count of the number of threads in which our svc() method is running. This is particularly important when we want to close() it down!

Definition at line 98 of file thread_pool.h.

Referenced by stop(), and svc().

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

- thread_pool.h
- thread_pool.cpp

5.19 TimePeriod Class Reference

```
#include <timeperiod.h>
```

5.19.1 Detailed Description

Since we're not dealing with single timestamps, but with time periods, that is, with pairs of timestamps (begin, end) we needed something more than the TimeStamp class. Hence the TimePeriod class

Definition at line 26 of file timeperiod.h.

Public Methods

- TimePeriod (const TimePeriod &source)

The copy constructor.

- TimePeriod (TimeStamp begints, TimeStamp endts)

  Constructor, accepting two TimeStamp objects as parameters.

- TimePeriod (long begints=0, long endts=86400)

  Default constructor, accepting two time_t objects as parameters, or no parameters at all.

- ∼TimePeriod ()

  Dummy destructor.

- TimeStamp & getBeginTS ()

  Returns a reference to the begin TimeStamp.

- TimeStamp & getEndTS ()

  Returns a reference to the end TimeStamp.

- bool dateInPeriod (TimeStamp ts)

  Returns true if given timestamp (TimeStamp object) is _in_ the current TimePeriod object.

- bool dateInPeriod (time_t ts)

  Returns true if given timestamp (time_t) is _in_ the current TimePeriod object.

- string toString (bool shortString=false)

  Returns a short or long string representation of the time period.

Private Attributes

- TimeStamp beginTimeStamp

  The Timestamp of the start of the period.

- TimeStamp endTimeStamp

  The Timestamp of the start of the period.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 TimePeriod::TimePeriod (const TimePeriod & *source*)

The copy constructor. Initializes a TimePeriod object using the contents of another.

Definition at line 17 of file timeperiod.cpp.

References beginTimeStamp, and endTimeStamp.

5.19.2.2 TimePeriod::TimePeriod (TimeStamp *begints*, TimeStamp *endts*)

A constructor. Takes two TimeStamp objects to create a TimePeriod object.

Definition at line 27 of file timeperiod.cpp.

5.19.2.3 TimePeriod::TimePeriod (long *begints* = 0, long *endts* = 86400)

The default constructor. Takes two timestamps (but this time of type time_t) to create a TimePeriod object. These values can be ommitted, taking defaults of 0 and 86400 respectively, meaning the TimePeriod is one day, starting Jan 1, 1970.

Definition at line 39 of file timeperiod.cpp.

5.19.2.4 TimePeriod::∼TimePeriod ()

Dummy destructor. We don't allocate anything dynamically

Definition at line 47 of file timeperiod.cpp.

5.19.3 Member Function Documentation

5.19.3.1 bool TimePeriod::dateInPeriod (time_t *ts*)

If ts (type time_t) is between beginTimeStamp and endTimeStamp then the method returns true, otherwise it returns false.

Definition at line 69 of file timeperiod.cpp.

5.19.3.2 bool TimePeriod::dateInPeriod (TimeStamp *ts*)

If ts (type TimeStamp object) is between beginTimeStamp and endTimeStamp then the method returns true, otherwise it returns false.

Definition at line 56 of file timeperiod.cpp.

References beginTimeStamp, and endTimeStamp.

Referenced by Day::assignPrizes().

5.19.3.3 TimeStamp & TimePeriod::getBeginTS ()

Returns the beginning of the period TimeStamp object

Definition at line 80 of file timeperiod.cpp.

References beginTimeStamp.

Referenced by Day::assignPrizes(), Day::Day(), Contest::giftIsGiven(), giftDetails::insertString(), Processor::process_i(), SQLTable< key, data >::selectDistinctObjects(), SQLTable< key, data >::selectDistinctObjectsMap(), SQLTable< key, data >::selectObjects(), SQLTable< key, data >::size(), SQLTable< key, data >::sizeofDistinctObjects(), SQLTable< key, data >::sumColumn(), and giftDetails::updateString().

