faketicket

Generated by Doxygen 1.9.3

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

2 Hierarchical Index

2.1 Class Hierarchy

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

3 Class Index

3.1 Class List

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

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Namespace Documentation

5.1 ticket Namespace Reference

Namespaces

- namespace command

  Classes and parsers for commands.
- namespace file

  File utilities.
- namespace response
- namespace rollback
- namespace Station

Classes

- struct BuyTicketEnqueued

  Utility class to represent the result of a buy ticket request that a pending order has been created.
- struct BuyTicketSuccess

  Utility class to represent the result of a buy ticket request that the order has been processed.
- class Cmp

  Comparison utilities.
- class Date

  Class representing a date between 2021-06-01 and 2021-08-31 (inclusive).
- class Duration

  Class representing a length of timespan.
- class Exception

  The base exception class.
- class HashMap

  An unordered hash-based map.
- class Instant

  Class representing a point of time in a day.

- class IoException
- class LruCache

  A fixed-size cache with a least recently used policy.
- class Map

  A sorted key-value map backed by a red-black tree.
- class NotFound
- class Optional

  A resemblence of std::optional.
- struct Order
- struct OrderBase
- struct OrderCache
- class OutOfBounds
- class Overflow
- class Pair

  A pair of objects.
- class ParseException
- class Result

  Result<Res, Err> = Res | Err.
- struct Ride
- struct RideSeats
- struct RideSeatsBase
- struct Train
- struct TrainBase
- class Triple

  A triplet of objects.
- class Underflow
- struct Unit

  An empty class, used at various places.
- struct User
- struct UserBase
- class Variant

  A tagged union, aka sum type.
- class Vector

  A data container like std::vector.

Typedefs

- using BuyTicketResponse = Variant< BuyTicketSuccess, BuyTicketEnqueued >
- using Response = Variant< Unit, User, Train, Vector< Train >, BuyTicketResponse, Vector< Order > >
- template<typename Lt = internal::LessOp>
  using Less = Cmp< Lt >
- template<typename Lt = internal::LessOp>
  using Greater = Cmp< internal::GreaterOp< Lt > >

Functions

- auto setTimestamp (int timestamp) -> void

  sets the current timestamp.
- auto makeUser (const command::AddUser &cmd) -> User
- template<typename Cmd >
  auto checkUser (const Cmd &cmd) -> Result< Pair< User, User >, Exception >
- template<typename Iterator , class Compare = Less<>>
  auto sort (Iterator first, Iterator last, Compare cmp={}) -> void

  sorts the elements between first and last.
- auto formatDateTime (Date date, Instant instant) -> std::string
- auto split (std::string &str, char sep) -> Vector< std::string_view >

  splits the string with sep into several substrings.
- auto copyStrings (const Vector< std::string_view > &vec) -> Vector< std::string >

  copies the strings in vec into an array of real strings.
- template<typename T >
  auto declval () -> T

  declare value, used in type annotations.
- template<typename T >
  auto move (T &val) -> T &&

  forcefully make an rvalue.
- auto isVisibleChar (char ch) -> bool

Variables

- HashMap< std::string, Unit > usersLoggedIn

  a set of users that are logged in.
- constexpr Unit unit

5.1.1 Detailed Description

This file defines exception classes used throughout the project. Throwing exceptions is not encouraged, since it has a poor stack unwinding performance.

5.1.2 Typedef Documentation

5.1.2.1 BuyTicketResponse `using ticket::BuyTicketResponse = typedef Variant< BuyTicketSuccess, BuyTicketEnqueued >`

5.1.2.2 Greater `template<typename Lt = internal::LessOp>`
`using ticket::Greater = typedef Cmp<internal::GreaterOp<Lt> >`

5.1.2.3 Less

```
template<typename Lt = internal::LessOp>
using ticket::Less = typedef Cmp<Lt>
```

5.1.2.4 Response

```
using ticket::Response = typedef Variant< Unit, User, Train, Vector<Train>,
BuyTicketResponse, Vector<Order> >
```

5.1.3 Function Documentation

5.1.3.1 checkUser()

```
template<typename Cmd >
auto ticket::checkUser (
            const Cmd & cmd ) -> Result<Pair<User, User>, Exception>   [inline]
```

5.1.3.2 copyStrings()

```
auto ticket::copyStrings (
            const Vector< std::string_view > & vec ) -> Vector< std::string >
```

copies the strings in vec into an array of real strings.

5.1.3.3 declval()

```
template<typename T >
auto ticket::declval ( ) -> T
```

declare value, used in type annotations.

5.1.3.4 formatDateTime()

```
auto ticket::formatDateTime (
            Date date,
            Instant instant ) -> std::string
```

5.1.3.5 isVisibleChar()

```
auto ticket::isVisibleChar (
            char ch ) -> bool   [inline]
```

5.1.3.6 makeUser()

```
auto ticket::makeUser (
            const command::AddUser & cmd ) -> User   [inline]
```

5.1.3.7 move() `template<typename T >`

```
auto ticket::move (
            T & val ) -> T &&
```

forcefully make an rvalue.

5.1.3.8 setTimestamp() `auto ticket::setTimestamp (`

```
            int timestamp ) -> void
```

sets the current timestamp.

5.1.3.9 sort() `template<typename Iterator , class Compare = Less<>>`

```
auto ticket::sort (
            Iterator first,
            Iterator last,
            Compare cmp = {} ) -> void
```

sorts the elements between first and last.

5.1.3.10 split() `auto ticket::split (`

```
            std::string & str,
            char sep ) -> Vector< std::string_view >
```

splits the string with sep into several substrings.

this function mutates the incoming string to make sure the result is properly zero-terminated.

the lifetime of the return value is the lifetime of the incoming string; that is to say, you need to keep the original string from destructured in order to use the result.

5.1.4 Variable Documentation

5.1.4.1 unit `constexpr Unit ticket::unit [inline], [constexpr]`

5.1.4.2 usersLoggedIn `HashMap<std::string, Unit> ticket::usersLoggedIn`

a set of users that are logged in.

5.2 ticket::command Namespace Reference

Classes and parsers for commands.

Classes

- struct AddTrain
- struct AddUser
- struct BuyTicket
- struct Clean
- struct DeleteTrain
- struct Exit
- struct Login
- struct Logout
- struct ModifyProfile
- struct QueryOrder
- struct QueryProfile
- struct QueryTicket
- struct QueryTrain
- struct QueryTransfer
- struct RefundTicket
- struct ReleaseTrain
- struct Rollback

Typedefs

- using Command = Variant< AddUser, Login, Logout, QueryProfile, ModifyProfile, AddTrain, DeleteTrain, ReleaseTrain, QueryTrain, QueryTicket, QueryTransfer, BuyTicket, QueryOrder, RefundTicket, Rollback, Clean, Exit >

Enumerations

- enum SortType { kTime , kCost }

Functions

- auto parse (std::string &str) -> Result< Command, ParseException >

  parses the command stored in str.
- auto parse (const Vector< std::string_view > &argv) -> Result< Command, ParseException >
- auto run (const AddUser &cmd) -> Result< Response, Exception >

  Visitor for the commands.
- auto run (const Login &cmd) -> Result< Response, Exception >
- auto run (const Logout &cmd) -> Result< Response, Exception >
- auto run (const QueryProfile &cmd) -> Result< Response, Exception >
- auto run (const ModifyProfile &cmd) -> Result< Response, Exception >
- auto run (const AddTrain &cmd) -> Result< Response, Exception >
- auto run (const DeleteTrain &cmd) -> Result< Response, Exception >
- auto run (const ReleaseTrain &cmd) -> Result< Response, Exception >
- auto run (const QueryTrain &cmd) -> Result< Response, Exception >
- auto run (const QueryTicket &cmd) -> Result< Response, Exception >
- auto run (const QueryTransfer &cmd) -> Result< Response, Exception >
- auto run (const BuyTicket &cmd) -> Result< Response, Exception >
- auto run (const QueryOrder &cmd) -> Result< Response, Exception >
- auto run (const RefundTicket &cmd) -> Result< Response, Exception >
- auto run (const Rollback &cmd) -> Result< Response, Exception >
- auto run (const Clean &cmd) -> Result< Response, Exception >
- auto run (const Exit &cmd) -> Result< Response, Exception >

5.2.1 Detailed Description

Classes and parsers for commands.

5.2.2 Typedef Documentation

5.2.2.1 Command

```
using ticket::command::Command = typedef Variant< AddUser, Login, Logout,
QueryProfile, ModifyProfile, AddTrain, DeleteTrain, ReleaseTrain, QueryTrain, QueryTicket,
QueryTransfer, BuyTicket, QueryOrder, RefundTicket, Rollback, Clean, Exit >
```

5.2.3 Enumeration Type Documentation

5.2.3.1 SortType

```
enum ticket::command::SortType
```

Enumerator

kTime	
kCost	

5.2.4 Function Documentation

5.2.4.1 parse() [1/2]

```
auto ticket::command::parse (
        const Vector< std::string_view > & argv ) -> Result< Command, ParseException >
```

5.2.4.2 parse() [2/2]

```
auto ticket::command::parse (
        std::string & str ) -> Result< Command, ParseException >
```

parses the command stored in str.

this function is autogenerated.

5.2.4.3 run() [1/17]

```
auto ticket::command::run (
        const AddTrain & cmd ) -> Result<Response, Exception>
```

5.2.4.4 run() [2/17]

```
auto ticket::command::run (
    const AddUser & cmd ) -> Result<Response, Exception>
```

Visitor for the commands.

The main function uses this visitor after parsing a command, to actually dispatch it. Overloads of operator() are callbacks of the commands.

The implementations are in the corresponding source files, not in parser.cpp.

5.2.4.5 run() [3/17]

```
auto ticket::command::run (
    const BuyTicket & cmd ) -> Result<Response, Exception>
```

5.2.4.6 run() [4/17]

```
auto ticket::command::run (
    const Clean & cmd ) -> Result<Response, Exception>
```

5.2.4.7 run() [5/17]

```
auto ticket::command::run (
    const DeleteTrain & cmd ) -> Result<Response, Exception>
```

5.2.4.8 run() [6/17]

```
auto ticket::command::run (
    const Exit & cmd ) -> Result<Response, Exception>
```

5.2.4.9 run() [7/17]

```
auto ticket::command::run (
    const Login & cmd ) -> Result<Response, Exception>
```

5.2.4.10 run() [8/17]

```
auto ticket::command::run (
    const Logout & cmd ) -> Result<Response, Exception>
```

5.2.4.11 run() [9/17]

```
auto ticket::command::run (
    const ModifyProfile & cmd ) -> Result<Response, Exception>
```

5.2.4.12 run() [10/17]

```
auto ticket::command::run (
    const QueryOrder & cmd ) -> Result<Response, Exception>
```

5.2.4.13 run() [11/17] auto ticket::command::run (

```
const QueryProfile & cmd ) -> Result<Response, Exception>
```

5.2.4.14 run() [12/17] auto ticket::command::run (

```
const QueryTicket & cmd ) -> Result<Response, Exception>
```

5.2.4.15 run() [13/17] auto ticket::command::run (

```
const QueryTrain & cmd ) -> Result<Response, Exception>
```

5.2.4.16 run() [14/17] auto ticket::command::run (

```
const QueryTransfer & cmd ) -> Result<Response, Exception>
```

5.2.4.17 run() [15/17] auto ticket::command::run (

```
const RefundTicket & cmd ) -> Result<Response, Exception>
```

5.2.4.18 run() [16/17] auto ticket::command::run (

```
const ReleaseTrain & cmd ) -> Result<Response, Exception>
```

5.2.4.19 run() [17/17] auto ticket::command::run (

```
const Rollback & cmd ) -> Result<Response, Exception>
```

5.3 ticket::file Namespace Reference

File utilities.

Classes

- struct Array

  An on-stack array with utility functions and bound checks.
- class BpTree

  an implementation of the B+ tree.
- class File

  A chunked file storage with manual garbage collection.
- class Index

  Class representing an index file.
- class Index< Varchar< maxLength >, Model >

  Specialization of Index on Varchar.
- class Managed

  an opinionated utility class wrapper for the objects to be stored.
- struct Set

  A sorted array with utility functions and bound checks.
- struct Varchar

  *A wrapper for const char * with utility functions and type conversions.*

Variables

- constexpr size_t kDefaultSzChunk = 4096

5.3.1 Detailed Description

File utilities.

5.3.2 Variable Documentation

5.3.2.1 kDefaultSzChunk `constexpr size_t ticket::file::kDefaultSzChunk = 4096 [constexpr]`

5.4 ticket::response Namespace Reference

Functions

- auto cout (const Unit &) -> void
- auto cout (const User &user) -> void
- auto cout (const Train &train) -> void
- auto cout (const Vector< Train > &trains) -> void
- auto cout (const BuyTicketResponse &ticket) -> void
- auto cout (const Vector< Order > &orders) -> void

5.4.1 Function Documentation

5.4.1.1 cout() [1/6]

```
auto ticket::response::cout (
        const BuyTicketResponse & ticket ) -> void
```

5.4.1.2 cout() [2/6]

```
auto ticket::response::cout (
        const Train & train ) -> void
```

5.4.1.3 cout() [3/6]

```
auto ticket::response::cout (
        const Unit &  ) -> void
```

5.4.1.4 cout() [4/6] auto ticket::response::cout (const User & *user*) -> void

5.4.1.5 cout() [5/6] auto ticket::response::cout (const Vector< Order > & *orders*) -> void

5.4.1.6 cout() [6/6] auto ticket::response::cout (const Vector< Train > & *trains*) -> void

5.5 ticket::rollback Namespace Reference

Classes

- struct AddTrain
- struct AddUser
- struct BuyTicket
- struct DeleteTrain
- struct FulfillOrder
- struct LogEntryBase
- struct ModifyProfile
- struct RefundTicket
- struct ReleaseTrain

Typedefs

- using LogEntry = file::Managed< LogEntryBase >

Functions

- auto log (const LogEntry::Content &content) -> void

  inserts a log entry.
- auto run (const AddUser &log) -> Result< Unit, Exception >

  Visitor for the log entries.
- auto run (const ModifyProfile &log) -> Result< Unit, Exception >
- auto run (const AddTrain &log) -> Result< Unit, Exception >
- auto run (const DeleteTrain &log) -> Result< Unit, Exception >
- auto run (const ReleaseTrain &log) -> Result< Unit, Exception >
- auto run (const BuyTicket &log) -> Result< Unit, Exception >
- auto run (const RefundTicket &log) -> Result< Unit, Exception >
- auto run (const FulfillOrder &log) -> Result< Unit, Exception >

5.5.1 Typedef Documentation

5.5.1.1 LogEntry

```
using ticket::rollback::LogEntry = typedef file::Managed<LogEntryBase>
```

5.5.2 Function Documentation

5.5.2.1 log()

```
auto ticket::rollback::log (
    const LogEntry::Content & content ) -> void
```

inserts a log entry.

5.5.2.2 run() [1/8]

```
auto ticket::rollback::run (
    const AddTrain & log ) -> Result<Unit, Exception>
```

5.5.2.3 run() [2/8]

```
auto ticket::rollback::run (
    const AddUser & log ) -> Result<Unit, Exception>
```

Visitor for the log entries.

The implementations are in the corresponding source files, not in rollback.cpp.

5.5.2.4 run() [3/8]

```
auto ticket::rollback::run (
    const BuyTicket & log ) -> Result<Unit, Exception>
```

5.5.2.5 run() [4/8]

```
auto ticket::rollback::run (
    const DeleteTrain & log ) -> Result<Unit, Exception>
```

5.5.2.6 run() [5/8]

```
auto ticket::rollback::run (
    const FulfillOrder & log ) -> Result<Unit, Exception>
```

5.5.2.7 run() [6/8]

```
auto ticket::rollback::run (
    const ModifyProfile & log ) -> Result<Unit, Exception>
```

5.5.2.8 run() [7/8]

```
auto ticket::rollback::run (
        const RefundTicket & log ) -> Result<Unit, Exception>
```

5.5.2.9 run() [8/8]

```
auto ticket::rollback::run (
        const ReleaseTrain & log ) -> Result<Unit, Exception>
```

5.6 ticket::Station Namespace Reference

Typedefs

- using Id = file::Varchar< 30 >

5.6.1 Typedef Documentation

5.6.1.1 Id

```
using ticket::Station::Id = typedef file::Varchar<30>
```

6 Class Documentation

6.1 ticket::command::AddTrain Struct Reference

```
#include <parser.h>
```

Public Attributes

- std::string id
- int stops
- int seats
- Vector< std::string > stations
- Vector< int > prices
- Instant departure
- Vector< Duration > durations
- Vector< Duration > stopoverTimes
- Vector< Date > dates
- char type

6.1.1 Member Data Documentation

6.1.1.1 dates

`Vector<Date> ticket::command::AddTrain::dates`

6.1.1.2 departure

`Instant ticket::command::AddTrain::departure`

6.1.1.3 durations

`Vector<Duration> ticket::command::AddTrain::durations`

6.1.1.4 id

`std::string ticket::command::AddTrain::id`

6.1.1.5 prices

`Vector<int> ticket::command::AddTrain::prices`

6.1.1.6 seats

`int ticket::command::AddTrain::seats`

6.1.1.7 stations

`Vector<std::string> ticket::command::AddTrain::stations`

6.1.1.8 stopoverTimes

`Vector<Duration> ticket::command::AddTrain::stopoverTimes`

6.1.1.9 stops

`int ticket::command::AddTrain::stops`

6.1.1.10 type

`char ticket::command::AddTrain::type`

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

- src/parser.h

6.2 ticket::rollback::AddTrain Struct Reference

`#include <rollback.h>`

Public Attributes

- int id

6.2.1 Member Data Documentation

6.2.1.1 id `int ticket::rollback::AddTrain::id`

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

- src/rollback.h

6.3 ticket::command::AddUser Struct Reference

`#include <parser.h>`

Public Attributes

- Optional< std::string > currentUser
- std::string username
- std::string password
- std::string name
- std::string email
- Optional< int > privilege

6.3.1 Member Data Documentation

6.3.1.1 currentUser `Optional<std::string> ticket::command::AddUser::currentUser`

6.3.1.2 email `std::string ticket::command::AddUser::email`

6.3.1.3 name `std::string ticket::command::AddUser::name`

6.3.1.4 password `std::string ticket::command::AddUser::password`

6.3.1.5 privilege `Optional<int> ticket::command::AddUser::privilege`

6.3.1.6 username `std::string ticket::command::AddUser::username`

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

- src/parser.h

6.4 ticket::rollback::AddUser Struct Reference

`#include <rollback.h>`

Public Attributes

- int id

6.4.1 Member Data Documentation

6.4.1.1 id `int ticket::rollback::AddUser::id`

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

- src/rollback.h

6.5 ticket::file::Array< T, maxLength, Cmp > Struct Template Reference

An on-stack array with utility functions and bound checks.

`#include <array.h>`

Public Member Functions

- auto indexOf (const T &element) -> size_t
  finds the index of element in the array.
- auto includes (const T &element) -> bool
  checks if the elements is included in the array.
- auto insert (const T &element, size_t offset) -> void
  moves the elements after offset backwards, and inserts the element at the offset.
- auto remove (const T &element) -> void
  removes the element, and moves forward the elements after it.
- auto removeAt (size_t offset) -> void
  removes the element at offset, and moves forward the elements after it.
- auto clear () -> void
  clears the array.
- auto copyFrom (const Array &other, size_t ixFrom, size_t ixTo, size_t count) -> void
  copies a portion of another array to this.
- auto operator[ ] (size_t index) -> T &
- auto operator[ ] (size_t index) const -> const T &
- auto pop () -> T
  pops the last element.
- auto shift () -> T
  pops the first element.
- auto push (const T &object) -> void
  pushes after the last element.
- auto unshift (const T &object) -> void
  pushes before the first element.
- template<typename Functor >
  auto forEach (const Functor &callback) -> T
  calls the callback for each element in the array.

Public Attributes

- size_t length = 0
- T content [maxLength]

6.5.1 Detailed Description

template<typename T, size_t maxLength, typename Cmp = Less<>>
struct ticket::file::Array< T, maxLength, Cmp >

An on-stack array with utility functions and bound checks.

The value type needs to be trivial.

6.5.2 Member Function Documentation

6.5.2.1 clear()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::clear ( ) -> void   [inline]
```

clears the array.

6.5.2.2 copyFrom()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::copyFrom (
            const Array< T, maxLength, Cmp > & other,
            size_t ixFrom,
            size_t ixTo,
            size_t count ) -> void   [inline]
```

copies a portion of another array to this.

6.5.2.3 forEach()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
template<typename Functor >
auto ticket::file::Array< T, maxLength, Cmp >::forEach (
            const Functor & callback ) -> T   [inline]
```

calls the callback for each element in the array.

6.5.2.4 includes()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::includes (
            const T & element ) -> bool   [inline]
```

checks if the elements is included in the array.

6.5.2.5 indexOf()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::indexOf (
            const T & element ) -> size_t   [inline]
```

finds the index of element in the array.

6.5.2.6 insert()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::insert (
            const T & element,
            size_t offset ) -> void   [inline]
```

moves the elements after offset backwards, and inserts the element at the offset.

6.5.2.7 operator[]() [1/2]

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::operator[] (
            size_t index ) -> T &   [inline]
```

6.5.2.8 operator[]() [2/2]

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::operator[] (
            size_t index ) const -> const T &   [inline]
```

6.5.2.9 pop()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::pop ( ) -> T   [inline]
```

pops the last element.

6.5.2.10 push()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::push (
            const T & object ) -> void   [inline]
```

pushes after the last element.

6.5.2.11 remove()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::remove (
            const T & element ) -> void   [inline]
```

removes the element, and moves forward the elements after it.

6.5.2.12 removeAt()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::removeAt (
            size_t offset ) -> void   [inline]
```

removes the element at offset, and moves forward the elements after it.

6.5.2.13 shift()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Array< T, maxLength, Cmp >::shift ( ) -> T   [inline]
```

pops the first element.

6.5.2.14 unshift() template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Array< T, maxLength, Cmp >::unshift (
            const T & object ) -> void   [inline]
```

pushes before the first element.

6.5.3 Member Data Documentation

6.5.3.1 content template<typename T , size_t maxLength, typename Cmp = Less<>>

```
T ticket::file::Array< T, maxLength, Cmp >::content[maxLength]
```

6.5.3.2 length template<typename T , size_t maxLength, typename Cmp = Less<>>

```
size_t ticket::file::Array< T, maxLength, Cmp >::length = 0
```

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

- lib/file/array.h

6.6 ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk > Class Template Reference

an implementation of the B+ tree.

```
#include <bptree.h>
```

Public Member Functions

- BpTree (const char *filename)
  constructs a B+ tree on the given file.
- auto insert (const KeyType &key, const ValueType &value) -> void
  inserts a key-value pair into the tree.
- auto remove (const KeyType &key, const ValueType &value) -> void
  removes a key-value pair from the tree.
- auto findOne (const KeyType &key) -> Optional< ValueType >
  finds the first entry with the given key.
- auto findMany (const KeyType &key) -> Vector< ValueType >
  finds all entries with the given key.
- auto findAll () -> Vector< ticket::Pair< KeyType, ValueType > >
  finds all entries.
- auto includes (const KeyType &key, const ValueType &value) -> bool
  checks if the given key-value pair exists in the tree.
- auto empty () -> bool
  checks if the tree is empty.
- auto getMeta () -> Meta
  gets user-provided metadata.
- auto setMeta (const Meta &meta) -> void
  sets user-provided metadata.
- auto clearCache () -> void
  clears the cache of the underlying file.
- auto truncate () -> void
  hard deletes all entries in the tree.

6.6.1 Detailed Description

template<typename KeyType, typename ValueType, typename CmpKey = Less<>, typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
class ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >

an implementation of the B+ tree.

it stores key and value together in order to support duplicate keys.

constraints: KeyType and ValueType need to be comparable.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 BpTree()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::BpTree (
            const char * filename )  [inline]
```

constructs a B+ tree on the given file.

6.6.3 Member Function Documentation

6.6.3.1 clearCache()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::clearCache (
) -> void   [inline]
```

clears the cache of the underlying file.

you may need to call this method periodically to avoid using up too much memory.

6.6.3.2 empty()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::empty ( )
-> bool   [inline]
```

checks if the tree is empty.

6.6.3.3 findAll()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::findAll ( )
-> Vector<ticket::Pair<KeyType, ValueType>>   [inline]
```

finds all entries.

6.6.3.4 findMany()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::findMany (
            const KeyType & key ) -> Vector<ValueType>   [inline]
```

finds all entries with the given key.

6.6.3.5 findOne()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::findOne (
            const KeyType & key ) -> Optional<ValueType>   [inline]
```

finds the first entry with the given key.

6.6.3.6 getMeta()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::getMeta ( )
-> Meta   [inline]
```

gets user-provided metadata.

6.6.3.7 includes()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::includes (
            const KeyType & key,
            const ValueType & value ) -> bool   [inline]
```

checks if the given key-value pair exists in the tree.

6.6.3.8 insert()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::insert (
            const KeyType & key,
            const ValueType & value ) -> void   [inline]
```

inserts a key-value pair into the tree.

duplicate keys is supported, though duplicate key-value pair leads to undefined behavior, and may lead to an invalid tree.

6.6.3.9 remove()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::remove (
            const KeyType & key,
            const ValueType & value ) -> void   [inline]
```

removes a key-value pair from the tree.

you must ensure that the entry is indeed in the tree. removing an nonexistent entry may lead to an invalid tree.

6.6.3.10 setMeta()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::setMeta (
            const Meta & meta ) -> void   [inline]
```

sets user-provided metadata.

6.6.3.11 truncate()

```
template<typename KeyType , typename ValueType , typename CmpKey = Less<>,
typename CmpValue = Less<>, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >::truncate ( )
-> void   [inline]
```

hard deletes all entries in the tree.

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

- lib/file/bptree.h

6.7 ticket::command::BuyTicket Struct Reference

`#include <parser.h>`

Public Attributes

- std::string currentUser
- std::string train
- Date date
- int seats
- std::string from
- std::string to
- bool queue = false

6.7.1 Member Data Documentation

6.7.1.1 currentUser `std::string ticket::command::BuyTicket::currentUser`

6.7.1.2 date `Date ticket::command::BuyTicket::date`

6.7.1.3 from `std::string ticket::command::BuyTicket::from`

6.7.1.4 queue `bool ticket::command::BuyTicket::queue = false`

6.7.1.5 seats `int ticket::command::BuyTicket::seats`

6.7.1.6 to `std::string ticket::command::BuyTicket::to`

6.7.1.7 train `std::string ticket::command::BuyTicket::train`

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

- src/parser.h

6.8 ticket::rollback::BuyTicket Struct Reference

`#include <rollback.h>`

Public Attributes

- int id

6.8.1 Member Data Documentation

6.8.1.1 id `int ticket::rollback::BuyTicket::id`

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

- src/rollback.h

6.9 ticket::BuyTicketEnqueued Struct Reference

Utility class to represent the result of a buy ticket request that a pending order has been created.

`#include <order.h>`

6.9.1 Detailed Description

Utility class to represent the result of a buy ticket request that a pending order has been created.

See BuyTicketResponse below for usage.

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

- src/order.h

6.10 ticket::BuyTicketSuccess Struct Reference

Utility class to represent the result of a buy ticket request that the order has been processed.

`#include <order.h>`

Public Attributes

- int price

6.10.1 Detailed Description

Utility class to represent the result of a buy ticket request that the order has been processed.

See BuyTicketResponse below for usage.

6.10.2 Member Data Documentation

6.10.2.1 price `int ticket::BuyTicketSuccess::price`

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

- src/order.h

6.11 ticket::command::Clean Struct Reference

`#include <parser.h>`

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

- src/parser.h

6.12 ticket::Cmp< Lt > Class Template Reference

Comparison utilities.

`#include <utility.h>`

Public Member Functions

- template<typename T , typename U >
  auto equals (const T &lhs, const U &rhs) -> bool
- template<typename T , typename U >
  auto ne (const T &lhs, const U &rhs) -> bool
- template<typename T , typename U >
  auto lt (const T &lhs, const U &rhs) -> bool
- template<typename T , typename U >
  auto gt (const T &lhs, const U &rhs) -> bool
- template<typename T , typename U >
  auto leq (const T &lhs, const U &rhs) -> bool
- template<typename T , typename U >
  auto geq (const T &lhs, const U &rhs) -> bool

6.12.1 Detailed Description

template<typename Lt>
class ticket::Cmp< Lt >

Comparison utilities.

6.12.2 Member Function Documentation

6.12.2.1 equals() `template<typename Lt >`

```
template<typename T , typename U >
auto ticket::Cmp< Lt >::equals (
            const T & lhs,
            const U & rhs ) -> bool   [inline]
```

6.12.2.2 geq() template<typename Lt >

```
template<typename T , typename U >
auto ticket::Cmp< Lt >::geq (
            const T & lhs,
            const U & rhs ) -> bool   [inline]
```

6.12.2.3 gt() template<typename Lt >

```
template<typename T , typename U >
auto ticket::Cmp< Lt >::gt (
            const T & lhs,
            const U & rhs ) -> bool   [inline]
```

6.12.2.4 leq() template<typename Lt >

```
template<typename T , typename U >
auto ticket::Cmp< Lt >::leq (
            const T & lhs,
            const U & rhs ) -> bool   [inline]
```

6.12.2.5 lt() template<typename Lt >

```
template<typename T , typename U >
auto ticket::Cmp< Lt >::lt (
            const T & lhs,
            const U & rhs ) -> bool   [inline]
```

6.12.2.6 ne() template<typename Lt >

```
template<typename T , typename U >
auto ticket::Cmp< Lt >::ne (
            const T & lhs,
            const U & rhs ) -> bool   [inline]
```

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

- lib/utility.h

6.13 ticket::HashMap< Key, Value, Hash, Equal >::const_iterator Class Reference

`#include <hashmap.h>`

Public Types

- using difference_type = std::ptrdiff_t
- using value_type = const HashMap::value_type
- using pointer = value_type *
- using reference = value_type &
- using iterator_category = std::output_iterator_tag

Public Member Functions

- const_iterator ()=default
- const_iterator (const ListNode *node, const HashMap *home)
- const_iterator (const iterator &other)
- auto operator++ (int) -> const_iterator
- auto operator++ () -> const_iterator &
- auto operator-- (int) -> const_iterator
- auto operator-- () -> const_iterator &
- auto operator* () const -> reference
- auto operator== (const iterator &rhs) const -> bool
- auto operator== (const const_iterator &rhs) const -> bool
- auto operator!= (const iterator &rhs) const -> bool
- auto operator!= (const const_iterator &rhs) const -> bool
- auto operator-> () const noexcept -> pointer

Friends

- class iterator
- class HashMap

6.13.1 Member Typedef Documentation

6.13.1.1 difference_type

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::difference_type = std::ptrdiff_t
```

6.13.1.2 iterator_category

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::iterator_category = std::output_iterator_tag
```

6.13.1.3 pointer

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::pointer = value_type *
```

6.13.1.4 reference

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::reference = value_type &
```

6.13.1.5 value_type

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::value_type = const HashMap::value_type
```

6.13.2 Constructor & Destructor Documentation

6.13.2.1 const_iterator() [1/3]

```
template<typename Key , typename Value , typename Hash = std↩
::hash<Key>, typename Equal = std::equal_to<Key>>
ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::const_iterator ( )  [default]
```

6.13.2.2 const_iterator() [2/3]

```
template<typename Key , typename Value , typename Hash = std↩
::hash<Key>, typename Equal = std::equal_to<Key>>
ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::const_iterator (
            const ListNode * node,
            const HashMap * home )  [inline]
```

6.13.2.3 const_iterator() [3/3]

```
template<typename Key , typename Value , typename Hash = std↩
::hash<Key>, typename Equal = std::equal_to<Key>>
ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::const_iterator (
            const iterator & other )  [inline]
```

6.13.3 Member Function Documentation

6.13.3.1 operator"!=() [1/2]

```
template<typename Key , typename Value , typename Hash = std↩
::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator!= (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.13.3.2 operator"!=() [2/2]

```
template<typename Key , typename Value , typename Hash = std↩
::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator!= (
            const iterator & rhs ) const -> bool   [inline]
```

6.13.3.3 operator*()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator* ( ) const -> reference
[inline]
```

6.13.3.4 operator++() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator++ ( ) -> const_iterator
&   [inline]
```

6.13.3.5 operator++() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator++ (
            int  ) -> const_iterator   [inline]
```

6.13.3.6 operator--() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator-- ( ) -> const_iterator
&   [inline]
```

6.13.3.7 operator--() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator-- (
            int  ) -> const_iterator   [inline]
```

6.13.3.8 operator->()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator-> ( ) const ->
pointer   [inline], [noexcept]
```

6.13.3.9 operator==() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator== (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.13.3.10 operator==() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::const_iterator::operator== (
            const iterator & rhs ) const -> bool   [inline]
```

6.13.4 Friends And Related Function Documentation

6.13.4.1 HashMap

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
friend class HashMap  [friend]
```

6.13.4.2 iterator

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
friend class iterator  [friend]
```

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

- lib/hashmap.h

6.14 ticket::Vector< T >::const_iterator Class Reference

```
#include <vector.h>
```

Public Types

- using difference_type = std::ptrdiff_t
- using value_type = T
- using pointer = T *
- using reference = T &
- using iterator_category = std::random_access_iterator_tag

Public Member Functions

- auto operator+ (const int &n) const -> const_iterator
- auto operator- (const int &n) const -> const_iterator
- auto operator- (const const_iterator &rhs) const -> int
- auto operator+= (const int &n) -> const_iterator &
- auto operator-= (const int &n) -> const_iterator &
- auto operator++ (int) -> const_iterator
- auto operator++ () -> const_iterator &
- auto operator-- (int) -> const_iterator
- auto operator-- () -> const_iterator &
- auto operator* () const -> const T &
- auto operator== (const iterator &rhs) const -> bool
- auto operator== (const const_iterator &rhs) const -> bool
- auto operator!= (const iterator &rhs) const -> bool
- auto operator!= (const const_iterator &rhs) const -> bool
- auto operator< (const iterator &rhs) const -> bool
- auto operator< (const const_iterator &rhs) const -> bool

Friends

- class iterator
- class Vector

6.14.1 Member Typedef Documentation

6.14.1.1 difference_type template<typename T >

```
using ticket::Vector< T >::const_iterator::difference_type = std::ptrdiff_t
```

6.14.1.2 iterator_category template<typename T >

```
using ticket::Vector< T >::const_iterator::iterator_category = std::random_access_iterator_tag
```

6.14.1.3 pointer template<typename T >

```
using ticket::Vector< T >::const_iterator::pointer = T *
```

6.14.1.4 reference template<typename T >

```
using ticket::Vector< T >::const_iterator::reference = T &
```

6.14.1.5 value_type template<typename T >

```
using ticket::Vector< T >::const_iterator::value_type = T
```

6.14.2 Member Function Documentation

6.14.2.1 operator"!=() [1/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator!= (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.14.2.2 operator"!=() [2/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator!= (
            const iterator & rhs ) const -> bool   [inline]
```

6.14.2.3 operator*() template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator* ( ) const -> const T &   [inline]
```

6.14.2.4 operator+() template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator+ (
            const int & n ) const -> const_iterator   [inline]
```

6.14.2.5 operator++() [1/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator++ ( ) -> const_iterator &   [inline]
```

6.14.2.6 operator++() [2/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator++ (
            int  ) -> const_iterator   [inline]
```

6.14.2.7 operator+=() template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator+= (
            const int & n ) -> const_iterator &   [inline]
```

6.14.2.8 operator-() [1/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator- (
            const const_iterator & rhs ) const -> int   [inline]
```

6.14.2.9 operator-() [2/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator- (
            const int & n ) const -> const_iterator   [inline]
```

6.14.2.10 operator--() [1/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator-- ( ) -> const_iterator &   [inline]
```

6.14.2.11 operator--() [2/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator-- (
            int  ) -> const_iterator   [inline]
```

6.14.2.12 operator-=() template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator-= (
            const int & n ) -> const_iterator &   [inline]
```

6.14.2.13 operator<() [1/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator< (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.14.2.14 operator<() [2/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator< (
            const iterator & rhs ) const -> bool   [inline]
```

6.14.2.15 operator==() [1/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator== (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.14.2.16 operator==() [2/2] template<typename T >

```
auto ticket::Vector< T >::const_iterator::operator== (
            const iterator & rhs ) const -> bool   [inline]
```

6.14.3 Friends And Related Function Documentation

6.14.3.1 iterator template<typename T >

```
friend class iterator  [friend]
```

6.14.3.2 Vector template<typename T >

```
friend class Vector  [friend]
```

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

- lib/vector.h

6.15 ticket::Date Class Reference

Class representing a date between 2021-06-01 and 2021-08-31 (inclusive).

```
#include <datetime.h>
```

Public Member Functions

- Date ()=default
- Date (int month, int date)
- Date (const char *str)

  constructs a Date from a MM-DD format string.
- auto month () const -> int

  gets the month of the Date. (Fri Jun 04 2021 -> 6)
- auto date () const -> int

  gets the date of the Date. (Fri Jun 04 2021 -> 4)
- operator std::string () const

  gets a MM-DD representation of the Date.
- auto operator+ (int dt) const -> Date

  calculates a date dt days after this Date. (06-04 + 3 == 06-07)
- auto operator- (int dt) const -> Date

  calculates a date dt days before this Date. (06-04 - 3 == 06-01)
- auto operator- (Date rhs) const -> int

  calculates the difference between two Dates. (06-04 - 06-01 == 3)
- auto operator< (const Date &rhs) const -> bool
- auto inRange (Date begin, Date end) const -> bool

  checks if this Date is in the given range (inclusive).

6.15.1 Detailed Description

Class representing a date between 2021-06-01 and 2021-08-31 (inclusive).

6.15.2 Constructor & Destructor Documentation

6.15.2.1 Date() [1/3] ticket::Date::Date ( ) [default]

6.15.2.2 Date() [2/3] ticket::Date::Date (int *month*, int *date*)

6.15.2.3 Date() [3/3] ticket::Date::Date (const char * *str*) [explicit]

constructs a Date from a MM-DD format string.

it is an undefined behavior if the string is not in MM-DD format, is nullptr, or points to invalid memory.