5.19.3.4 TimeStamp & TimePeriod::getEndTS ()

Returns the end of the period TimeStamp object

Definition at line 88 of file timeperiod.cpp.

References endTimeStamp.

Referenced by Contest::handle_timeout(), giftDetails::insertString(), SQLTable< key, data >::selectDistinctObjects(), SQLTable< key, data >::selectDistinctObjectsMap(), SQLTable< key, data >::selectObjects(), SQLTable< key, data >::size(), SQLTable< key, data >::sizeofDistinctObjects(), SQLTable< key, data >::sumColumn(), and giftDetails::updateString().

5.19.3.5 string TimePeriod::toString (bool *shortString* = false)

Returns a string representation of the TimePeriod object. If the parameter shortString is false, then it just returns the locale string representation of the TimeStamp objects (just as date command would) separated by '-'. Otherwise the dates are of the form YYYYMMDD.

Definition at line 100 of file timeperiod.cpp.

References beginTimeStamp, endTimeStamp, and TimeStamp::toString().

Referenced by Day::Day().

5.19.4 Field Documentation

5.19.4.1 TimeStamp TimePeriod::beginTimeStamp `[private]`

Definition at line 28 of file timeperiod.h.

Referenced by dateInPeriod(), getBeginTS(), TimePeriod(), and toString().

5.19.4.2 TimeStamp TimePeriod::endTimeStamp `[private]`

Definition at line 31 of file timeperiod.h.

Referenced by dateInPeriod(), getEndTS(), TimePeriod(), and toString().

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

- timeperiod.h
- timeperiod.cpp

5.20 TimeStamp Class Reference

`#include <timestamp.h>`

5.20.1 Detailed Description

Since we are keeping all timing information in UNIX timestamp form, we have created a class to handle time and date information in that form. This class allows a creation of an object given a date, a timeinfo struct, a timestamp. It also allows simple arithmetic operations and comparisons to be performed between TimeStamp objects. One of the most useful methods is the toString() which returns a string representation in short or long form.

Definition at line 34 of file timestamp.h.

Public Methods

- TimeStamp (const TimeStamp &source)

  The copy constructor.

- TimeStamp (int year, int month, int mday, int hour=0, int mins=0, int secs=0)

  Construct a TimeStamp object given a date.

- TimeStamp (struct tm ∗timeinfo)

  Construct a TimeStamp given a timeinfo structure.

- TimeStamp (time_t ts=0)

  Construct a TimeStamp object given a UNIX timestamp.

- ~TimeStamp ()

  Typical destructor.

- void operator++ (int)

  Add a day to the current timestamp.

- bool operator> (time_t ts)

  Comparison operator between current TimeStamp and an integer.

- bool operator> (TimeStamp ts)

  Comparison operator between current and another TimeStamp object.

- bool operator< (time_t ts)

  Comparison operator between current TimeStamp and an integer.

- bool operator< (TimeStamp ts)

  Comparison operator between current and another TimeStamp object.

- bool operator== (TimeStamp ts)

  Equality operator between current and another TimeStamp object.

- bool operator== (time_t ts)

  Comparison operator between current TimeStamp and an integer.

- bool operator!= (TimeStamp ts)

  Difference operator between current and another TimeStamp object.

- bool operator!= (time_t ts)

  Comparison operator between current TimeStamp and an integer.

- const time_t getTimeStamp ()

  Returns the current timestamp in time_t format.

- void setTimeStamp (time_t ts)

  Sets current timestamp to the given value.

- time_t nextDay ()

  Returns the current timestamp + 1 day in time_t format.

- time_t previousDay ()

  Returns the current timestamp - 1 day in time_t format.

- time_t nextWeek ()

  Returns the current timestamp + 1 week in time_t format.

- time_t nextMonth ()

Returns the current timestamp + 1 month in time_t format.

- string toString (bool isShort=false)

  Returns a string representation (short or long form).

Private Attributes

- tm timeinfo

  Internally the class keeps time information in a timeinfo struct.

- time_t timestamp

  ... and a unix timestamp.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 TimeStamp::TimeStamp (const TimeStamp & *source*)

The copy constructor

Definition at line 62 of file timestamp.cpp.

References timeinfo, and timestamp.

5.20.2.2 TimeStamp::TimeStamp (int *year*, int *month*, int *mday*, int *hour* = 0, int *mins* = 0, int *secs* = 0)

The default constructor, creates a TimeStamp object given the actual date as parameters. You may omit the hour, mins and secs.

Definition at line 17 of file timestamp.cpp.

References timeinfo, and timestamp.

5.20.2.3 TimeStamp::TimeStamp (struct tm ∗ *ti*)

Another constructor, it can use a timeinfo structure to create the TimeStamp.

Definition at line 37 of file timestamp.cpp.

References timeinfo, and timestamp.

5.20.2.4 TimeStamp::TimeStamp (time_t *ts* = 0)

This constructor uses the actual timestamp (number of seconds since Jan 1, 1970) to create the TimeStamp object.

Definition at line 50 of file timestamp.cpp.

References timeinfo, and timestamp.

5.20.2.5 TimeStamp::∼TimeStamp ()

Dummy destructor.

Definition at line 74 of file timestamp.cpp.

5.20.3 Member Function Documentation

5.20.3.1 const time_t TimeStamp::getTimeStamp ()

Returns the UNIX timestamp of the current object

Definition at line 196 of file timestamp.cpp.

References timestamp.

Referenced by Day::Day(), Contest::initTimers(), giftDetails::insertString(), operator!=(), operator<(), operator==(), operator>(), SQLTable< key, data >::selectDistinctObjects(), SQLTable< key, data >::selectDistinctObjectsMap(), SQLTable< key, data >::selectObjects(), Contest::shutdown(), SQLTable< key, data >::size(), SQLTable< key, data >::sizeofDistinctObjects(), SQLTable< key, data >::sumColumn(), and giftDetails::updateString().

5.20.3.2 time_t TimeStamp::nextDay ()

Returns the UNIX timestamp of the next day of the current object, that is 24 hours from the timestamp of this object. Note: In contrast to the operator++ and –, this method does not change the current object but creates a copy of it.

Definition at line 222 of file timestamp.cpp.

References timeinfo.

Referenced by Day::Day(), and Contest::handle_timeout().

5.20.3.3 time_t TimeStamp::nextMonth ()

Returns the UNIX timestamp of the next month of the current object, that is 1 month after the timestamp of this object. This is better than adding 30 days, because it also takes care of shorter months such as February. The same notes apply as in nextDay().

Definition at line 290 of file timestamp.cpp.

References timeinfo.

5.20.3.4 time_t TimeStamp::nextWeek ()

Returns the UNIX timestamp of the next week of the current object, that is 7 days after the timestamp of this object. The same notes apply as in nextDay().

Definition at line 266 of file timestamp.cpp.

References timeinfo.

Referenced by Contest::initTimers().

5.20.3.5 bool TimeStamp::operator!= (time_t *ts*)

A TimeStamp may be compared to another TimeStamp object or a UNIX timestamp (which is of time_t). This overloads the non-equality operator!=.

Definition at line 185 of file timestamp.cpp.

References timestamp.

5.20.3.6 bool TimeStamp::operator!= (TimeStamp *ts*)

A TimeStamp may be compared to another TimeStamp object or a UNIX timestamp (which is of time_t). This overloads the non-equality operator!=.

Definition at line 172 of file timestamp.cpp.

References getTimeStamp(), and timestamp.

5.20.3.7 void TimeStamp::operator++ (int)

We have overloaded the ++ operator to increase the date by one day.

Definition at line 81 of file timestamp.cpp.

References timeinfo, and timestamp.

5.20.3.8 bool TimeStamp::operator< (TimeStamp *ts*)

A TimeStamp may be compared to another TimeStamp object or a UNIX timestamp (which is of time_t). This overloads the operator<

Definition at line 107 of file timestamp.cpp.