6.15.3 Member Function Documentation

6.15.3.1 date() `auto ticket::Date::date ( ) const -> int`

gets the date of the Date. (Fri Jun 04 2021 -> 4)

6.15.3.2 inRange() `auto ticket::Date::inRange (`

```
Date begin,
Date end ) const -> bool
```

checks if this Date is in the given range (inclusive).

6.15.3.3 month() `auto ticket::Date::month ( ) const -> int`

gets the month of the Date. (Fri Jun 04 2021 -> 6)

6.15.3.4 operator std::string() `ticket::Date::operator std::string ( ) const`

gets a MM-DD representation of the Date.

6.15.3.5 operator+() `auto ticket::Date::operator+ (`

```
int dt ) const -> Date
```

calculates a date dt days after this Date. (06-04 + 3 == 06-07)

6.15.3.6 operator-() [1/2] `auto ticket::Date::operator- (`

```
Date rhs ) const -> int
```

calculates the difference between two Dates. (06-04 - 06-01 == 3)

6.15.3.7 operator-() [2/2] `auto ticket::Date::operator- (`

```
int dt ) const -> Date
```

calculates a date dt days before this Date. (06-04 - 3 == 06-01)

6.15.3.8 operator<() `auto ticket::Date::operator< ( const Date & rhs ) const -> bool`

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

- lib/datetime.h
- lib/datetime.cpp

6.16 ticket::command::DeleteTrain Struct Reference

`#include <parser.h>`

Public Attributes

- std::string id

6.16.1 Member Data Documentation

6.16.1.1 id `std::string ticket::command::DeleteTrain::id`

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

- src/parser.h

6.17 ticket::rollback::DeleteTrain Struct Reference

`#include <rollback.h>`

Public Attributes

- int id

6.17.1 Member Data Documentation

6.17.1.1 id `int ticket::rollback::DeleteTrain::id`

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

- src/rollback.h

6.18 ticket::Duration Class Reference

Class representing a length of timespan.

`#include <datetime.h>`

Public Member Functions

- Duration ()=default
- Duration (int minutes)
- auto minutes () const -> int

  gets how many minutes are there in this Duration.
- auto operator+ (Duration dt) const -> Duration
- auto operator- (Duration dt) const -> Duration
- auto operator- () const -> Duration

  negates the Duration.
- auto operator< (const Duration &rhs) const -> bool

6.18.1 Detailed Description

Class representing a length of timespan.

The length may be positive, zero or negative.

Not to be confused with Instant, which is a fixed point of time. For example, 02:10 as in "brewing the tea takes 02:10" is a duration, while 02:10 as in "it's 02:10 now, go to sleep right now" is an instant.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Duration() [1/2] ticket::Duration::Duration ( ) [default]

6.18.2.2 Duration() [2/2] ticket::Duration::Duration (int *minutes*) [inline], [explicit]

6.18.3 Member Function Documentation

6.18.3.1 minutes() auto ticket::Duration::minutes ( ) const -> int

gets how many minutes are there in this Duration.

6.18.3.2 operator+() `auto ticket::Duration::operator+ ( Duration dt ) const -> Duration`

6.18.3.3 operator-() [1/2] `auto ticket::Duration::operator- ( ) const -> Duration`

negates the Duration.

6.18.3.4 operator-() [2/2] `auto ticket::Duration::operator- ( Duration dt ) const -> Duration`

6.18.3.5 operator<() `auto ticket::Duration::operator< ( const Duration & rhs ) const -> bool`

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

- lib/datetime.h
- lib/datetime.cpp

6.19 ticket::TrainBase::Edge Struct Reference

`#include <train.h>`

Public Attributes

- int price
- Instant departure
- Instant arrival

6.19.1 Member Data Documentation

6.19.1.1 arrival `Instant ticket::TrainBase::Edge::arrival`

6.19.1.2 departure `Instant ticket::TrainBase::Edge::departure`

6.19.1.3 price `int ticket::TrainBase::Edge::price`

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

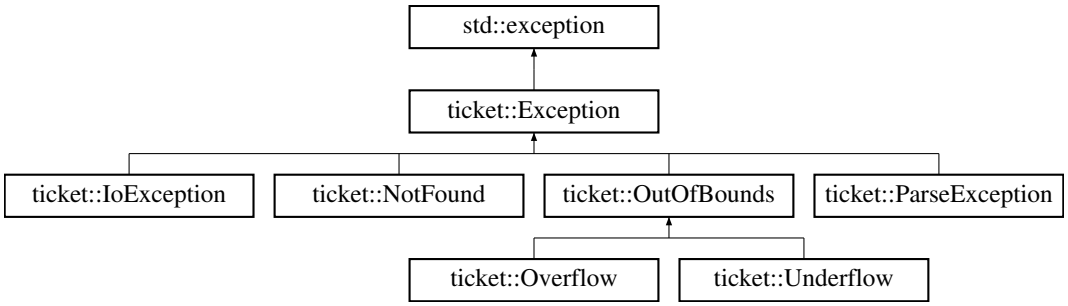
- src/train.h

6.20 ticket::Exception Class Reference

The base exception class.

`#include <exception.h>`

Inheritance diagram for ticket::Exception:



Public Member Functions

- Exception ()=default
- Exception (const char *what)
- ~Exception () override=default
- virtual auto what () const noexcept -> const char *

  returns a human-readable description of the exception.

6.20.1 Detailed Description

The base exception class.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 Exception() [1/2] `ticket::Exception::Exception ( ) [default]`

6.20.2.2 Exception() [2/2] `ticket::Exception::Exception ( const char * what ) [inline]`

6.20.2.3 ~Exception() `ticket::Exception::~Exception ( ) [override], [default]`

6.20.3 Member Function Documentation

6.20.3.1 what() `virtual auto ticket::Exception::what ( ) const -> const char * [inline], [virtual], [noexcept]`

returns a human-readable description of the exception.

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

- lib/exception.h

6.21 ticket::command::Exit Struct Reference

`#include <parser.h>`

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

- src/parser.h

6.22 ticket::file::File< Meta, szChunk > Class Template Reference

A chunked file storage with manual garbage collection.

`#include <file.h>`

Public Member Functions

- template<typename Functor >
  File (const char *filename, const Functor &initializer)
  initializes the file at filename.
- File (const char *filename)
- ~File ()
- auto get (void *buf, size_t index, size_t n) -> void
  read n bytes at index into buf.
- auto set (const void *buf, size_t index, size_t n) -> void
  write n bytes at index from buf.
- auto push (const void *buf, size_t n) -> size_t
- auto remove (size_t index) -> void
- auto getMeta () -> Meta
  gets user-provided metadata.
- auto setMeta (const Meta &user) -> void
  sets user-provided metadata.
- auto clearCache () -> void
  clears the cache.
- auto truncate () -> void
  clears file contents.

6.22.1 Detailed Description

template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
class ticket::file::File< Meta, szChunk >

A chunked file storage with manual garbage collection.

It is of chunk size of szChunk and has cache powered by HashMap.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 File() [1/2]

```
template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
template<typename Functor >
ticket::file::File< Meta, szChunk >::File (
            const char * filename,
            const Functor & initializer )  [inline]
```

initializes the file at filename.

it is not thread-safe.

Parameters

filename	the file to open
initializer	callback called on the creation of the file, when the file is empty.

6.22.2.2 File() [2/2]

```
template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
ticket::file::File< Meta, szChunk >::File (
            const char * filename )  [inline]
```

6.22.2.3 ~File()

```
template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
ticket::file::File< Meta, szChunk >::~File ( )  [inline]
```

6.22.3 Member Function Documentation

6.22.3.1 clearCache()

```
template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::File< Meta, szChunk >::clearCache ( ) -> void  [inline]
```

clears the cache.

6.22.3.2 get() template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>

```
auto ticket::file::File< Meta, szChunk >::get (
            void * buf,
            size_t index,
            size_t n ) -> void   [inline]
```

read n bytes at index into buf.

6.22.3.3 getMeta() template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>

```
auto ticket::file::File< Meta, szChunk >::getMeta ( ) -> Meta   [inline]
```

gets user-provided metadata.

6.22.3.4 push() template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>

```
auto ticket::file::File< Meta, szChunk >::push (
            const void * buf,
            size_t n ) -> size_t   [inline]
```

Returns

the stored index of the object

6.22.3.5 remove() template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>

```
auto ticket::file::File< Meta, szChunk >::remove (
            size_t index ) -> void   [inline]
```

6.22.3.6 set() template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>

```
auto ticket::file::File< Meta, szChunk >::set (
            const void * buf,
            size_t index,
            size_t n ) -> void   [inline]
```

write n bytes at index from buf.

6.22.3.7 setMeta() template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>

```
auto ticket::file::File< Meta, szChunk >::setMeta (
            const Meta & user ) -> void   [inline]
```

sets user-provided metadata.

6.22.3.8 truncate() `template<typename Meta = Unit, size_t szChunk = kDefaultSzChunk>`
`auto ticket::file::File< Meta, szChunk >::truncate ( ) -> void [inline]`

clears file contents.

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

- lib/file/file.h

6.23 ticket::rollback::FulfillOrder Struct Reference

`#include <rollback.h>`

Public Attributes

- int id

6.23.1 Member Data Documentation

6.23.1.1 id `int ticket::rollback::FulfillOrder::id`

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

- src/rollback.h

6.24 ticket::HashMap< Key, Value, Hash, Equal > Class Template Reference

An unordered hash-based map.

`#include <hashmap.h>`

Classes

- class const_iterator
- class iterator

Public Types

- using value_type = Pair< const Key, Value >

Public Member Functions

- HashMap ()=default
- HashMap (const HashMap &other)
- auto operator= (const HashMap &other) -> HashMap &
- ~HashMap ()
- auto at (const Key &key) -> Value &
- auto at (const Key &key) const -> const Value &
- auto operator[ ] (const Key &key) -> Value &
- auto operator[ ] (const Key &key) const -> const Value &

  behave like at() throw index_out_of_bound if such key does not exist.
- auto begin () -> iterator

  return a iterator to the beginning
- auto cbegin () const -> const_iterator
- auto end () -> iterator

  return a iterator to the end
- auto cend () const -> const_iterator
- auto empty () const -> bool

  checks whether the container is empty
- auto size () const -> size_t

  returns the number of elements.
- auto clear () -> void

  clears the contents
- auto insert (const value_type &value) -> Pair< iterator, bool >
- auto erase (iterator pos) -> void
- auto count (const Key &key) const -> size_t
- auto contains (const Key &key) const -> bool

  Checks if there is an element with key equivalent to key in the container.
- auto find (const Key &key) -> iterator
- auto find (const Key &key) const -> const_iterator

6.24.1 Detailed Description

template<typename Key, typename Value, typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
class ticket::HashMap< Key, Value, Hash, Equal >

An unordered hash-based map.

In HashMap, iteration ordering is differ from map, which is the order in which keys were inserted into the map. You should maintain a doubly-linked list running through all of its entries to keep the correct iteration order.

Note that insertion order is not affected if a key is re-inserted into the map.

6.24.2 Member Typedef Documentation

6.24.2.1 value_type

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::value_type = Pair<const Key, Value>
```

6.24.3 Constructor & Destructor Documentation

6.24.3.1 HashMap() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
ticket::HashMap< Key, Value, Hash, Equal >::HashMap ( )  [default]
```

6.24.3.2 HashMap() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
ticket::HashMap< Key, Value, Hash, Equal >::HashMap (
            const HashMap< Key, Value, Hash, Equal > & other )  [inline]
```

6.24.3.3 ~HashMap()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
ticket::HashMap< Key, Value, Hash, Equal >::~HashMap ( )  [inline]
```

6.24.4 Member Function Documentation

6.24.4.1 at() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::at (
            const Key & key ) -> Value &   [inline]
```

access specified element with bounds checking Returns a reference to the mapped value of the element with key equivalent to key. If no such element exists, an exception of type 'index_out_of_bound'

6.24.4.2 at() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::at (
            const Key & key ) const -> const Value &   [inline]
```

6.24.4.3 begin()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::begin ( ) -> iterator   [inline]
```

return a iterator to the beginning

6.24.4.4 cbegin()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::cbegin ( ) const -> const_iterator   [inline]
```

6.24.4.5 cend()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::cend ( ) const -> const_iterator   [inline]
```

6.24.4.6 clear()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::clear ( ) -> void   [inline]
```

clears the contents

6.24.4.7 contains()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::contains (
            const Key & key ) const -> bool   [inline]
```

Checks if there is an element with key equivalent to key in the container.

6.24.4.8 count()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::count (
            const Key & key ) const -> size_t   [inline]
```

Returns the number of elements with key that compares equivalent to the specified argument, which is either 1 or 0 since this container does not allow duplicates.

6.24.4.9 empty()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::empty ( ) const -> bool   [inline]
```

checks whether the container is empty

6.24.4.10 end()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::end ( ) -> iterator   [inline]
```

return a iterator to the end

6.24.4.11 erase()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::erase (
            iterator pos ) -> void   [inline]
```

erase the element at pos. throw if pos pointed to a bad element (pos == this->end() || pos points an element out of this)

6.24.4.12 find() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::find (
            const Key & key ) -> iterator   [inline]
```

Finds an element with key equivalent to key. key value of the element to search for. Iterator to an element with key equivalent to key. If no such element is found, past-the-end (see end()) iterator is returned.

6.24.4.13 find() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::find (
            const Key & key ) const -> const_iterator   [inline]
```

6.24.4.14 insert()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::insert (
            const value_type & value ) -> Pair<iterator, bool>   [inline]
```

insert an element. return a pair, the first of the pair is the iterator to the new element (or the element that prevented the insertion), the second one is true if insert successfully, or false.

6.24.4.15 operator=()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::operator= (
            const HashMap< Key, Value, Hash, Equal > & other ) -> HashMap &   [inline]
```

6.24.4.16 operator[]() [1/2]

```
template<typename Key , typename Value , typename Hash = std↩
::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::operator[] (
            const Key & key ) -> Value &   [inline]
```

access specified element Returns a reference to the value that is mapped to a key equivalent to key, performing an insertion if such key does not already exist.

6.24.4.17 operator[]() [2/2]

```
template<typename Key , typename Value , typename Hash = std↩
::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::operator[] (
            const Key & key ) const -> const Value &   [inline]
```

behave like at() throw index_out_of_bound if such key does not exist.

6.24.4.18 size() template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>> auto ticket::HashMap< Key, Value, Hash, Equal >::size ( ) const -> size_t [inline]

returns the number of elements.

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

- lib/hashmap.h

6.25 ticket::file::Index< Key, Model > Class Template Reference

Class representing an index file.

`#include <index.h>`

Public Member Functions

- Index (Key Model::*ptr, const char *filename)
  initializes the index.
- auto insert (const Model &model) -> void
  inserts an object into the index.
- auto remove (const Model &model) -> void
  removes an object from the index.
- auto findOne (const Key &key) -> Optional< Model >
  finds one Model in the index.
- auto findOneId (const Key &key) -> Optional< int >
  finds one identifier in the index.
- auto findMany (const Key &key) -> Vector< Model >
  finds all Models of the given key in the index.
- auto findManyId (const Key &key) -> Vector< int >
  finds all IDs of the given keys in the index.
- auto empty () -> bool
  checks if the index is empty.
- auto truncate () -> void
  deletes all entries.

6.25.1 Detailed Description

template<typename Key, typename Model>
class ticket::file::Index< Key, Model >

Class representing an index file.

The Index maps Key to Model's numerical identifier, and provides methods to directly retrieve model objects from data files.

Model needs to be a subclass of ManagedObject.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 Index() template<typename Key , typename Model > ticket::file::Index< Key, Model >::Index (Key Model::* *ptr*, const char * *filename*) [inline]

initializes the index.

Parameters

ptr	the member pointer of the key.
filename	file to store the key.
datafile	the main file where data is stored.

6.25.3 Member Function Documentation

6.25.3.1 empty() template<typename Key , typename Model >

```
auto ticket::file::Index< Key, Model >::empty ( ) -> bool   [inline]
```

checks if the index is empty.

6.25.3.2 findMany() template<typename Key , typename Model >

```
auto ticket::file::Index< Key, Model >::findMany (
            const Key & key ) -> Vector<Model>   [inline]
```

finds all Models of the given key in the index.

6.25.3.3 findManyId() template<typename Key , typename Model >

```
auto ticket::file::Index< Key, Model >::findManyId (
            const Key & key ) -> Vector<int>   [inline]
```

finds all IDs of the given keys in the index.

6.25.3.4 findOne() template<typename Key , typename Model >

```
auto ticket::file::Index< Key, Model >::findOne (
            const Key & key ) -> Optional<Model>   [inline]
```

finds one Model in the index.

6.25.3.5 findOneId() template<typename Key , typename Model >

```
auto ticket::file::Index< Key, Model >::findOneId (
            const Key & key ) -> Optional<int>   [inline]
```

finds one identifier in the index.

6.25.3.6 insert() template<typename Key , typename Model >

```
auto ticket::file::Index< Key, Model >::insert (
            const Model & model ) -> void   [inline]
```

inserts an object into the index.

6.25.3.7 remove() template<typename Key , typename Model >

```
auto ticket::file::Index< Key, Model >::remove (
            const Model & model ) -> void   [inline]
```

removes an object from the index.

6.25.3.8 truncate() template<typename Key , typename Model >

```
auto ticket::file::Index< Key, Model >::truncate ( ) -> void   [inline]
```

deletes all entries.

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

- lib/file/index.h

6.26 ticket::file::Index< Varchar< maxLength >, Model > Class Template Reference

Specialization of Index on Varchar.

`#include <index.h>`

Public Member Functions

- Index (Key Model::*ptr, const char *filename)
  initializes the index.
- auto insert (const Model &model) -> void
  inserts an object into the index.
- auto remove (const Model &model) -> void
  removes an object from the index.
- auto findOne (const Key &key) -> Optional< Model >
  finds one Model in the index.
- auto findOneId (const Key &key) -> Optional< int >
  finds one identifier in the index.
- auto findMany (const Key &key) -> Vector< Model >
  finds all Models of the given key in the index.
- auto findManyId (const Key &key) -> Vector< int >
  finds all IDs of the given keys in the index.
- auto empty () -> bool
  checks if the index is empty.
- auto truncate () -> void
  deletes all entries.

6.26.1 Detailed Description

template<size_t maxLength, typename Model>
class ticket::file::Index< Varchar< maxLength >, Model >

Specialization of Index on Varchar.

It makes use of hashes to speed up the process.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Index() `template<size_t maxLength, typename Model >`

```
ticket::file::Index< Varchar< maxLength >, Model >::Index (
            Key Model::* ptr,
            const char * filename )  [inline]
```

initializes the index.

Parameters

ptr	the member pointer of the key.
filename	file to store the key.
datafile	the main file where data is stored.

6.26.3 Member Function Documentation

6.26.3.1 empty() `template<size_t maxLength, typename Model >`

```
auto ticket::file::Index< Varchar< maxLength >, Model >::empty ( ) -> bool   [inline]
```

checks if the index is empty.

6.26.3.2 findMany() `template<size_t maxLength, typename Model >`

```
auto ticket::file::Index< Varchar< maxLength >, Model >::findMany (
            const Key & key ) -> Vector<Model>   [inline]
```

finds all Models of the given key in the index.

6.26.3.3 findManyId() template<size_t maxLength, typename Model >

```
auto ticket::file::Index< Varchar< maxLength >, Model >::findManyId (
            const Key & key ) -> Vector<int>   [inline]
```

finds all IDs of the given keys in the index.

6.26.3.4 findOne() template<size_t maxLength, typename Model >

```
auto ticket::file::Index< Varchar< maxLength >, Model >::findOne (
            const Key & key ) -> Optional<Model>   [inline]
```

finds one Model in the index.

6.26.3.5 findOneId() template<size_t maxLength, typename Model >

```
auto ticket::file::Index< Varchar< maxLength >, Model >::findOneId (
            const Key & key ) -> Optional<int>   [inline]
```

finds one identifier in the index.

6.26.3.6 insert() template<size_t maxLength, typename Model >

```
auto ticket::file::Index< Varchar< maxLength >, Model >::insert (
            const Model & model ) -> void   [inline]
```

inserts an object into the index.

6.26.3.7 remove() template<size_t maxLength, typename Model >

```
auto ticket::file::Index< Varchar< maxLength >, Model >::remove (
            const Model & model ) -> void   [inline]
```

removes an object from the index.

6.26.3.8 truncate() template<size_t maxLength, typename Model >

```
auto ticket::file::Index< Varchar< maxLength >, Model >::truncate ( ) -> void   [inline]
```

deletes all entries.

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

- lib/file/index.h

6.27 ticket::Instant Class Reference

Class representing a point of time in a day.

`#include <datetime.h>`

Public Member Functions

- Instant ()=default
- Instant (int hour, int minute)
- Instant (const char *str)

  constructs an Instant from an HH:MM format string.
- auto daysOverflow () const -> int
- auto hour () const -> int
- auto minute () const -> int
- operator std::string () const

  gets an HH:MM representation of the Instant.
- auto operator+ (Duration dt) const -> Instant
- auto operator- (Duration dt) const -> Instant
- auto operator- (Instant rhs) const -> Duration
- auto operator< (const Instant &rhs) const -> bool

6.27.1 Detailed Description

Class representing a point of time in a day.

An Instant may overflow, and this class takes care of that by daysOverflow().

Not to be confused with Duration, see notes in Duration.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 Instant() [1/3] ticket::Instant::Instant ( ) [default]

6.27.2.2 Instant() [2/3] ticket::Instant::Instant (int *hour*, int *minute*)

6.27.2.3 Instant() [3/3] ticket::Instant::Instant (const char * *str*) [explicit]

constructs an Instant from an HH:MM format string.

6.27.3 Member Function Documentation

6.27.3.1 daysOverflow() `auto ticket::Instant::daysOverflow ( ) const -> int`

6.27.3.2 hour() `auto ticket::Instant::hour ( ) const -> int`

6.27.3.3 minute() `auto ticket::Instant::minute ( ) const -> int`

6.27.3.4 operator std::string() `ticket::Instant::operator std::string ( ) const`

gets an HH:MM representation of the Instant.

6.27.3.5 operator+()

```
auto ticket::Instant::operator+ (
    Duration dt ) const -> Instant
```

6.27.3.6 operator-() [1/2]

```
auto ticket::Instant::operator- (
    Duration dt ) const -> Instant
```

6.27.3.7 operator-() [2/2]

```
auto ticket::Instant::operator- (
    Instant rhs ) const -> Duration
```

6.27.3.8 operator<()

```
auto ticket::Instant::operator< (
    const Instant & rhs ) const -> bool
```

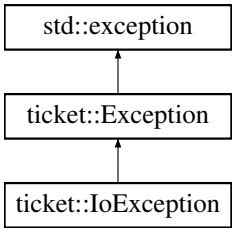
The documentation for this class was generated from the following files:

- lib/datetime.h
- lib/datetime.cpp

6.28 ticket::IoException Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::IoException:



Public Member Functions

- IoException ()
- IoException (const char *what)

6.28.1 Constructor & Destructor Documentation

6.28.1.1 IoException() [1/2] ticket::IoException::IoException ( ) [inline]

6.28.1.2 IoException() [2/2] ticket::IoException::IoException (const char * *what*) [inline]

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

- lib/exception.h

6.29 ticket::HashMap< Key, Value, Hash, Equal >::iterator Class Reference

`#include <hashmap.h>`

Public Types

- using difference_type = std::ptrdiff_t
- using value_type = HashMap::value_type
- using pointer = value_type *
- using reference = value_type &
- using iterator_category = std::output_iterator_tag

Public Member Functions

- iterator ()=default
- iterator (ListNode *node, HashMap *home)
- auto operator++ (int) -> iterator
- auto operator++ () -> iterator &
- auto operator-- (int) -> iterator
- auto operator-- () -> iterator &
- auto operator* () const -> reference
- auto operator== (const iterator &rhs) const -> bool
- auto operator== (const const_iterator &rhs) const -> bool
- auto operator!= (const iterator &rhs) const -> bool
- auto operator!= (const const_iterator &rhs) const -> bool
- auto operator-> () const noexcept -> pointer

Friends

- class const_iterator
- class HashMap

6.29.1 Member Typedef Documentation

6.29.1.1 difference_type

```
template<typename Key , typename Value , typename Hash = std::hash<↩
Key>, typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::iterator::difference_type = std::ptrdiff_t
```

6.29.1.2 iterator_category

```
template<typename Key , typename Value , typename Hash = std::hash<↩
Key>, typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::iterator::iterator_category = std::output_↩
iterator_tag
```

6.29.1.3 pointer

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::iterator::pointer = value_type *
```

6.29.1.4 reference

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::iterator::reference = value_type &
```

6.29.1.5 value_type

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
using ticket::HashMap< Key, Value, Hash, Equal >::iterator::value_type = HashMap::value_type
```

6.29.2 Constructor & Destructor Documentation

6.29.2.1 iterator() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<↩
Key>, typename Equal = std::equal_to<Key>>
ticket::HashMap< Key, Value, Hash, Equal >::iterator::iterator ( )  [default]
```

6.29.2.2 iterator() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<↩
Key>, typename Equal = std::equal_to<Key>>
ticket::HashMap< Key, Value, Hash, Equal >::iterator::iterator (
            ListNode * node,
            HashMap * home )  [inline]
```

6.29.3 Member Function Documentation

6.29.3.1 operator"!=() [1/2]

```
template<typename Key , typename Value , typename Hash = std↩
::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator!= (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.29.3.2 operator"!=() [2/2]

```
template<typename Key , typename Value , typename Hash = std↩
::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator!= (
            const iterator & rhs ) const -> bool   [inline]
```

6.29.3.3 operator*()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>,
typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator* ( ) const -> reference
[inline]
```

6.29.3.4 operator++() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator++ ( ) -> iterator &   [inline]
```

6.29.3.5 operator++() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator++ (
            int  ) -> iterator   [inline]
```

6.29.3.6 operator--() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator-- ( ) -> iterator &   [inline]
```

6.29.3.7 operator--() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator-- (
            int  ) -> iterator   [inline]
```

6.29.3.8 operator->()

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator-> ( ) const -> pointer
[inline], [noexcept]
```

6.29.3.9 operator==() [1/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator== (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.29.3.10 operator==() [2/2]

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
auto ticket::HashMap< Key, Value, Hash, Equal >::iterator::operator== (
            const iterator & rhs ) const -> bool   [inline]
```

6.29.4 Friends And Related Function Documentation

6.29.4.1 const_iterator

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
friend class const_iterator  [friend]
```

6.29.4.2 HashMap

```
template<typename Key , typename Value , typename Hash = std::hash<Key>, typename Equal = std::equal_to<Key>>
friend class HashMap  [friend]
```

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

- lib/hashmap.h

6.30 ticket::Vector< T >::iterator Class Reference

`#include <vector.h>`

Public Types

- using difference_type = std::ptrdiff_t
- using value_type = T
- using pointer = T *
- using reference = T &
- using iterator_category = std::random_access_iterator_tag

Public Member Functions

- auto operator+ (const int &n) const -> iterator
- auto operator- (const int &n) const -> iterator
- auto operator- (const iterator &rhs) const -> int
- auto operator+= (const int &n) -> iterator &
- auto operator-= (const int &n) -> iterator &
- auto operator++ (int) -> iterator
- auto operator++ () -> iterator &
- auto operator-- (int) -> iterator
- auto operator-- () -> iterator &
- auto operator* () const -> T &
- auto operator== (const iterator &rhs) const -> bool
- auto operator== (const const_iterator &rhs) const -> bool
- auto operator!= (const iterator &rhs) const -> bool
- auto operator!= (const const_iterator &rhs) const -> bool
- auto operator< (const iterator &rhs) const -> bool
- auto operator< (const const_iterator &rhs) const -> bool

Friends

- class const_iterator
- class Vector

6.30.1 Member Typedef Documentation

6.30.1.1 difference_type template<typename T >

```
using ticket::Vector< T >::iterator::difference_type = std::ptrdiff_t
```

6.30.1.2 iterator_category template<typename T >

```
using ticket::Vector< T >::iterator::iterator_category = std::random_access_iterator_tag
```

6.30.1.3 pointer template<typename T >

```
using ticket::Vector< T >::iterator::pointer = T *
```

6.30.1.4 reference template<typename T >

```
using ticket::Vector< T >::iterator::reference = T &
```

6.30.1.5 value_type template<typename T >

```
using ticket::Vector< T >::iterator::value_type = T
```

6.30.2 Member Function Documentation

6.30.2.1 operator"!=() [1/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator!= (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.30.2.2 operator"!=() [2/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator!= (
            const iterator & rhs ) const -> bool   [inline]
```

some other operator for iterator.

6.30.2.3 operator*() template<typename T >

```
auto ticket::Vector< T >::iterator::operator* ( ) const -> T &   [inline]
```

6.30.2.4 operator+() template<typename T >

```
auto ticket::Vector< T >::iterator::operator+ (
            const int & n ) const -> iterator   [inline]
```

6.30.2.5 operator++() [1/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator++ ( ) -> iterator &   [inline]
```

6.30.2.6 operator++() [2/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator++ (
            int  ) -> iterator   [inline]
```

6.30.2.7 operator+=() template<typename T >

```
auto ticket::Vector< T >::iterator::operator+= (
            const int & n ) -> iterator &   [inline]
```

6.30.2.8 operator-() [1/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator- (
            const int & n ) const -> iterator   [inline]
```

6.30.2.9 operator-() [2/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator- (
            const iterator & rhs ) const -> int   [inline]
```

6.30.2.10 operator--() [1/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator-- ( ) -> iterator &   [inline]
```

6.30.2.11 operator--() [2/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator-- (
            int  ) -> iterator   [inline]
```

6.30.2.12 operator-=() template<typename T >

```
auto ticket::Vector< T >::iterator::operator-= (
            const int & n ) -> iterator &   [inline]
```

6.30.2.13 operator<() [1/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator< (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.30.2.14 operator<() [2/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator< (
            const iterator & rhs ) const -> bool   [inline]
```

6.30.2.15 operator==() [1/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator== (
            const const_iterator & rhs ) const -> bool   [inline]
```

6.30.2.16 operator==() [2/2] template<typename T >

```
auto ticket::Vector< T >::iterator::operator== (
            const iterator & rhs ) const -> bool   [inline]
```

a operator to check whether two iterators are same (pointing to the same memory address).

6.30.3 Friends And Related Function Documentation

6.30.3.1 const_iterator template<typename T >

```
friend class const_iterator  [friend]
```

6.30.3.2 Vector template<typename T >

```
friend class Vector  [friend]
```

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

- lib/vector.h

6.31 ticket::rollback::LogEntryBase Struct Reference

`#include <rollback.h>`

Public Types

- using Content = Variant< AddUser, ModifyProfile, AddTrain, DeleteTrain, ReleaseTrain, BuyTicket, RefundTicket, FulfillOrder >

Public Attributes

- int timestamp
- Content content

Static Public Attributes

- static constexpr const char * filename = "rollback-log"

6.31.1 Member Typedef Documentation

6.31.1.1 Content

`using ticket::rollback::LogEntryBase::Content = Variant< AddUser, ModifyProfile, AddTrain, DeleteTrain, ReleaseTrain, BuyTicket, RefundTicket, FulfillOrder >`

6.31.2 Member Data Documentation

6.31.2.1 content

`Content ticket::rollback::LogEntryBase::content`

6.31.2.2 filename

`constexpr const char* ticket::rollback::LogEntryBase::filename = "rollback-log" [static], [constexpr]`

6.31.2.3 timestamp

`int ticket::rollback::LogEntryBase::timestamp`

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

- src/rollback.h

6.32 ticket::command::Login Struct Reference

`#include <parser.h>`

Public Attributes

- std::string username
- std::string password

6.32.1 Member Data Documentation

6.32.1.1 password `std::string ticket::command::Login::password`

6.32.1.2 username `std::string ticket::command::Login::username`

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

- src/parser.h

6.33 ticket::command::Logout Struct Reference

`#include <parser.h>`

Public Attributes

- std::string username

6.33.1 Member Data Documentation

6.33.1.1 username `std::string ticket::command::Logout::username`

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

- src/parser.h

6.34 ticket::LruCache< Key, kSize > Class Template Reference

A fixed-size cache with a least recently used policy.

`#include <lru-cache.h>`

Public Member Functions

- ~LruCache ()
- auto get (const Key &key) -> Optional< void * >

  tries to obtain the value at the designated key.
- auto upsert (const Key &key, const void *buf, int length) -> bool

  upserts an cache entry.
- auto remove (const Key &key) -> void

  removes the key from the cache.
- auto clear () -> void

  clears the cache.

6.34.1 Detailed Description

template<typename Key, int kSize>
class ticket::LruCache< Key, kSize >

A fixed-size cache with a least recently used policy.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 ~LruCache() `template<typename Key , int kSize>`

`ticket::LruCache< Key, kSize >::~LruCache ( ) [inline]`

6.34.3 Member Function Documentation

6.34.3.1 clear() `template<typename Key , int kSize>`

`auto ticket::LruCache< Key, kSize >::clear ( ) -> void [inline]`

clears the cache.

6.34.3.2 get() template<typename Key , int kSize>

```
auto ticket::LruCache< Key, kSize >::get (
            const Key & key ) -> Optional<void *>   [inline]
```

tries to obtain the value at the designated key.

6.34.3.3 remove() template<typename Key , int kSize>

```
auto ticket::LruCache< Key, kSize >::remove (
            const Key & key ) -> void   [inline]
```

removes the key from the cache.

6.34.3.4 upsert() template<typename Key , int kSize>

```
auto ticket::LruCache< Key, kSize >::upsert (
            const Key & key,
            const void * buf,
            int length ) -> bool   [inline]
```

upserts an cache entry.

Returns

true if the cache state has changed.

performs an insert if the key is not in the cache, or an update if the key is in the cache.

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

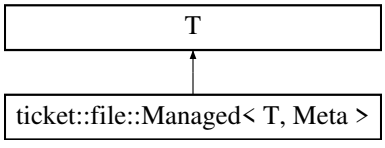
- lib/lru-cache.h

6.35 ticket::file::Managed< T, Meta > Class Template Reference

an opinionated utility class wrapper for the objects to be stored.

`#include <file.h>`

Inheritance diagram for ticket::file::Managed< T, Meta >:



Public Member Functions

- auto id () const -> int

  the unique immutable numeral identifier of the object.
- auto save () -> void

  saves the object into the file.
- auto update () -> void

  updates a modified object.
- auto destroy () -> void

  removes the object from the file.

Static Public Member Functions

- static auto get (size_t id) -> Managed

  gets the object at id in file.
- static auto truncate () -> void

  hard deletes all objects.

Static Public Attributes

- static File< Meta, sizeof(T)> file { T::filename }

  The underlying file storage.

6.35.1 Detailed Description

template<typename T, typename Meta = Unit>
class ticket::file::Managed< T, Meta >

an opinionated utility class wrapper for the objects to be stored.

it handles get, update, and push for the object.

the base class needs to have a static char *filename.

6.35.2 Member Function Documentation

6.35.2.1 destroy() `template<typename T , typename Meta = Unit>`

`auto ticket::file::Managed< T, Meta >::destroy ( ) -> void [inline]`

removes the object from the file.

6.35.2.2 get() template<typename T , typename Meta = Unit>

```
static auto ticket::file::Managed< T, Meta >::get (
            size_t id ) -> Managed   [inline], [static]
```

gets the object at id in file.

6.35.2.3 id() template<typename T , typename Meta = Unit>

```
auto ticket::file::Managed< T, Meta >::id ( ) const -> int   [inline]
```

the unique immutable numeral identifier of the object.

this identifier would not change on update, but may be reused when deleted.

6.35.2.4 save() template<typename T , typename Meta = Unit>

```
auto ticket::file::Managed< T, Meta >::save ( ) -> void   [inline]
```

saves the object into the file.

The object needs to be new, i.e. not saved before. To update the object after a modification, use update().

6.35.2.5 truncate() template<typename T , typename Meta = Unit>

```
static auto ticket::file::Managed< T, Meta >::truncate ( ) -> void   [inline], [static]
```

hard deletes all objects.

6.35.2.6 update() template<typename T , typename Meta = Unit>

```
auto ticket::file::Managed< T, Meta >::update ( ) -> void   [inline]
```

updates a modified object.

6.35.3 Member Data Documentation

6.35.3.1 file template<typename T , typename Meta >

```
File< Meta, sizeof(T)> ticket::file::Managed< T, Meta >::file { T::filename }  [static]
```

The underlying file storage.

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

- lib/file/file.h

6.36 ticket::Map< KeyType, ValueType, Compare > Class Template Reference

A sorted key-value map backed by a red-black tree.

`#include <map.h>`

Public Types

- using value_type = Pair< const KeyType, ValueType >
- using iterator = typename TreeType::iterator
- using const_iterator = typename TreeType::const_iterator

Public Member Functions

- Map ()=default
- auto at (const KeyType &key) -> ValueType &
- auto at (const KeyType &key) const -> const ValueType &
- auto operator[ ] (const KeyType &key) -> ValueType &
- auto operator[ ] (const KeyType &key) const -> const ValueType &
- auto begin () -> iterator
- auto cbegin () const -> const_iterator
- auto end () -> iterator
- auto cend () const -> const_iterator
- auto empty () const -> bool
- auto size () const -> size_t
- auto clear () -> void
- auto insert (const value_type &value) -> Pair< iterator, bool >
- auto erase (iterator pos) -> void
- auto count (const KeyType &key) const -> size_t
- auto find (const KeyType &key) -> iterator
- auto find (const KeyType &key) const -> const_iterator

6.36.1 Detailed Description

template<typename KeyType, typename ValueType, typename Compare = internal::LessOp>
class ticket::Map< KeyType, ValueType, Compare >

A sorted key-value map backed by a red-black tree.