References getTimeStamp(), and timestamp.

5.20.3.9 bool TimeStamp::operator< (time_t *ts*)

A TimeStamp may be compared to another TimeStamp object or a UNIX timestamp (which is of time_t). This overloads the operator<

Definition at line 94 of file timestamp.cpp.

References timestamp.

5.20.3.10 bool TimeStamp::operator== (time_t *ts*)

A TimeStamp may be compared to another TimeStamp object or a UNIX timestamp (which is of time_t). This overloads the equality operator==.

Definition at line 159 of file timestamp.cpp.

References timestamp.

5.20.3.11 bool TimeStamp::operator== (TimeStamp *ts*)

A TimeStamp may be compared to another TimeStamp object or a UNIX timestamp (which is of time_t). This overloads the equality operator==.

Definition at line 146 of file timestamp.cpp.

References getTimeStamp(), and timestamp.

5.20.3.12 bool TimeStamp::operator> (TimeStamp *ts*)

A TimeStamp may be compared to another TimeStamp object or a UNIX timestamp (which is of time_t). This overloads the operator>

Definition at line 133 of file timestamp.cpp.

References getTimeStamp(), and timestamp.

5.20.3.13 bool TimeStamp::operator> (time_t *ts*)

A TimeStamp may be compared to another TimeStamp object or a UNIX timestamp (which is of time_t). This overloads the operator>

Definition at line 120 of file timestamp.cpp.

References timestamp.

5.20.3.14 time_t TimeStamp::previousDay ()

Returns the UNIX timestamp of the previous day of the current object, that is 24 hours before the timestamp of this object. The same notes apply as in nextDay().

Definition at line 244 of file timestamp.cpp.

References timeinfo.

Referenced by Day::assignPrizes().

5.20.3.15 void TimeStamp::setTimeStamp (time_t *ts*)

Sets the timestamp of the current object to the given one. Instead of just copying the timestamp, it also synchronizes the timeinfo structure as well.

Definition at line 206 of file timestamp.cpp.

References timeinfo, and timestamp.

Referenced by Contest::initTimers().

5.20.3.16 string TimeStamp::toString (bool *isShort* = false)

Returns a string representation of the current TimeStamp object. If isShort is set to false, the representation will be analytical (just as the output of the command date), while if it is true, the string will be of the form YYYYMMDD. This is very useful in filenames and in statistics.

Definition at line 314 of file timestamp.cpp.

References timeinfo, and timestamp.

Referenced by Day::Day(), Contest::initTimers(), operator<<(), Processor::process_i(), and TimePeriod::toString().

5.20.4 Field Documentation

5.20.4.1 struct tm TimeStamp::timeinfo `[private]`

Definition at line 37 of file timestamp.h.

Referenced by nextDay(), nextMonth(), nextWeek(), operator++(), previousDay(), setTimeStamp(), TimeStamp(), and toString().

5.20.4.2 time_t TimeStamp::timestamp `[private]`

Definition at line 40 of file timestamp.h.

Referenced by getTimeStamp(), operator!=(), operator++(), operator<(), operator==(), operator>(), setTimeStamp(), TimeStamp(), and toString().

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

- timestamp.h
- timestamp.cpp

6 File Documentation

6.1 client_acceptor.cpp File Reference

```
#include "client_acceptor.h"
```

6.2 client_acceptor.h File Reference

```
#include "ace/Acceptor.h"
#include "ace/SOCK_Acceptor.h"
#include "client_handler.h"
#include "thread_pool.h"
```

Namespaces

- namespace std

Data Structures

- class Client_Acceptor

Typedefs

- typedef ACE_Acceptor< Client_Handler, ACE_SOCK_ACCEPTOR > Client_Acceptor_Base

  The ACE Acceptor that is used in evalserver.