6.36.2 Member Typedef Documentation

6.36.2.1 const_iterator

```
template<typename KeyType , typename ValueType , typename Compare =
internal::LessOp>
using ticket::Map< KeyType, ValueType, Compare >::const_iterator = typename TreeType::const_↩
iterator
```

6.36.2.2 iterator

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
using ticket::Map< KeyType, ValueType, Compare >::iterator = typename TreeType::iterator
```

6.36.2.3 value_type

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
using ticket::Map< KeyType, ValueType, Compare >::value_type = Pair<const KeyType, ValueType>
```

6.36.3 Constructor & Destructor Documentation

6.36.3.1 Map()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
ticket::Map< KeyType, ValueType, Compare >::Map ( )  [default]
```

6.36.4 Member Function Documentation

6.36.4.1 at() [1/2]

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::at (
            const KeyType & key ) -> ValueType &   [inline]
```

access specified element with bounds checking Returns a reference to the mapped value of the element with key equivalent to key. If no such element exists, an exception of type 'index_out_of_bound'

6.36.4.2 at() [2/2]

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::at (
            const KeyType & key ) const -> const ValueType &   [inline]
```

6.36.4.3 begin()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::begin ( ) -> iterator   [inline]
```

return a iterator to the beginning

6.36.4.4 cbegin()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::cbegin ( ) const -> const_iterator   [inline]
```

6.36.4.5 cend()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::cend ( ) const -> const_iterator   [inline]
```

6.36.4.6 clear()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::clear ( ) -> void   [inline]
```

clears the contents

6.36.4.7 count()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::count (
            const KeyType & key ) const -> size_t   [inline]
```

Returns the number of elements with key that compares equivalent to the specified argument, which is either 1 or 0 since this container does not allow duplicates. The default method of check the equivalence is !(a < b || b > a)

6.36.4.8 empty()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::empty ( ) const -> bool   [inline]
```

checks whether the container is empty return true if empty, otherwise false.

6.36.4.9 end()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::end ( ) -> iterator   [inline]
```

return a iterator to the end in fact, it returns past-the-end.

6.36.4.10 erase()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::erase (
            iterator pos ) -> void   [inline]
```

erase the element at pos. throw if pos pointed to a bad element (pos == this->end() || pos points an element out of this)

6.36.4.11 find() [1/2]

```
template<typename KeyType , typename ValueType , typename Compare =
internal::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::find (
            const KeyType & key ) -> iterator   [inline]
```

Finds an element with key equivalent to key. key value of the element to search for. Iterator to an element with key equivalent to key. If no such element is found, past-the-end (see end()) iterator is returned.

6.36.4.12 find() [2/2]

```
template<typename KeyType , typename ValueType , typename Compare =
internal::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::find (
            const KeyType & key ) const -> const_iterator   [inline]
```

6.36.4.13 insert()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::insert (
            const value_type & value ) -> Pair<iterator, bool>   [inline]
```

insert an element. return a pair, the first of the pair is the iterator to the new element (or the element that prevented the insertion), the second one is true if insert successfully, or false.

6.36.4.14 operator[]() [1/2]

```
template<typename KeyType , typename ValueType , typename Compare =
internal::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::operator[] (
            const KeyType & key ) -> ValueType &   [inline]
```

access specified element Returns a reference to the value that is mapped to a key equivalent to key, performing an insertion if such key does not already exist.

6.36.4.15 operator[]() [2/2]

```
template<typename KeyType , typename ValueType , typename Compare =
internal::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::operator[] (
            const KeyType & key ) const -> const ValueType &   [inline]
```

behave like at() throw index_out_of_bound if such key does not exist.

6.36.4.16 size()

```
template<typename KeyType , typename ValueType , typename Compare = internal↩
::LessOp>
auto ticket::Map< KeyType, ValueType, Compare >::size ( ) const -> size_t   [inline]
```

returns the number of elements.

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

- lib/map.h

6.37 ticket::command::ModifyProfile Struct Reference

`#include <parser.h>`

Public Attributes

- std::string currentUser
- std::string username
- Optional< std::string > password
- Optional< std::string > name
- Optional< std::string > email
- Optional< int > privilege

6.37.1 Member Data Documentation

6.37.1.1 currentUser `std::string ticket::command::ModifyProfile::currentUser`

6.37.1.2 email `Optional<std::string> ticket::command::ModifyProfile::email`

6.37.1.3 name `Optional<std::string> ticket::command::ModifyProfile::name`

6.37.1.4 password `Optional<std::string> ticket::command::ModifyProfile::password`

6.37.1.5 privilege `Optional<int> ticket::command::ModifyProfile::privilege`

6.37.1.6 username `std::string ticket::command::ModifyProfile::username`

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

- src/parser.h

6.38 ticket::rollback::ModifyProfile Struct Reference

`#include <rollback.h>`

Public Attributes

- int id
- Optional< User::Password > password
- Optional< User::Name > name
- Optional< User::Email > email
- Optional< User::Privilege > privilege

6.38.1 Member Data Documentation

6.38.1.1 email `Optional<User::Email> ticket::rollback::ModifyProfile::email`

6.38.1.2 id `int ticket::rollback::ModifyProfile::id`

6.38.1.3 name `Optional<User::Name> ticket::rollback::ModifyProfile::name`

6.38.1.4 password `Optional<User::Password> ticket::rollback::ModifyProfile::password`

6.38.1.5 privilege `Optional<User::Privilege> ticket::rollback::ModifyProfile::privilege`

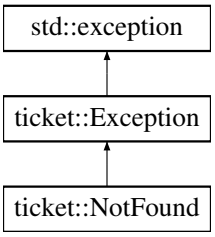
The documentation for this struct was generated from the following file:

- src/rollback.h

6.39 ticket::NotFound Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::NotFound:



Public Member Functions

- NotFound ()
- NotFound (const char *what)

6.39.1 Constructor & Destructor Documentation

6.39.1.1 NotFound() [1/2] ticket::NotFound::NotFound ( ) [inline]

6.39.1.2 NotFound() [2/2] ticket::NotFound::NotFound (const char * what) [inline]

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

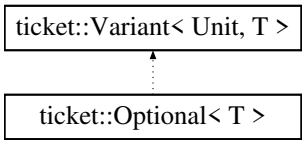
- lib/exception.h

6.40 ticket::Optional< T > Class Template Reference

A resemblence of std::optional.

`#include <optional.h>`

Inheritance diagram for ticket::Optional< T >:



Public Member Functions

- Optional ()=default
- Optional (Unit)

  constructs a empty optional.
- template<typename Init , typename = std::enable_if_t<!std::is_same_v<Init, Unit>>>
  Optional (const Init &value)

  constructs a filled optional.
- auto operator= (Unit unit) -> Optional &
- template<typename Init , typename = std::enable_if_t<!std::is_same_v<Init, Unit>>>
  auto operator= (const Init &value) -> Optional &
- operator bool () const

  true if the optional has value.
- auto operator* () -> T &

  provides access to the actual object.
- auto operator* () const -> const T &
- auto operator-> () -> T *
- auto operator-> () const -> const T *

6.40.1 Detailed Description

template<typename T>
class ticket::Optional< T >

A resemblence of std::optional.

This class represents a state, or nothing at all. This is sometimes better than using null pointers, as it avoids the problem that a reference cannot be null. Internally it is a variant of Unit and T, therefore some may write Optional<↩ T> = T? = T | Unit = T | null or whatever.

6.40.2 Constructor & Destructor Documentation

6.40.2.1 Optional() [1/3] template<typename T >

```
ticket::Optional< T >::Optional ( )  [default]
```

6.40.2.2 Optional() [2/3] template<typename T >

```
ticket::Optional< T >::Optional (
            Unit  )  [inline]
```

constructs a empty optional.

6.40.2.3 Optional() [3/3] template<typename T >

```
template<typename Init , typename = std::enable_if_t<!std::is_same_v<Init, Unit>>>
ticket::Optional< T >::Optional (
            const Init & value )  [inline]
```

constructs a filled optional.

6.40.3 Member Function Documentation

6.40.3.1 operator bool() template<typename T >

```
ticket::Optional< T >::operator bool ( ) const  [inline]
```

true if the optional has value.

6.40.3.2 operator*() [1/2] template<typename T >

```
auto ticket::Optional< T >::operator* ( ) -> T &   [inline]
```

provides access to the actual object.

6.40.3.3 operator*() [2/2] template<typename T >

```
auto ticket::Optional< T >::operator* ( ) const -> const T &   [inline]
```

6.40.3.4 operator->() [1/2] template<typename T >

```
auto ticket::Optional< T >::operator-> ( ) -> T *   [inline]
```

6.40.3.5 operator->() [2/2] template<typename T >

```
auto ticket::Optional< T >::operator-> ( ) const -> const T *   [inline]
```

6.40.3.6 operator=() [1/2] template<typename T >

```
template<typename Init , typename = std::enable_if_t<!std::is_same_v<Init, Unit>>>
auto ticket::Optional< T >::operator= (
            const Init & value ) -> Optional &   [inline]
```

6.40.3.7 operator=() [2/2] template<typename T >

```
auto ticket::Optional< T >::operator= (
            Unit unit ) -> Optional &   [inline]
```

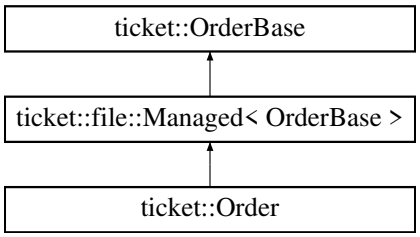
The documentation for this class was generated from the following file:

- lib/optional.h

6.41 ticket::Order Struct Reference

```
#include <order.h>
```

Inheritance diagram for ticket::Order:



Public Member Functions

- Order ()=default
- Order (const file::Managed< OrderBase > &order)

Static Public Attributes

- static file::Index< User::Id, Order > ixUserId {&Order::user, "orders.user.ix"}
- static file::Index< Ride, Order > pendingOrders

Additional Inherited Members

6.41.1 Constructor & Destructor Documentation

6.41.1.1 Order() [1/2] ticket::Order::Order ( ) [default]

6.41.1.2 Order() [2/2] ticket::Order::Order (const file::Managed< OrderBase > & *order*) [inline]

6.41.2 Member Data Documentation

6.41.2.1 ixUserId file::Index< User::Id, Order > ticket::Order::ixUserId {&Order::user, "orders.↩ user.ix"} [static]

6.41.2.2 pendingOrders file::Index< Ride, Order > ticket::Order::pendingOrders [static]

Initial value:

```
{
  &Order::ride,
  "orders-pending.ride.ix"
}
```

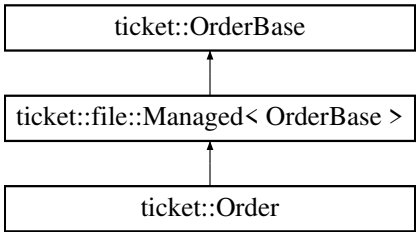
The documentation for this struct was generated from the following files:

- src/order.h
- src/order.cpp

6.42 ticket::OrderBase Struct Reference

`#include <order.h>`

Inheritance diagram for ticket::OrderBase:



Public Types

- enum Status { kSuccess , kPending , kRefunded }
- using Id = int

Public Member Functions

- auto getTrain () -> Train

  gets the corresponding train object.

Static Public Member Functions

- static auto statusString (Status status) -> const char *

  gets the string representation of the status.

Public Attributes

- User::Id user
- Ride ride
- int ixFrom
- int ixTo
- int seats
- int price
- Status status
- OrderCache cache

Static Public Attributes

- static constexpr const char * filename = "orders"

6.42.1 Member Typedef Documentation

6.42.1.1 Id `using ticket::OrderBase::Id = int`

6.42.2 Member Enumeration Documentation

6.42.2.1 Status `enum ticket::OrderBase::Status`

Enumerator

kSuccess	
kPending	
kRefunded	

6.42.3 Member Function Documentation

6.42.3.1 getTrain() auto ticket::OrderBase::getTrain ( ) -> Train

gets the corresponding train object.

6.42.3.2 statusString() static auto ticket::OrderBase::statusString (Status *status*) -> const char * [inline], [static]

gets the string representation of the status.

6.42.4 Member Data Documentation

6.42.4.1 cache OrderCache ticket::OrderBase::cache

6.42.4.2 filename constexpr const char* ticket::OrderBase::filename = "orders" [static], [constexpr]

6.42.4.3 ixFrom int ticket::OrderBase::ixFrom

6.42.4.4 ixTo int ticket::OrderBase::ixTo

6.42.4.5 price int ticket::OrderBase::price

6.42.4.6 ride Ride ticket::OrderBase::ride

6.42.4.7 seats `int ticket::OrderBase::seats`

6.42.4.8 status `Status ticket::OrderBase::status`

6.42.4.9 user `User::Id ticket::OrderBase::user`

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

- src/order.h

6.43 ticket::OrderCache Struct Reference

`#include <order.h>`

Public Attributes

- Train::Id trainId
- Station::Id from
- Station::Id to
- Instant timeDeparture
- Instant timeArrival

6.43.1 Member Data Documentation

6.43.1.1 from `Station::Id ticket::OrderCache::from`

6.43.1.2 timeArrival `Instant ticket::OrderCache::timeArrival`

6.43.1.3 timeDeparture `Instant ticket::OrderCache::timeDeparture`

6.43.1.4 to `Station::Id ticket::OrderCache::to`

6.43.1.5 trainId `Train::Id ticket::OrderCache::trainId`

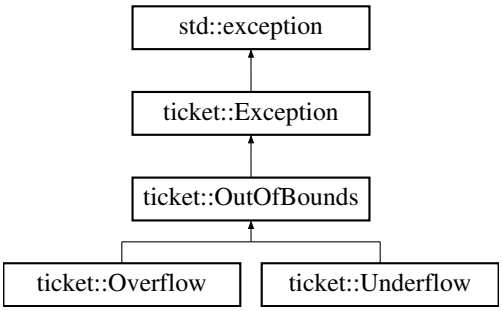
The documentation for this struct was generated from the following file:

- src/order.h

6.44 ticket::OutOfBounds Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::OutOfBounds:



Public Member Functions

- OutOfBounds ()
- OutOfBounds (const char *what)

6.44.1 Constructor & Destructor Documentation

6.44.1.1 OutOfBounds() [1/2] `ticket::OutOfBounds::OutOfBounds ( ) [inline]`

6.44.1.2 OutOfBounds() [2/2] `ticket::OutOfBounds::OutOfBounds ( const char * what ) [inline]`

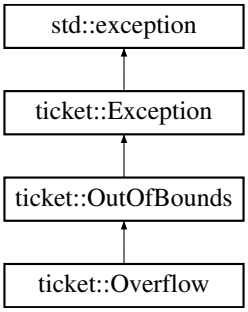
The documentation for this class was generated from the following file:

- lib/exception.h

6.45 ticket::Overflow Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::Overflow:



Public Member Functions

- Overflow ()
- Overflow (const char *what)

6.45.1 Constructor & Destructor Documentation

6.45.1.1 Overflow() [1/2] ticket::Overflow::Overflow ( ) [inline]

6.45.1.2 Overflow() [2/2] ticket::Overflow::Overflow (const char * *what*) [inline]

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

- lib/exception.h

6.46 ticket::Pair< T1, T2 > Class Template Reference

A pair of objects.

`#include <utility.h>`

Public Member Functions

- constexpr Pair ()
- Pair (const Pair &other)=default
- Pair (Pair &&other) noexcept=default
- Pair (const T1 &x, const T2 &y)
- template<class U1 , class U2 >
  Pair (U1 &&x, U2 &&y)
- template<class U1 , class U2 >
  Pair (const Pair< U1, U2 > &other)
- template<class U1 , class U2 >
  Pair (Pair< U1, U2 > &&other)

Public Attributes

- T1 first
- T2 second

6.46.1 Detailed Description

template<typename T1, typename T2>
class ticket::Pair< T1, T2 >

A pair of objects.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 Pair() [1/7] template<typename T1 , typename T2 >

```
constexpr ticket::Pair< T1, T2 >::Pair ( )  [inline], [constexpr]
```

6.46.2.2 Pair() [2/7] template<typename T1 , typename T2 >

```
ticket::Pair< T1, T2 >::Pair (
            const Pair< T1, T2 > & other )  [default]
```

6.46.2.3 Pair() [3/7] template<typename T1 , typename T2 >

```
ticket::Pair< T1, T2 >::Pair (
            Pair< T1, T2 > && other )  [default], [noexcept]
```

6.46.2.4 Pair() [4/7] template<typename T1 , typename T2 >

```
ticket::Pair< T1, T2 >::Pair (
            const T1 & x,
            const T2 & y )  [inline]
```

6.46.2.5 Pair() [5/7] template<typename T1 , typename T2 >

```
template<class U1 , class U2 >
ticket::Pair< T1, T2 >::Pair (
            U1 && x,
            U2 && y )  [inline]
```

6.46.2.6 Pair() [6/7] template<typename T1 , typename T2 >

```
template<class U1 , class U2 >
ticket::Pair< T1, T2 >::Pair (
            const Pair< U1, U2 > & other )  [inline]
```

6.46.2.7 Pair() [7/7] template<typename T1 , typename T2 >

```
template<class U1 , class U2 >
ticket::Pair< T1, T2 >::Pair (
            Pair< U1, U2 > && other )  [inline]
```

6.46.3 Member Data Documentation

6.46.3.1 first template<typename T1 , typename T2 >

```
T1 ticket::Pair< T1, T2 >::first
```

6.46.3.2 second template<typename T1 , typename T2 >

```
T2 ticket::Pair< T1, T2 >::second
```

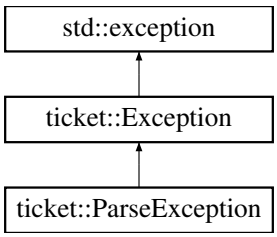
The documentation for this class was generated from the following file:

- lib/utility.h

6.47 ticket::ParseException Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::ParseException:



Public Member Functions

- ParseException ()
- ParseException (const char *what)

6.47.1 Constructor & Destructor Documentation

6.47.1.1 ParseException() [1/2] ticket::ParseException::ParseException ( ) [inline]

6.47.1.2 ParseException() [2/2] ticket::ParseException::ParseException (const char * *what*) [inline]

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

- lib/exception.h

6.48 ticket::command::QueryOrder Struct Reference

`#include <parser.h>`

Public Attributes

- std::string currentUser

6.48.1 Member Data Documentation

6.48.1.1 currentUser `std::string ticket::command::QueryOrder::currentUser`

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

- src/parser.h

6.49 ticket::command::QueryProfile Struct Reference

`#include <parser.h>`

Public Attributes

- std::string currentUser
- std::string username

6.49.1 Member Data Documentation

6.49.1.1 currentUser `std::string ticket::command::QueryProfile::currentUser`

6.49.1.2 username `std::string ticket::command::QueryProfile::username`

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

- src/parser.h

6.50 ticket::command::QueryTicket Struct Reference

`#include <parser.h>`

Public Attributes

- std::string from
- std::string to
- Date date
- SortType sort = kTime

6.50.1 Member Data Documentation

6.50.1.1 date `Date ticket::command::QueryTicket::date`

6.50.1.2 from `std::string ticket::command::QueryTicket::from`

6.50.1.3 sort `SortType ticket::command::QueryTicket::sort = kTime`

6.50.1.4 to `std::string ticket::command::QueryTicket::to`

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

- src/parser.h

6.51 ticket::command::QueryTrain Struct Reference

`#include <parser.h>`

Public Attributes

- std::string id
- Date date

6.51.1 Member Data Documentation

6.51.1.1 date `Date ticket::command::QueryTrain::date`

6.51.1.2 id `std::string ticket::command::QueryTrain::id`

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

- src/parser.h

6.52 ticket::command::QueryTransfer Struct Reference

`#include <parser.h>`

Public Attributes

- std::string from
- std::string to
- Date date
- SortType sort = kTime

6.52.1 Member Data Documentation

6.52.1.1 date `Date ticket::command::QueryTransfer::date`

6.52.1.2 from `std::string ticket::command::QueryTransfer::from`

6.52.1.3 sort `SortType ticket::command::QueryTransfer::sort = kTime`

6.52.1.4 to `std::string ticket::command::QueryTransfer::to`

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

- src/parser.h

6.53 ticket::command::RefundTicket Struct Reference

`#include <parser.h>`

Public Attributes

- std::string currentUser
- int index = 1

6.53.1 Member Data Documentation

6.53.1.1 currentUser `std::string ticket::command::RefundTicket::currentUser`

6.53.1.2 index `int ticket::command::RefundTicket::index = 1`

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

- src/parser.h

6.54 ticket::rollback::RefundTicket Struct Reference

`#include <rollback.h>`

Public Attributes

- int id
- Order::Status status

6.54.1 Member Data Documentation

6.54.1.1 id `int ticket::rollback::RefundTicket::id`

6.54.1.2 status `Order::Status ticket::rollback::RefundTicket::status`

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

- src/rollback.h

6.55 ticket::command::ReleaseTrain Struct Reference

`#include <parser.h>`

Public Attributes

- std::string id

6.55.1 Member Data Documentation

6.55.1.1 id `std::string ticket::command::ReleaseTrain::id`

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

- src/parser.h

6.56 ticket::rollback::ReleaseTrain Struct Reference

`#include <rollback.h>`

Public Attributes

- int id

6.56.1 Member Data Documentation

6.56.1.1 id `int ticket::rollback::ReleaseTrain::id`

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

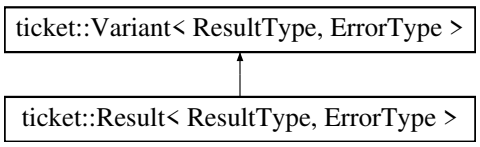
- src/rollback.h

6.57 ticket::Result< ResultType, ErrorType > Class Template Reference

Result<Res, Err> = Res | Err.

`#include <result.h>`

Inheritance diagram for ticket::Result< ResultType, ErrorType >:



Public Member Functions

- Result ()=delete
- template<typename T , typename = std::enable_if_t< std::is_constructible_v<ResultType, const T &> && !std::is_constructible_v<ErrorType, const T &> >>
  Result (const T &value)
- template<typename T , typename = std::enable_if_t< !std::is_constructible_v<ResultType, const T &> || std::is_same_v<ErrorType, T> >, typename = std::enable_if_t<std::is_constructible_v<ErrorType, const T &>>>
  Result (const T &value)
- auto result () -> ResultType &
- auto result () const -> const ResultType &
- auto error () -> ErrorType *
- auto error () const -> const ErrorType *
- auto success () const -> bool

  returns true if the result is in its successful state.

6.57.1 Detailed Description

template<typename ResultType, typename ErrorType>
class ticket::Result< ResultType, ErrorType >

Result<Res, Err> = Res | Err.

This class provides a wrapper around variant to make error handling a little easier. Recommended usage:

```
auto foo = doSomethingThatMightFail(args);
if (auto err = foo.error()) {
  // handles error, or rethrow:
  return *err;
}
std::cout << foo.result() << std::endl;
```

Therefore, result() returns a reference, while error() returns a pointer. This design is subject to change.

6.57.2 Constructor & Destructor Documentation

6.57.2.1 Result() [1/3] template<typename ResultType , typename ErrorType >

ticket::Result< ResultType, ErrorType >::Result ( ) [delete]

6.57.2.2 Result() [2/3] template<typename ResultType , typename ErrorType >

template<typename T , typename = std::enable_if_t< std::is_constructible_v<ResultType, const T &> && !std::is_constructible_v<ErrorType, const T &> >>
ticket::Result< ResultType, ErrorType >::Result (
const T & *value*) [inline]

6.57.2.3 Result() [3/3] template<typename ResultType , typename ErrorType >

template<typename T , typename = std::enable_if_t< !std::is_constructible_v<ResultType, const T &> || std::is_same_v<ErrorType, T> >, typename = std::enable_if_t<std::is_constructible_v<ErrorType, const T &>>>
ticket::Result< ResultType, ErrorType >::Result (
const T & *value*) [inline]

6.57.3 Member Function Documentation

6.57.3.1 error() [1/2] template<typename ResultType , typename ErrorType >

auto ticket::Result< ResultType, ErrorType >::error ( ) -> ErrorType * [inline]

6.57.3.2 error() [2/2] template<typename ResultType , typename ErrorType >

auto ticket::Result< ResultType, ErrorType >::error ( ) const -> const ErrorType * [inline]

6.57.3.3 result() [1/2] template<typename ResultType , typename ErrorType >

auto ticket::Result< ResultType, ErrorType >::result ( ) -> ResultType & [inline]

6.57.3.4 result() [2/2] template<typename ResultType , typename ErrorType >

auto ticket::Result< ResultType, ErrorType >::result ( ) const -> const ResultType & [inline]

6.57.3.5 success() template<typename ResultType , typename ErrorType >

auto ticket::Result< ResultType, ErrorType >::success ( ) const -> bool [inline]

returns true if the result is in its successful state.

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

- lib/result.h

6.58 ticket::Ride Struct Reference

#include <train.h>

Public Member Functions

- auto operator< (const Ride &rhs) const -> bool

Public Attributes

- int train

  the numerical id of the train.
- Date date

6.58.1 Member Function Documentation

6.58.1.1 operator<() auto ticket::Ride::operator< (const Ride & *rhs*) const -> bool

6.58.2 Member Data Documentation

6.58.2.1 date `Date ticket::Ride::date`

6.58.2.2 train `int ticket::Ride::train`

the numerical id of the train.

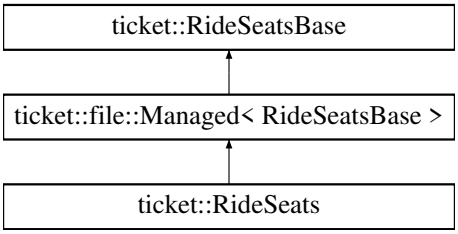
The documentation for this struct was generated from the following files:

- src/train.h
- src/train.cpp

6.59 ticket::RideSeats Struct Reference

`#include <train.h>`

Inheritance diagram for ticket::RideSeats:



Public Member Functions

- RideSeats ()=default
- RideSeats (const file::Managed< RideSeatsBase > &rideSeats)

Static Public Attributes

- static file::Index< Ride, RideSeats > ixRide {&RideSeats::ride, "ride-seats.ride.ix"}

Additional Inherited Members

6.59.1 Constructor & Destructor Documentation

6.59.1.1 RideSeats() [1/2] ticket::RideSeats::RideSeats ( ) [default]

6.59.1.2 RideSeats() [2/2] ticket::RideSeats::RideSeats (const file::Managed< RideSeatsBase > & *rideSeats*) [inline]

6.59.2 Member Data Documentation

6.59.2.1 ixRide file::Index< Ride, RideSeats > ticket::RideSeats::ixRide {&RideSeats::ride, "ride-seats.ride.ix"} [static]

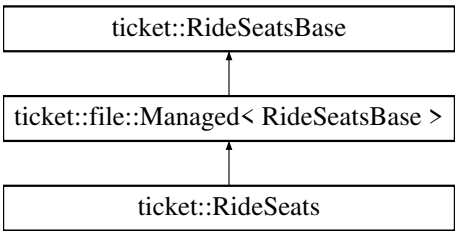
The documentation for this struct was generated from the following files:

- src/train.h
- src/train.cpp

6.60 ticket::RideSeatsBase Struct Reference

`#include <train.h>`

Inheritance diagram for ticket::RideSeatsBase:



Public Member Functions

- auto ticketsAvailable (int ixFrom, int ixTo) const -> int

  calculates how many tickets are still available.
- auto rangeAdd (int dx, int ixFrom, int ixTo) -> void

  adds dx to seatsRemaining[ixFrom, ixTo] inclusive.

Public Attributes

- Ride ride
- file::Array< int, 99 > seatsRemaining

Static Public Attributes

- static constexpr const char * filename = "ride-seats"

6.60.1 Member Function Documentation

6.60.1.1 rangeAdd()

```
auto ticket::RideSeatsBase::rangeAdd (
        int dx,
        int ixFrom,
        int ixTo ) -> void
```

adds dx to seatsRemaining[ixFrom, ixTo] inclusive.

6.60.1.2 ticketsAvailable()

```
auto ticket::RideSeatsBase::ticketsAvailable (
        int ixFrom,
        int ixTo ) const -> int
```

calculates how many tickets are still available.

Parameters

ixFrom	index of the departing stop
ixTo	index of the arriving stop

6.60.2 Member Data Documentation

6.60.2.1 filename

```
constexpr const char* ticket::RideSeatsBase::filename = "ride-seats"  [static], [constexpr]
```

6.60.2.2 ride

```
Ride ticket::RideSeatsBase::ride
```

6.60.2.3 seatsRemaining

```
file::Array<int, 99> ticket::RideSeatsBase::seatsRemaining
```

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

- src/train.h
- src/train.cpp

6.61 ticket::command::Rollback Struct Reference

`#include <parser.h>`

Public Attributes

- int timestamp

6.61.1 Member Data Documentation

6.61.1.1 timestamp `int ticket::command::Rollback::timestamp`

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

- src/parser.h

6.62 ticket::file::Set< T, maxLength, Cmp > Struct Template Reference

A sorted array with utility functions and bound checks.

`#include <set.h>`

Public Member Functions

- Set ()=default
- auto indexOfInsert (const T &element) -> size_t
- auto indexOf (const T &element) -> size_t
  finds the index of element in the set.
- auto includes (const T &element) -> bool
  checks if the elements is included in the set.
- auto insert (const T &element) -> void
  inserts the element into the set.
- auto remove (const T &element) -> void
  removes the element from the set.
- auto removeAt (size_t offset) -> void
  removes the element at offset.
- auto clear () -> void
  clears the set.
- void copyFrom (const Set &other, size_t ixFrom, size_t ixTo, size_t count)
  copies a portion of another set to this.
- auto operator[ ] (size_t index) -> T &
- auto operator[ ] (size_t index) const -> const T &
- auto pop () -> T
  pops the greatest element.
- auto shift () -> T
  pops the least element.
- template<typename Functor >
  auto forEach (const Functor &callback) -> void
  calls the callback for each element in the array.

Public Attributes

- size_t length = 0
- T content [maxLength]

6.62.1 Detailed Description

template<typename T, size_t maxLength, typename Cmp = Less<>>
struct ticket::file::Set< T, maxLength, Cmp >

A sorted array with utility functions and bound checks.

6.62.2 Constructor & Destructor Documentation

6.62.2.1 Set()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
ticket::file::Set< T, maxLength, Cmp >::Set ( )  [default]
```

6.62.3 Member Function Documentation

6.62.3.1 clear()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
auto ticket::file::Set< T, maxLength, Cmp >::clear ( ) -> void   [inline]
```

clears the set.

6.62.3.2 copyFrom()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
void ticket::file::Set< T, maxLength, Cmp >::copyFrom (
            const Set< T, maxLength, Cmp > & other,
            size_t ixFrom,
            size_t ixTo,
            size_t count )  [inline]
```

copies a portion of another set to this.

6.62.3.3 forEach()

```
template<typename T , size_t maxLength, typename Cmp = Less<>>
template<typename Functor >
auto ticket::file::Set< T, maxLength, Cmp >::forEach (
            const Functor & callback ) -> void   [inline]
```

calls the callback for each element in the array.

6.62.3.4 includes() template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::includes (
            const T & element ) -> bool   [inline]
```

checks if the elements is included in the set.

6.62.3.5 indexOf() template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::indexOf (
            const T & element ) -> size_t   [inline]
```

finds the index of element in the set.

6.62.3.6 indexOfInsert() template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::indexOfInsert (
            const T & element ) -> size_t   [inline]
```

6.62.3.7 insert() template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::insert (
            const T & element ) -> void   [inline]
```

inserts the element into the set.

6.62.3.8 operator[]() [1/2] template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::operator[] (
            size_t index ) -> T &   [inline]
```

6.62.3.9 operator[]() [2/2] template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::operator[] (
            size_t index ) const -> const T &   [inline]
```

6.62.3.10 pop() template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::pop ( ) -> T   [inline]
```

pops the greatest element.

6.62.3.11 remove() template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::remove (
            const T & element ) -> void   [inline]
```

removes the element from the set.

6.62.3.12 removeAt() template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::removeAt (
            size_t offset ) -> void   [inline]
```

removes the element at offset.

6.62.3.13 shift() template<typename T , size_t maxLength, typename Cmp = Less<>>

```
auto ticket::file::Set< T, maxLength, Cmp >::shift ( ) -> T   [inline]
```

pops the least element.

6.62.4 Member Data Documentation

6.62.4.1 content template<typename T , size_t maxLength, typename Cmp = Less<>>

```
T ticket::file::Set< T, maxLength, Cmp >::content[maxLength]
```

6.62.4.2 length template<typename T , size_t maxLength, typename Cmp = Less<>>

```
size_t ticket::file::Set< T, maxLength, Cmp >::length = 0
```

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

- lib/file/set.h

6.63 ticket::TrainBase::Stop Struct Reference

```
#include <train.h>
```

Public Attributes

- Station::Id name

6.63.1 Member Data Documentation

6.63.1.1 name `Station::Id ticket::TrainBase::Stop::name`

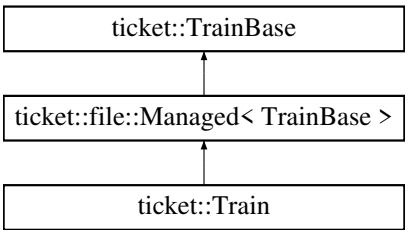
The documentation for this struct was generated from the following file:

- src/train.h

6.64 ticket::Train Struct Reference

`#include <train.h>`

Inheritance diagram for ticket::Train:



Public Member Functions

- Train ()=default
- Train (const file::Managed< TrainBase > &train)
- auto indexOfStop (const std::string &name) const -> Optional< int >

  finds the index of the station of the given name.
- auto totalPrice (int ixFrom, int ixTo) const -> int

  calculates the total price of a trip.
- auto getRide (Date date) const -> Optional< RideSeats >

  gets the remaining seats object on a given date.
- auto getRide (Date date, int ixDeparture) const -> Optional< RideSeats >

  gets the remaining seats object on a given date at a given stop.

Static Public Attributes

- static file::Index< Train::Id, Train > ixId {&Train::trainId, "trains.train-id.ix"}
- static file::BpTree< size_t, int > ixStop {"trains.stop.ix"}

Additional Inherited Members

6.64.1 Constructor & Destructor Documentation

6.64.1.1 Train() [1/2] ticket::Train::Train ( ) [default]

6.64.1.2 Train() [2/2] ticket::Train::Train (
const file::Managed< TrainBase > & *train*) [inline]

6.64.2 Member Function Documentation

6.64.2.1 getRide() [1/2] auto ticket::Train::getRide (
Date *date*) const -> Optional<RideSeats>

gets the remaining seats object on a given date.

Parameters

date	the departure date of the entire train (i.e. not the departure date of a stop).

6.64.2.2 getRide() [2/2] auto ticket::Train::getRide (
Date *date*,
int *ixDeparture*) const -> Optional<RideSeats>

gets the remaining seats object on a given date at a given stop.

Parameters

date	the departure date of a stop.
ixDeparture	the index of the departing stop.

6.64.2.3 indexOfStop() auto ticket::Train::indexOfStop (
const std::string & *name*) const -> Optional<int>

finds the index of the station of the given name.

6.64.2.4 totalPrice() auto ticket::Train::totalPrice (
int *ixFrom*,
int *ixTo*) const -> int

calculates the total price of a trip.

6.64.3 Member Data Documentation

6.64.3.1 ixId

`file::Index< Train::Id, Train > ticket::Train::ixId {&Train::trainId, "trains.↩ train-id.ix"} [static]`

6.64.3.2 ixStop

`file::BpTree< size_t, int > ticket::Train::ixStop {"trains.stop.ix"} [static]`

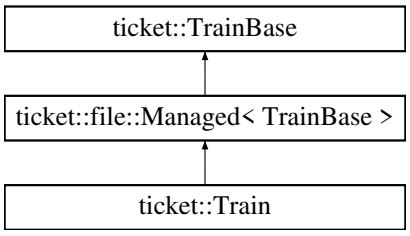
The documentation for this struct was generated from the following files:

- src/train.h
- src/train.cpp

6.65 ticket::TrainBase Struct Reference

`#include <train.h>`

Inheritance diagram for ticket::TrainBase:



Classes

- struct Edge
- struct Stop

Public Types

- using Id = file::Varchar< 20 >
- using Type = char

Public Attributes

- Id trainId
- file::Array< Stop, 100 > stops
- file::Array< Edge, 99 > edges
- int seats
- Date begin
- Date end
- Type type
- bool released = false
- bool deleted = false

Static Public Attributes

- static constexpr const char * filename = "trains"

6.65.1 Member Typedef Documentation

6.65.1.1 Id

```
using ticket::TrainBase::Id = file::Varchar<20>
```

6.65.1.2 Type

```
using ticket::TrainBase::Type = char
```

6.65.2 Member Data Documentation

6.65.2.1 begin

```
Date ticket::TrainBase::begin
```

6.65.2.2 deleted

```
bool ticket::TrainBase::deleted = false
```

6.65.2.3 edges

```
file::Array<Edge, 99> ticket::TrainBase::edges
```

6.65.2.4 end

```
Date ticket::TrainBase::end
```

6.65.2.5 filename

```
constexpr const char* ticket::TrainBase::filename = "trains"  [static], [constexpr]
```

6.65.2.6 released

```
bool ticket::TrainBase::released = false
```

6.65.2.7 seats `int ticket::TrainBase::seats`

6.65.2.8 stops `file::Array<Stop, 100> ticket::TrainBase::stops`

6.65.2.9 trainId `Id ticket::TrainBase::trainId`

6.65.2.10 type `Type ticket::TrainBase::type`

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

- src/train.h

6.66 ticket::Triple< T1, T2, T3 > Class Template Reference

A triplet of objects.

`#include <utility.h>`

Public Member Functions

- constexpr Triple ()
- Triple (const Triple &other)=default
- Triple (Triple &&other) noexcept=default
- Triple (const T1 &x, const T2 &y, const T3 &z)

Public Attributes

- T1 first
- T2 second
- T3 third

6.66.1 Detailed Description

template<typename T1, typename T2, typename T3>
class ticket::Triple< T1, T2, T3 >

A triplet of objects.

6.66.2 Constructor & Destructor Documentation

6.66.2.1 Triple() [1/4] template<typename T1 , typename T2 , typename T3 >

constexpr ticket::Triple< T1, T2, T3 >::Triple ( ) [inline], [constexpr]

6.66.2.2 Triple() [2/4] template<typename T1 , typename T2 , typename T3 >

ticket::Triple< T1, T2, T3 >::Triple (
const Triple< T1, T2, T3 > & *other*) [default]

6.66.2.3 Triple() [3/4] template<typename T1 , typename T2 , typename T3 >

ticket::Triple< T1, T2, T3 >::Triple (
Triple< T1, T2, T3 > && *other*) [default], [noexcept]

6.66.2.4 Triple() [4/4] template<typename T1 , typename T2 , typename T3 >

ticket::Triple< T1, T2, T3 >::Triple (
const T1 & *x*,
const T2 & *y*,
const T3 & *z*) [inline]

6.66.3 Member Data Documentation

6.66.3.1 first template<typename T1 , typename T2 , typename T3 >

T1 ticket::Triple< T1, T2, T3 >::first

6.66.3.2 second template<typename T1 , typename T2 , typename T3 >

T2 ticket::Triple< T1, T2, T3 >::second

6.66.3.3 third template<typename T1 , typename T2 , typename T3 >

T3 ticket::Triple< T1, T2, T3 >::third

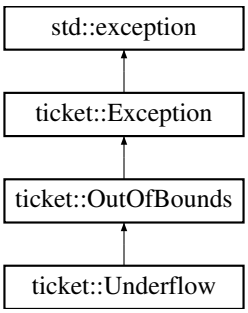
The documentation for this class was generated from the following file:

- lib/utility.h

6.67 ticket::Underflow Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::Underflow:



Public Member Functions

- Underflow ()
- Underflow (const char *what)

6.67.1 Constructor & Destructor Documentation

6.67.1.1 Underflow() [1/2] `ticket::Underflow::Underflow ( ) [inline]`

6.67.1.2 Underflow() [2/2] `ticket::Underflow::Underflow ( const char * what ) [inline]`

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

- lib/exception.h

6.68 ticket::Unit Struct Reference

An empty class, used at various places.