6.2.1 Typedef Documentation

6.2.1.1 typedef ACE_Acceptor<Client_Handler, ACE_SOCK_ACCEPTOR> Client_Acceptor_Base

Parameterize the ACE_Acceptor<> such that it will listen for socket connection attempts and create Client_Handler objects when they happen. In Tutorial 001, we wrote the basic acceptor logic on our own before we realized that ACE_Acceptor<> was available. You'll get spoiled using the ACE templates because they take away a lot of the tedious details!

Definition at line 47 of file client_acceptor.h.

6.3 client_handler.cpp File Reference

```
#include "client_acceptor.h"
#include "client_handler.h"
#include "contest.h"
```

Defines

- #define REGISTER_MASK ACE_Event_Handler::READ_MASK
- #define REMOVE_MASK (ACE_Event_Handler::READ_MASK | ACE_Event_Handler::DONT_CALL)

6.3.1 Define Documentation

6.3.1.1 #define REGISTER_MASK ACE_Event_Handler::READ_MASK

We're going to be registering and unregistering a couple of times. To make sure that we use the same flags every time, I've created these handy macros.

Definition at line 14 of file client_handler.cpp.

Referenced by Client_Handler::handle_input(), and Client_Handler::open().

6.3.1.2 #define REMOVE_MASK (ACE_Event_Handler::READ_MASK | ACE_Event_Handler::DONT_CALL)

Definition at line 15 of file client_handler.cpp.

Referenced by Client_Handler::destroy(), and Client_Handler::handle_input().

6.4 client_handler.h File Reference

```
#include "ace/Svc_Handler.h"
#include "ace/SOCK_Stream.h"
```

Data Structures

- class Client_Handler

6.5 config.h File Reference

Defines

- #define PACKAGE ”evalserver”
- #define VERSION ”1.1.0”
- #define HAVE_TEMPLATE_REPOSITORY 1

6.5.1 Define Documentation

6.5.1.1 #define HAVE_TEMPLATE_REPOSITORY 1

Definition at line 11 of file config.h.

6.5.1.2 #define PACKAGE ”evalserver”

Definition at line 5 of file config.h.

6.5.1.3 #define VERSION ”1.1.0”

Definition at line 8 of file config.h.

6.6 connectionmsgblock.cpp File Reference

```
#include "connectionmsgblock.h"
```

6.7 connectionmsgblock.h File Reference

```
#include <vector>
#include <iostream>
#include "partdetails.h"
#include "crc_32.h"
```

Data Structures

- class connectionMsgBlock

  A wrapper class for the message block that holds participation information.

6.8 contest.cpp File Reference

```
#include "contest.h"
```

6.9 contest.h File Reference

```
#include <ace/Reactor.h>
#include <ace/Timer_Queue.h>
#include <ace/Thread.h>
#include <ace/Synch.h>
#include <ace/Auto_Ptr.h>
#include <vector>
#include <set>
#include <algorithm>
#include "Count.h"
#include "client_acceptor.h"
#include "connectionmsgblock.h"
#include "day.h"
#include "giftdetails.h"
#include "logger.h"
#include "processor.h"
```

```
#include "sqliteconnection.h"
#include "sqltable.h"
```

Data Structures

- class Contest

Defines

- #define DBNAME ”competition.db”

  The filename of the SQLite database.

- #define LOGFILENAME ”evalserver.log”

  The filename of the logfile.

6.9.1 Define Documentation

6.9.1.1 #define DBNAME ”competition.db”

Author:
Konstantinos Margaritis

Definition at line 51 of file contest.h.

Referenced by Contest::initDB().

6.9.1.2 #define LOGFILENAME ”evalserver.log”

Definition at line 54 of file contest.h.

Referenced by Contest::initLogger().

6.10 Count.h File Reference

```
#include <ace/Version.h>
#include <ace/Synch_T.h>
#include <string>
#include <sstream>
```

Data Structures

- class Count

Typedefs

- typedef ACE_Atomic_Op< ACE_Mutex, unsigned int > counter_t

6.10.1 Typedef Documentation

6.10.1.1 typedef ACE_Atomic_Op<ACE_Mutex, unsigned int> counter_t

Definition at line 36 of file Count.h.

Referenced by Count::Count().