`#include <utility.h>`

Public Member Functions

- constexpr Unit ()=default
- template<typename T >
  constexpr Unit (const T &)
- auto operator< (const Unit &) -> bool

6.68.1 Detailed Description

An empty class, used at various places.

6.68.2 Constructor & Destructor Documentation

6.68.2.1 Unit() [1/2] constexpr ticket::Unit::Unit ( ) [constexpr], [default]

6.68.2.2 Unit() [2/2] template<typename T >

```
constexpr ticket::Unit::Unit (
            const T &  ) [inline], [constexpr]
```

6.68.3 Member Function Documentation

6.68.3.1 operator<() auto ticket::Unit::operator< (

```
            const Unit &  ) -> bool   [inline]
```

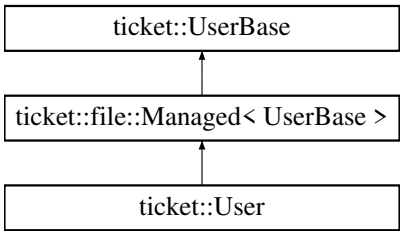
The documentation for this struct was generated from the following file:

- lib/utility.h

6.69 ticket::User Struct Reference

```
#include <user.h>
```

Inheritance diagram for ticket::User:



Public Member Functions

- User ()=default
- User (const file::Managed< UserBase > &user)

Static Public Attributes

- static file::Index< User::Id, User > ixUsername {&User::username, "users.username.ix"}

Additional Inherited Members

6.69.1 Constructor & Destructor Documentation

6.69.1.1 User() [1/2] ticket::User::User ( ) [default]

6.69.1.2 User() [2/2] ticket::User::User (const file::Managed< UserBase > & *user*) [inline]

6.69.2 Member Data Documentation

6.69.2.1 ixUsername file::Index< User::Id, User > ticket::User::ixUsername {&User::username, "users.username.ix"} [static]

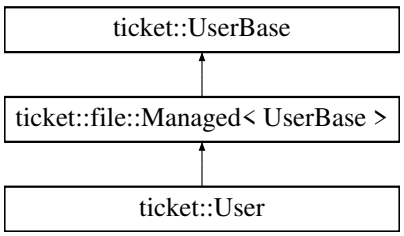
The documentation for this struct was generated from the following files:

- src/user.h
- src/user.cpp

6.70 ticket::UserBase Struct Reference

`#include <user.h>`

Inheritance diagram for ticket::UserBase:



Public Types

- using Id = file::Varchar< 20 >
- using Password = file::Varchar< 30 >
- using Name = file::Varchar< 15 >
- using Email = file::Varchar< 30 >
- using Privilege = int

Static Public Member Functions

- static auto has (const char *username) -> bool

  checks if there is a user with the given username.
- static auto isLoggedIn (const std::string &username) -> bool

  checks if the user is logged in.

Public Attributes

- Id username
- Password password
- Name name
- Email email
- Privilege privilege

Static Public Attributes

- static constexpr const char * filename = "users"

6.70.1 Member Typedef Documentation

6.70.1.1 Email `using ticket::UserBase::Email = file::Varchar<30>`

6.70.1.2 Id `using ticket::UserBase::Id = file::Varchar<20>`

6.70.1.3 Name `using ticket::UserBase::Name = file::Varchar<15>`

6.70.1.4 Password `using ticket::UserBase::Password = file::Varchar<30>`

6.70.1.5 Privilege `using ticket::UserBase::Privilege = int`

6.70.2 Member Function Documentation

6.70.2.1 has() `auto ticket::UserBase::has ( const char * username ) -> bool [static]`

checks if there is a user with the given username.

6.70.2.2 isLoggedIn() `auto ticket::UserBase::isLoggedIn ( const std::string & username ) -> bool [static]`

checks if the user is logged in.

6.70.3 Member Data Documentation

6.70.3.1 email `Email ticket::UserBase::email`

6.70.3.2 filename `constexpr const char* ticket::UserBase::filename = "users" [static], [constexpr]`

6.70.3.3 name `Name ticket::UserBase::name`

6.70.3.4 password `Password ticket::UserBase::password`

6.70.3.5 privilege `Privilege ticket::UserBase::privilege`

6.70.3.6 username `Id ticket::UserBase::username`

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

- src/user.h
- src/user.cpp

6.71 ticket::file::Varchar< maxLength > Struct Template Reference

A wrapper for const char * with utility functions and type conversions.

`#include <varchar.h>`

Public Member Functions

- Varchar ()
- Varchar (const std::string &s)
- Varchar (const char *cstr)
- template<int A>
  Varchar (const Varchar< A > &that)
- operator std::string () const
- auto str () const -> std::string
- auto length () const -> int
- template<int A>
  auto operator= (const Varchar< A > &that) -> Varchar &
- template<int A>
  auto operator< (const Varchar< A > &that) const -> bool
- template<int A>
  auto operator== (const Varchar< A > &that) const -> bool
- template<int A>
  auto operator!= (const Varchar< A > &that) const -> bool
- auto hash () const -> size_t

Static Public Attributes

- static constexpr int kMaxLength = maxLength

Friends

- template<int A>
  class Varchar

6.71.1 Detailed Description

template<int maxLength>
struct ticket::file::Varchar< maxLength >

A wrapper for const char * with utility functions and type conversions.

the trailing zero is not counted in maxLength.

its default ordering is hash order. this is for a maximum performance. you need to write a comparator if you want dictionary order.

6.71.2 Constructor & Destructor Documentation

6.71.2.1 Varchar() [1/4] template<int maxLength>

```
ticket::file::Varchar< maxLength >::Varchar ( )  [inline]
```

6.71.2.2 Varchar() [2/4] template<int maxLength>

```
ticket::file::Varchar< maxLength >::Varchar (
            const std::string & s )  [inline]
```

6.71.2.3 Varchar() [3/4] template<int maxLength>

```
ticket::file::Varchar< maxLength >::Varchar (
            const char * cstr )  [inline]
```

6.71.2.4 Varchar() [4/4] template<int maxLength>

```
template<int A>
ticket::file::Varchar< maxLength >::Varchar (
            const Varchar< A > & that )  [inline]
```

6.71.3 Member Function Documentation

6.71.3.1 hash() template<int maxLength>

```
auto ticket::file::Varchar< maxLength >::hash ( ) const -> size_t   [inline]
```

6.71.3.2 length() template<int maxLength>

```
auto ticket::file::Varchar< maxLength >::length ( ) const -> int   [inline]
```

6.71.3.3 operator std::string() template<int maxLength>

```
ticket::file::Varchar< maxLength >::operator std::string ( ) const  [inline]
```

6.71.3.4 operator"!=() template<int maxLength>

```
template<int A>
auto ticket::file::Varchar< maxLength >::operator!= (
            const Varchar< A > & that ) const -> bool   [inline]
```

6.71.3.5 operator<() template<int maxLength>

```
template<int A>
auto ticket::file::Varchar< maxLength >::operator< (
            const Varchar< A > & that ) const -> bool   [inline]
```

6.71.3.6 operator=() template<int maxLength>

```
template<int A>
auto ticket::file::Varchar< maxLength >::operator= (
            const Varchar< A > & that ) -> Varchar &   [inline]
```

6.71.3.7 operator==() template<int maxLength>

```
template<int A>
auto ticket::file::Varchar< maxLength >::operator== (
            const Varchar< A > & that ) const -> bool   [inline]
```

6.71.3.8 str() template<int maxLength>

```
auto ticket::file::Varchar< maxLength >::str ( ) const -> std::string   [inline]
```

6.71.4 Friends And Related Function Documentation

6.71.4.1 Varchar template<int maxLength>

```
template<int A>
friend class Varchar  [friend]
```

6.71.5 Member Data Documentation

6.71.5.1 kMaxLength template<int maxLength>

```
constexpr int ticket::file::Varchar< maxLength >::kMaxLength = maxLength  [static], [constexpr]
```

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

- lib/file/varchar.h

6.72 ticket::Variant< Ts > Class Template Reference

A tagged union, aka sum type.

`#include <variant.h>`

Public Member Functions

- Variant ()
- template<typename T , int ix = Traits::template indexOf<T>()>
  Variant (const T &value)
- Variant (const Variant &other)
- Variant (Variant &&other) noexcept
- virtual ~Variant ()
- auto operator= (const Variant &other) -> Variant &
- auto operator= (Variant &&other) noexcept -> Variant &
- template<typename T , int ix = Traits::template indexOf<T>()>
  auto operator= (const T &value) -> Variant &

  sets the variant to one of its member types.
- template<typename T >
  auto is () const -> bool

  checks if T is the current type of this variant.
- auto index () const -> int

  returns the current index of the current state.
- template<typename T >
  auto get () -> T *

  if the current state is of type T, return it. else null.
- template<typename T >
  auto get () const -> const T *

  if the current state is of type T, return it. else null.
- template<int ix>
  auto get () -> typename Traits::template NthType< ix > *

  if the current state is of index ix, return it. else null.
- template<int ix>
  auto get () const -> const typename Traits::template NthType< ix > *

  if the current state is of index ix, return it. else null.
- template<typename Visitor >
  auto visit (const Visitor &f) const -> void

  visits the variant using a polymorphic functor.

6.72.1 Detailed Description

template<typename ... Ts>
class ticket::Variant< Ts >

A tagged union, aka sum type.

This object holds exactly one of its member types, but which type it holds is not statically known. It is entirely on stack, no extra memory allocations are made.

Member types need to be unique and not overlapping.

6.72.2 Constructor & Destructor Documentation

6.72.2.1 Variant() [1/4] template<typename ... Ts>

```
ticket::Variant< Ts >::Variant ( )  [inline]
```

6.72.2.2 Variant() [2/4] template<typename ... Ts>

```
template<typename T , int ix = Traits::template indexOf<T>()>
ticket::Variant< Ts >::Variant (
            const T & value )  [inline]
```

constructs the variant from one of its member types.

6.72.2.3 Variant() [3/4] template<typename ... Ts>

```
ticket::Variant< Ts >::Variant (
            const Variant< Ts > & other )  [inline]
```

6.72.2.4 Variant() [4/4] template<typename ... Ts>

```
ticket::Variant< Ts >::Variant (
            Variant< Ts > && other )  [inline], [noexcept]
```

6.72.2.5 ~Variant() template<typename ... Ts>

```
virtual ticket::Variant< Ts >::~Variant ( )  [inline], [virtual]
```

6.72.3 Member Function Documentation

6.72.3.1 get() [1/4] template<typename ... Ts>

```
template<typename T >
auto ticket::Variant< Ts >::get ( ) -> T *  [inline]
```

if the current state is of type T, return it. else null.

6.72.3.2 get() [2/4] template<typename ... Ts>

```
template<int ix>
auto ticket::Variant< Ts >::get ( ) -> typename Traits::template NthType<ix> *  [inline]
```

if the current state is of index ix, return it. else null.

6.72.3.3 get() [3/4] template<typename ... Ts>

```
template<typename T >
auto ticket::Variant< Ts >::get ( ) const -> const T *   [inline]
```

if the current state is of type T, return it. else null.

6.72.3.4 get() [4/4] template<typename ... Ts>

```
template<int ix>
auto ticket::Variant< Ts >::get ( ) const -> const typename Traits::template NthType<ix> *
[inline]
```

if the current state is of index ix, return it. else null.

6.72.3.5 index() template<typename ... Ts>

```
auto ticket::Variant< Ts >::index ( ) const -> int   [inline]
```

returns the current index of the current state.

6.72.3.6 is() template<typename ... Ts>

```
template<typename T >
auto ticket::Variant< Ts >::is ( ) const -> bool   [inline]
```

checks if T is the current type of this variant.

6.72.3.7 operator=() [1/3] template<typename ... Ts>

```
template<typename T , int ix = Traits::template indexOf<T>()>
auto ticket::Variant< Ts >::operator= (
            const T & value ) -> Variant &   [inline]
```

sets the variant to one of its member types.

6.72.3.8 operator=() [2/3] template<typename ... Ts>

```
auto ticket::Variant< Ts >::operator= (
            const Variant< Ts > & other ) -> Variant &   [inline]
```

6.72.3.9 operator=() [3/3] template<typename ... Ts>

```
auto ticket::Variant< Ts >::operator= (
            Variant< Ts > && other ) -> Variant &   [inline], [noexcept]
```

6.72.3.10 visit() template<typename ... Ts> template<typename Visitor > auto ticket::Variant< Ts >::visit (const Visitor & f) const -> void [inline]

visits the variant using a polymorphic functor.

pass in a polymorphic visitor function, and we will call it with the correct type. If the current type is T, then we would call f(T &). Note that this method deliberately disregards const status. This is to ensure that it still works when this is const.

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

- lib/variant.h

6.73 ticket::Vector< T > Class Template Reference

A data container like std::vector.

`#include <vector.h>`

Classes

- class const_iterator
- class iterator

Public Member Functions

- Vector ()=default
- Vector (const Vector &other)
- Vector (Vector &&other) noexcept
- ~Vector ()
- auto operator= (const Vector &other) -> Vector &
- auto operator= (Vector &&other) noexcept -> Vector &
- auto at (const size_t &pos) -> T &
- auto at (const size_t &pos) const -> const T &
- auto operator[ ] (const size_t &pos) -> T &
- auto operator[ ] (const size_t &pos) const -> const T &
- auto front () const -> const T &
- auto back () const -> const T &
- auto begin () -> iterator
- auto begin () const -> const_iterator
- auto cbegin () const -> const_iterator
- auto end () -> iterator
- auto end () const -> const_iterator
- auto cend () const -> const_iterator
- auto empty () const -> bool
- auto size () const -> size_t
- auto clear () -> void
- auto insert (iterator pos, const T &value) -> iterator
- auto insert (const size_t &ix, const T &value) -> iterator
- auto erase (iterator pos) -> iterator

- auto erase (const size_t &ix) -> iterator
- auto push_back (const T &value) -> void
- auto pop_back () -> void
- auto reserve (size_t capacity) -> void
- template<typename Functor >
  auto map (const Functor &fn) const -> Vector< decltype(fn(at(0)))>
- template<typename Functor >
  auto reduce (const Functor &fn) const -> T
- template<typename Functor , typename Res >
  auto reduce (const Functor &fn, const Res &init) const -> Res

6.73.1 Detailed Description

template<typename T>
class ticket::Vector< T >

A data container like std::vector.

store data in a successive memory and support random access.

6.73.2 Constructor & Destructor Documentation

6.73.2.1 Vector() [1/3] template<typename T >

```
ticket::Vector< T >::Vector ( )  [default]
```

6.73.2.2 Vector() [2/3] template<typename T >

```
ticket::Vector< T >::Vector (
            const Vector< T > & other )  [inline]
```

6.73.2.3 Vector() [3/3] template<typename T >

```
ticket::Vector< T >::Vector (
            Vector< T > && other )  [inline], [noexcept]
```

6.73.2.4 ~Vector() template<typename T >

```
ticket::Vector< T >::~Vector ( )  [inline]
```

6.73.3 Member Function Documentation

6.73.3.1 at() [1/2] template<typename T >

```
auto ticket::Vector< T >::at (
            const size_t & pos ) -> T &   [inline]
```

assigns specified element with bounds checking throw index_out_of_bound if pos is not in [0, size)

6.73.3.2 at() [2/2] template<typename T >

```
auto ticket::Vector< T >::at (
            const size_t & pos ) const -> const T &   [inline]
```

6.73.3.3 back() template<typename T >

```
auto ticket::Vector< T >::back ( ) const -> const T &   [inline]
```

access the last element. throw container_is_empty if size == 0

6.73.3.4 begin() [1/2] template<typename T >

```
auto ticket::Vector< T >::begin ( ) -> iterator   [inline]
```

returns an iterator to the beginning.

6.73.3.5 begin() [2/2] template<typename T >

```
auto ticket::Vector< T >::begin ( ) const -> const_iterator   [inline]
```

6.73.3.6 cbegin() template<typename T >

```
auto ticket::Vector< T >::cbegin ( ) const -> const_iterator   [inline]
```

6.73.3.7 cend() template<typename T >

```
auto ticket::Vector< T >::cend ( ) const -> const_iterator   [inline]
```

6.73.3.8 clear() template<typename T >

```
auto ticket::Vector< T >::clear ( ) -> void   [inline]
```

clears the contents

6.73.3.9 empty() template<typename T >

```
auto ticket::Vector< T >::empty ( ) const -> bool   [inline]
```

checks whether the container is empty

6.73.3.10 end() [1/2] template<typename T >

```
auto ticket::Vector< T >::end ( ) -> iterator   [inline]
```

returns an iterator to the end.

6.73.3.11 end() [2/2] template<typename T >

```
auto ticket::Vector< T >::end ( ) const -> const_iterator   [inline]
```

6.73.3.12 erase() [1/2] template<typename T >

```
auto ticket::Vector< T >::erase (
            const size_t & ix ) -> iterator   [inline]
```

removes the element with index ind. return an iterator pointing to the following element. throw index_out_of_bound if ind >= size

6.73.3.13 erase() [2/2] template<typename T >

```
auto ticket::Vector< T >::erase (
            iterator pos ) -> iterator   [inline]
```

removes the element at pos. return an iterator pointing to the following element. If the iterator pos refers the last element, the end() iterator is returned.

6.73.3.14 front() template<typename T >

```
auto ticket::Vector< T >::front ( ) const -> const T &   [inline]
```

access the first element. throw container_is_empty if size == 0

6.73.3.15 insert() [1/2] template<typename T >

```
auto ticket::Vector< T >::insert (
            const size_t & ix,
            const T & value ) -> iterator   [inline]
```

inserts value at index ind. after inserting, this->at(ind) == value returns an iterator pointing to the inserted value. throw index_out_of_bound if ind > size (in this situation ind can be size because after inserting the size will increase 1.)

6.73.3.16 insert() [2/2] template<typename T >

```
auto ticket::Vector< T >::insert (
            iterator pos,
            const T & value ) -> iterator   [inline]
```

inserts value before pos returns an iterator pointing to the inserted value.