6.11 crc_32.cpp File Reference

```
#include "crc_32.h"
```

6.12 crc_32.h File Reference

```
#include <sys/types.h>
#include <iostream>
```

Data Structures

- class CRC_32

  Class to return CRC-32 error detection.

6.13 day.cpp File Reference

```
#include "day.h"
```

6.14 day.h File Reference

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <sys/stat.h>
#include <cmath>
#include <ace/Synch.h>
#include "contest.h"
#include "timeperiod.h"
#include "Count.h"
#include "strtokenizer.h"
```

Data Structures

- class Day

This class holds info about a day in the competition.

6.15 giftdetails.cpp File Reference

```
#include "giftdetails.h"
#include "strtokenizer.h"
```

6.16 giftdetails.h File Reference

```
#include <map>
#include <vector>
#include <string>
#include <algorithm>
#include "timeperiod.h"
```

Data Structures

- class giftDetails

  Wrapper class to deal with the prizes.

6.17 logger.cpp File Reference

```
#include "logger.h"
#include "logmsg_mo.h"
```

6.18 logger.h File Reference

```
#include <ace/Synch.h>
#include <ace/Task.h>
#include <ace/Future.h>
#include <ace/Activation_Queue.h>
#include <ace/Method_Object.h>
#include <string>
#include <memory>
#include <fstream>
#include <iomanip>
#include "timestamp.h"
```

Data Structures

- class Logger

    Log a message to stdout and/or to a file.

Enumerations

- enum output_t { OUTPUT_STDOUT, OUTPUT_BOTH, OUTPUT_FILEONLY }

    Enum to specify the type of the output method.

6.18.1 Enumeration Type Documentation

6.18.1.1 enum output_t

Enumeration values:

OUTPUT_STDOUT

OUTPUT_BOTH

OUTPUT_FILEONLY

Definition at line 42 of file logger.h.

Referenced by Logger::Logger().

6.19 logmsg_mo.cpp File Reference

```
#include "logmsg_mo.h"
```

6.20 logmsg_mo.h File Reference

```
#include <ace/Method_Object.h>
#include "logger.h"
```

Data Structures

- class logMsg_MO

    This class is the method object that is queued by the Logger object.

6.21 main.cpp File Reference

```
#include <iostream>
#include <stdlib.h>
#include "contest.h"
```

Functions

- int main (int argc, char ∗argv[ ])

6.21.1 Function Documentation

6.21.1.1 int main (int *argc*, char ∗ *argv*[ ])

Definition at line 21 of file main.cpp.

References Contest::start().

6.22 partdetails.cpp File Reference

```
#include "partdetails.h"
#include "strtokenizer.h"
```

Functions

- ostream & operator<< (ostream &out, partDetails &pd)

6.22.1 Function Documentation

6.22.1.1 ostream& operator<< (ostream & *out*, partDetails & *pd*)

We overload the operator<< to allow a partDetails object to be output to a C++ stream. The output will be of the form: MSISDN\tGID\tDATESTRING

Definition at line 68 of file partdetails.cpp.

References partDetails::getGiftId(), partDetails::getMSISDN(), partDetails::getTimestamp(), and TimeStamp::toString().

6.23 partdetails.h File Reference

```
#include <vector>
#include <string>
#include <iterator>
#include <stdlib.h>
#include <stdio.h>
#include "timestamp.h"
#include "giftdetails.h"
```

Data Structures

- class partDetails

  Wrapper class to deal with the participations.

6.24 processor.cpp File Reference

```
#include "processor.h"
#include "processor_mo.h"
#include "contest.h"
```

6.25 processor.h File Reference

```
#include <ace/Synch.h>
#include <ace/Task.h>
#include <ace/Future.h>
#include <ace/Activation_Queue.h>
#include <ace/Method_Object.h>
#include "connectionmsgblock.h"
```

Data Structures

- class Processor

  Process the connectionMsgBlock messages.

6.26 processor_mo.cpp File Reference

```
#include "processor_mo.h"
#include "contest.h"
```

6.27 processor_mo.h File Reference

```
#include <ace/Method_Object.h>
#include "processor.h"
#include "connectionmsgblock.h"
```

Data Structures

- class processor_MO

  This class is the method object that is queued by the Processor object.

6.28 sqliteconnection.cpp File Reference

```
#include "sqliteconnection.h"
```

6.29 sqliteconnection.h File Reference

```
#include <ace/Synch.h>
#include <ace/Task.h>
#include <ace/Future.h>
#include <ace/Activation_Queue.h>
#include <ace/Method_Object.h>
#include <sqlite.h>
#include <sstream>
#include <map>
#include <vector>
#include <algorithm>
#include "logger.h"
#include "timestamp.h"
```

Data Structures

- class SQLiteConnection

  Provide a C++ class for the C API of SQLite.

Defines

- #define TRANSLOGPATH ”TransactionLogs/”
- #define TRANSLOGSUFFIX ”.sql”

6.29.1 Define Documentation

6.29.1.1 #define TRANSLOGPATH ”TransactionLogs/”

Definition at line 49 of file sqliteconnection.h.

Referenced by SQLiteConnection::pickFilename().

6.29.1.2 #define TRANSLOGSUFFIX ”.sql”

Definition at line 50 of file sqliteconnection.h.

Referenced by SQLiteConnection::pickFilename().

6.30 sqltable.cpp File Reference

```
#include "sqltable.h"
```

6.31 sqltable.h File Reference

```
#include <sstream>
#include <vector>
#include <map>
#include <algorithm>
#include <ace/OS.h>
#include "sqliteconnection.h"
#include "timeperiod.h"
```

Data Structures

- class SQLTable

  Template class to provide an easy way to access an SQL table.

6.32 strtokenizer.cpp File Reference

```
#include "strtokenizer.h"
```

Functions

- bool strTokenizer (string &str, vector< string > &splitted, const string delimiter)

  Separates a given string using the given delimiters and outputs the result to a vector.

6.32.1 Function Documentation

6.32.1.1 bool strTokenizer (string & *str*, vector< string > & *splitted*, const string *delimiter*)

This helper function, takes a given string (str), separates it using the given delimiters, and outputs the result to the given vector<string> (splitted).

Definition at line 16 of file strtokenizer.cpp.

Referenced by Day::Day(), giftDetails::giftDetails(), and partDetails::partDetails().

6.33 strtokenizer.h File Reference

```
#include <string>
#include <vector>
```

Functions

- bool strTokenizer (string &str, vector< string > &splitted, const string delimiter)

  Separates a given string using the given delimiters and outputs the result to a vector.

6.33.1 Function Documentation

6.33.1.1 bool strTokenizer (string & *str*, vector< string > & *splitted*, const string *delimiter*)

This helper function, takes a given string (str), separates it using the given delimiters, and outputs the result to the given vector<string> (splitted).

Definition at line 16 of file strtokenizer.cpp.

Referenced by Day::Day(), giftDetails::giftDetails(), and partDetails::partDetails().

6.34 thread_pool.cpp File Reference

```
#include "thread_pool.h"
#include "ace/Event_Handler.h"
```

Data Structures

- class Counter_Guard
- class Message_Block_Guard

6.35 thread_pool.h File Reference

```
#include "ace/Task.h"
#include <ace/Version.h>
#include <ace/Synch_T.h>
```

Data Structures

- class Thread_Pool

  Provides an independant mechanism for a Thread Pool.

6.36 timeperiod.cpp File Reference

```
#include "timeperiod.h"
```

6.37 timeperiod.h File Reference

```
#include "timestamp.h"
```

Data Structures

- class TimePeriod

  Class to handle timeperiods consistently.

6.38 timestamp.cpp File Reference

```
#include "timestamp.h"
```

6.39 timestamp.h File Reference

```
#include <string>
#include <sstream>
#include <iomanip>
#include <ctime>
```

Data Structures

- class TimeStamp

  TimeStamp manipulation and handling class.