6.73.3.17 map() template<typename T >

```
template<typename Functor >
auto ticket::Vector< T >::map (
            const Functor & fn ) const -> Vector<decltype(fn(at(0)))>   [inline]
```

6.73.3.18 operator=() [1/2] template<typename T >

```
auto ticket::Vector< T >::operator= (
            const Vector< T > & other ) -> Vector &   [inline]
```

6.73.3.19 operator=() [2/2] template<typename T >

```
auto ticket::Vector< T >::operator= (
            Vector< T > && other ) -> Vector &   [inline], [noexcept]
```

6.73.3.20 operator[]() [1/2] template<typename T >

```
auto ticket::Vector< T >::operator[] (
            const size_t & pos ) -> T &   [inline]
```

assigns specified element with bounds checking throw index_out_of_bound if pos is not in [0, size) !!! Pay attentions In STL this operator does not check the boundary but I want you to do.

6.73.3.21 operator[]() [2/2] template<typename T >

```
auto ticket::Vector< T >::operator[] (
            const size_t & pos ) const -> const T &   [inline]
```

6.73.3.22 pop_back() template<typename T >

```
auto ticket::Vector< T >::pop_back ( ) -> void   [inline]
```

remove the last element from the end. throw container_is_empty if size() == 0

6.73.3.23 push_back() template<typename T >

```
auto ticket::Vector< T >::push_back (
            const T & value ) -> void   [inline]
```

adds an element to the end.

6.73.3.24 reduce() [1/2] template<typename T >

```
template<typename Functor >
auto ticket::Vector< T >::reduce (
            const Functor & fn ) const -> T   [inline]
```

6.73.3.25 reduce() [2/2] template<typename T >

```
template<typename Functor , typename Res >
auto ticket::Vector< T >::reduce (
            const Functor & fn,
            const Res & init ) const -> Res   [inline]
```

6.73.3.26 reserve() template<typename T >

```
auto ticket::Vector< T >::reserve (
            size_t capacity ) -> void   [inline]
```

6.73.3.27 size() template<typename T >

```
auto ticket::Vector< T >::size ( ) const -> size_t   [inline]
```

returns the number of elements

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

- lib/vector.h

7 File Documentation

7.1 lib/algorithm.h File Reference

```
#include <iostream>
#include <algorithm>
#include "utility.h"
```

Namespaces

- namespace ticket

Macros

- #define TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(name, cf)

Functions

- template<typename Iterator , class Compare = Less<>>
  auto ticket::sort (Iterator first, Iterator last, Compare cmp={}) -> void

  sorts the elements between first and last.

7.1.1 Macro Definition Documentation

7.1.1.1 TICKET_ALGORIGHM_DEFINE_BOUND_FUNC

```
#define TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(
        name,
        cf )
```

Value:

```
template<class Iterator, class T, class Compare = Less<» \
auto name (Iterator first, Iterator last, const T &value, Compare cmp = {}) \
  -> Iterator { \
  int length = distance(first, last); \
  while (length != 0) { \
    auto it = first; \
    int mid = length / 2; \
    advance(it, mid); \
    if (cmp.cf(value, *it)) { \
      first = ++it; \
      length -= mid + 1; \
    } else { \
      length = mid; \
    } \
  } \
  return first; \
}
```

7.2 algorithm.h

Go to the documentation of this file.

```
1 // This file includes some common algorithms.
2 #ifndef TICKET_LIB_ALGORITHM_H_
3 #define TICKET_LIB_ALGORITHM_H_
4
5 #include <iostream>
6 #ifndef ONLINE_JUDGE
7 #include <algorithm>
8 #endif // ONLINE_JUDGE
9
10 #include "utility.h"
11
12 namespace ticket {
13
14 using std::distance, std::advance;
15
16 #define TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(name, cf) \
17 template<class Iterator, class T, class Compare = Less<» \
18 auto name (Iterator first, Iterator last, const T &value, Compare cmp = {}) \
19   -> Iterator { \
20   int length = distance(first, last); \
21   while (length != 0) { \
22     auto it = first; \
23     int mid = length / 2; \
24     advance(it, mid); \
25     if (cmp.cf(value, *it)) { \
26       first = ++it; \
27       length -= mid + 1; \
28     } else { \
29       length = mid; \
30     } \
31   } \
32   return first; \
33 }
34 TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(upperBound, geq)
35 TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(lowerBound, gt)
36 #undef TICKET_ALGORIGHM_DEFINE_BOUND_FUNC
37
39 template <typename Iterator, class Compare = Less<»
40 auto sort (Iterator first, Iterator last, Compare cmp = {})
41   -> void {
42   auto distance = std::distance(first, last);
43   if (distance <= 1) return;
44   auto mid = first;
45   std::advance(mid, distance / 2);
46   sort(first, mid, cmp);
47   sort(mid, last, cmp);
48   std::remove_cvref_t<decltype(*first)> tmp[distance + 1];
49   int s = 0;
50   auto p = first;
51   auto q = mid;
52   while (s < distance) {
53     if (p != mid && (q == last || cmp.lt(*p, *q))) {
54       tmp[s++] = *p++;
55     } else {
```

```
56       tmp[s++] = *q++;
57     }
58   }
59   int i = 0;
60   while (first != last) *first++ = tmp[i++];
61   TICKET_ASSERT(i == distance);
62 }
63
64 } // namespace ticket
65
66 #endif // TICKET_LIB_ALGORITHM_H_
```

7.3 lib/datetime.cpp File Reference

```
#include "datetime.h"
#include "utility.h"
```

Namespaces

- namespace ticket

Functions

- auto ticket::formatDateTime (Date date, Instant instant) -> std::string

7.4 lib/datetime.h File Reference

```
#include <iostream>
```

Classes

- class ticket::Date

  Class representing a date between 2021-06-01 and 2021-08-31 (inclusive).
- class ticket::Duration

  Class representing a length of timespan.
- class ticket::Instant

  Class representing a point of time in a day.

Namespaces

- namespace ticket

Functions

- auto ticket::formatDateTime (Date date, Instant instant) -> std::string

7.5 datetime.h

Go to the documentation of this file.

```
1 // This file includes date and time utilities.
2 #ifndef TICKET_LIB_DATETIME_H_
3 #define TICKET_LIB_DATETIME_H_
4
5 #include <iostream>
6
7 namespace ticket {
8
13 class Date {
14  public:
15   Date () = default;
16   Date (int month, int date);
23   explicit Date (const char *str);
25   auto month () const -> int;
27   auto date () const -> int;
29   operator std::string () const;
34   auto operator+ (int dt) const -> Date;
39   auto operator- (int dt) const -> Date;
44   auto operator- (Date rhs) const -> int;
45   auto operator< (const Date &rhs) const -> bool;
47   auto inRange (Date begin, Date end) const -> bool;
48  private:
49   explicit Date (int days) : days_(days) {}
50   int days_ = 0;
51 };
52
63 class Duration {
64  public:
65   Duration () = default;
66   explicit Duration (int minutes) : minutes_(minutes) {}
68   auto minutes () const -> int;
69   auto operator+ (Duration dt) const -> Duration;
70   auto operator- (Duration dt) const -> Duration;
72   auto operator- () const -> Duration;
73   auto operator< (const Duration &rhs) const -> bool;
74  private:
75   int minutes_ = 0;
76 };
77
86 class Instant {
87  public:
88   Instant () = default;
89   Instant (int hour, int minute);
91   explicit Instant (const char *str);
92   auto daysOverflow () const -> int;
93   auto hour () const -> int;
94   auto minute () const -> int;
96   operator std::string () const;
97   auto operator+ (Duration dt) const -> Instant;
98   auto operator- (Duration dt) const -> Instant;
99   auto operator- (Instant rhs) const -> Duration;
100   auto operator< (const Instant &rhs) const -> bool;
101  private:
102   explicit Instant (int minutes) : minutes_(minutes) {}
103   int minutes_ = 0;
104 };
105
106 auto formatDateTime (Date date, Instant instant)
107   -> std::string;
108
109 } // namespace ticket
110
111 #endif // TICKET_LIB_DATETIME_H_
```

7.6 lib/exception.h File Reference

```
#include <iostream>
```

Classes

- class ticket::Exception

The base exception class.

- class ticket::IoException
- class ticket::OutOfBounds
- class ticket::Overflow
- class ticket::Underflow
- class ticket::NotFound
- class ticket::ParseException

Namespaces

- namespace ticket

7.7 exception.h

Go to the documentation of this file.

```
1
6 #ifndef TICKET_LIB_EXCEPTION_H_
7 #define TICKET_LIB_EXCEPTION_H_
8
9 #include <iostream>
10
11 namespace ticket {
12
14 class Exception : public std::exception {
15  public:
16   Exception () = default;
17   Exception (const char *what) : what_(what) {}
18   ~Exception () override = default;
20   virtual auto what () const noexcept -> const char * {
21     return what_;
22   }
23  private:
24   const char * const what_ = "unknown exception";
25 };
26
27 class IoException : public Exception {
28  public:
29   IoException () : Exception("IO exception") {}
30   IoException (const char *what) : Exception(what) {}
31 };
32
33 class OutOfBounds : public Exception {
34  public:
35   OutOfBounds () : Exception("out of bounds") {}
36   OutOfBounds (const char *what) : Exception(what) {}
37 };
38
39 class Overflow : public OutOfBounds {
40  public:
41   Overflow () : OutOfBounds("overflow") {}
42   Overflow (const char *what) : OutOfBounds(what) {}
43 };
44
45 class Underflow : public OutOfBounds {
46  public:
47   Underflow () : OutOfBounds("underflow") {}
48   Underflow (const char *what) : OutOfBounds(what) {}
49 };
50
51 class NotFound : public Exception {
52  public:
53   NotFound () : Exception("underflow") {}
54   NotFound (const char *what) : Exception(what) {}
55 };
56
57 class ParseException : public Exception {
58  public:
59   ParseException () : Exception("parse exception") {}
60   ParseException (const char *what) : Exception(what) {}
61 };
62
63 } // namespace ticket
64
65 #endif // TICKET_LIB_EXCEPTION_H_
```

7.8 lib/file/array.h File Reference

```
#include <cstring>
#include "exception.h"
#include "utility.h"
```

Classes

- struct ticket::file::Array< T, maxLength, Cmp >

  An on-stack array with utility functions and bound checks.

Namespaces

- namespace ticket
- namespace ticket::file

  File utilities.

7.9 array.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_FILE_ARRAY_H_
2 #define TICKET_LIB_FILE_ARRAY_H_
3
4 #include <cstring>
5
6 #include "exception.h"
7 #include "utility.h"
8
9 namespace ticket::file {
10
17 template <typename T, size_t maxLength, typename Cmp = Less<»
18 struct Array {
19  private:
20   auto boundsCheck_ (size_t index) const -> void {
21     if (index >= length) throw OutOfBounds("Array: overflow or underflow");
22   }
23   Cmp cmp_;
24  public:
25   size_t length = 0;
26   T content[maxLength];
28   auto indexOf (const T &element) -> size_t {
29     for (size_t i = 0; i < length; ++i) {
30       if (cmp_.equals(element, content[i])) return i;
31     }
32     throw NotFound("Array::indexOf: element not found");
33   }
35   auto includes (const T &element) -> bool {
36     for (size_t i = 0; i < length; ++i) {
37       if (cmp_.equals(element, content[i])) return true;
38     }
39     return false;
40   }
45   auto insert (const T &element, size_t offset) -> void {
46     if (offset != length) boundsCheck_(offset);
47     if (length == maxLength) {
48       throw Overflow("Array::insert: overflow");
49     }
50     if (offset != length) {
51       memmove(
52         &content[offset + 1],
53         &content[offset],
54         (length - offset) * sizeof(content[0])
55       );
56     }
57     content[offset] = element;
58     ++length;
59   }
60
```

```
62   auto remove (const T &element) -> void {
63     removeAt(indexOf(element));
64   }
69   auto removeAt (size_t offset) -> void {
70     boundsCheck_(offset);
71     if (offset != length - 1) {
72       memmove(
73         &content[offset],
74         &content[offset + 1],
75         (length - offset - 1) * sizeof(content[0])
76       );
77     }
78     --length;
79   }
81   auto clear () -> void { length = 0; }
82
84   auto copyFrom (
85     const Array &other,
86     size_t ixFrom,
87     size_t ixTo,
88     size_t count
89   ) -> void {
90     if (this == &other) {
91       memmove(
92         &content[ixTo],
93         &content[ixFrom],
94         count * sizeof(content[0])
95       );
96     } else {
97       memcpy(
98         &content[ixTo],
99         &other.content[ixFrom],
100         count * sizeof(content[0])
101       );
102     }
103   }
104
105   auto operator[] (size_t index) -> T & {
106     boundsCheck_(index);
107     return content[index];
108   }
109   auto operator[] (size_t index) const -> const T & {
110     boundsCheck_(index);
111     return content[index];
112   }
113
115   auto pop () -> T {
116     if (length == 0) throw Underflow("Array::pop: underflow");
117     return content[--length];
118   }
120   auto shift () -> T {
121     if (length == 0) throw Underflow("Array::pop: underflow");
122     T result = content[0];
123     removeAt(0);
124     return result;
125   }
127   auto push (const T &object) -> void { insert(object, length); }
129   auto unshift (const T &object) -> void { insert(object, 0); }
130
132   template <typename Functor>
133   auto forEach (const Functor &callback) -> T {
134     for (size_t i = 0; i < length; ++i) callback(content[i]);
135   }
136 };
137
138 } // namespace ticket::file
139
140 #endif // TICKET_LIB_FILE_ARRAY_H_
```

7.10 lib/file/bptree.h File Reference

```
#include <cstring>
#include "algorithm.h"
#include "file/array.h"
#include "file/file.h"
#include "file/internal/file.h"
#include "file/set.h"
#include "optional.h"
#include "utility.h"
```

```
#include "vector.h"
```

Classes

- class ticket::file::BpTree< KeyType, ValueType, CmpKey, CmpValue, Meta, szChunk >

  an implementation of the B+ tree.

Namespaces

- namespace ticket
- namespace ticket::file

  File utilities.

7.11 bptree.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_FILE_BPTREE_H_
2 #define TICKET_LIB_FILE_BPTREE_H_
3
4 #include <cstring>
5
6 #include "algorithm.h"
7 #include "file/array.h"
8 #include "file/file.h"
9 #include "file/internal/file.h"
10 #include "file/set.h"
11 #include "optional.h"
12 #include "utility.h"
13 #include "vector.h"
14
15 #ifdef TICKET_DEBUG
16 #include <iostream>
17 #endif
18
19 namespace ticket::file {
20
29 template <
30   typename KeyType,
31   typename ValueType,
32   typename CmpKey = Less<>,
33   typename CmpValue = Less<>,
34   typename Meta = Unit,
35   size_t szChunk = kDefaultSzChunk
36 >
37 class BpTree {
38  private:
39   struct Node;
40  public:
42   BpTree (const char *filename) : file_(filename, [this] () { this->init_(); }) {}
50   auto insert (const KeyType &key, const ValueType &value) -> void {
51     Node root = Node::root(*this);
52     insert_({ .key = key, .value = value }, root);
53     if (root.shouldSplit()) split_(root, root, 0);
54     root.update();
55   }
63   auto remove (const KeyType &key, const ValueType &value) -> void {
64     Node root = Node::root(*this);
65     remove_({ .key = key, .value = value }, root);
66     if (root.shouldMerge()) merge_(root, root, 0);
67     root.update();
68   }
70   auto findOne (const KeyType &key) -> Optional<ValueType> {
71     return findOne_(key, Node::root(*this));
72   }
74   auto findMany (const KeyType &key) -> Vector<ValueType> {
75     return findMany_(key, Node::root(*this));
76   }
78   auto findAll () -> Vector<ticket::Pair<KeyType, ValueType>> {
79     return findAll_(Node::root(*this));
80   }
82   auto includes (const KeyType &key, const ValueType &value) -> bool {
```

```
83      return includes_({ .key = key, .value = value }, Node::root(*this));
84    }
86    auto empty () -> bool {
87      return Node::root(*this).length() == 0;
88    }
89
91    auto getMeta () -> Meta {
92      return file_.getMeta();
93    }
95    auto setMeta (const Meta &meta) -> void {
96      return file_.setMeta(meta);
97    }
98
105    auto clearCache () -> void { file_.clearCache(); }
107    auto truncate () -> void {
108      file_.truncate();
109      init_();
110    }
111
112 #ifdef TICKET_DEBUG
113    auto print () -> void { print_(Node::root(*this)); }
114 #endif
115
116   private:
117    File<Meta, szChunk> file_;
118    CmpKey cmpKey_;
119    CmpValue cmpValue_;
120
121    // data structures
123    struct Pair {
124      KeyType key;
125      ValueType value;
126      auto operator< (const Pair &that) const -> bool {
127        CmpKey cmpKey_;
128        CmpValue cmpValue_;
129        if (!cmpKey_.equals(key, that.key)) return cmpKey_.lt(key, that.key);
130        return cmpValue_.lt(value, that.value);
131      }
132    };
134    class KeyComparatorLess_ {
135     public:
136      auto operator() (const Pair &lhs, const KeyType &rhs) -> bool {
137        return cmpKey_.lt(lhs.key, rhs);
138      }
139      auto operator() (const KeyType &lhs, const Pair &rhs) -> bool {
140        return cmpKey_.geq(rhs.key, lhs);
141      }
142     private:
143      CmpKey cmpKey_;
144      CmpValue cmpValue_;
145    };
146
147    using NodeId = unsigned int;
148    // ROOT and INTERMEDIATE nodes are index nodes
149    enum NodeType { kRoot, kIntermediate, kRecord };
150    // if k > kLengthMax, there must be an overflow.
151    static constexpr size_t kLengthMax = 18446744073709000000ULL;
152    struct IndexPayload {
153      static constexpr size_t k = (szChunk - 2 * sizeof(NodeId)) / (sizeof(NodeId) + sizeof(Pair)) / 2 -
       1;
154      static_assert(k >= 2 && k < kLengthMax);
155      bool leaf = false;
157      Array<NodeId, 2 * k> children;
158      Set<Pair, 2 * k> splits;
159    };
160    struct RecordPayload {
161      static constexpr size_t l = (szChunk - 3 * sizeof(NodeId)) / sizeof(Pair) / 2 - 1;
162      static_assert(l >= 2 && l < kLengthMax);
163      NodeId prev = 0;
164      NodeId next = 0;
165      Set<Pair, 2 * l> entries;
166    };
167    union NodePayload {
168      IndexPayload index;
169      RecordPayload record;
170      NodePayload () {} // NOLINT
171    };
172    struct Node : public internal::UnmanagedObject<Node, Meta, szChunk> {
173      char _start[0];
174      NodeType type;
175      NodePayload payload;
176      char _end[0];
177      static_assert(sizeof(NodeType) + sizeof(NodePayload) <= szChunk);
178
179      // dynamically type-safe accessors
180      auto leaf () -> bool & { TICKET_ASSERT(type != kRecord); return payload.index.leaf; }
181      auto children () -> Array<NodeId, 2 * IndexPayload::k> & { TICKET_ASSERT(type != kRecord); return
```

```
     payload.index.children; }
182    auto splits () -> Set<Pair, 2 * IndexPayload::k> & { TICKET_ASSERT(type != kRecord); return
     payload.index.splits; }
183    auto prev () -> NodeId & { TICKET_ASSERT(type == kRecord); return payload.record.prev; }
184    auto next () -> NodeId & { TICKET_ASSERT(type == kRecord); return payload.record.next; }
185    auto entries () -> Set<Pair, 2 * RecordPayload::l> & { TICKET_ASSERT(type == kRecord); return
     payload.record.entries; }
186
187    Node (BpTree &tree, NodeType type) : internal::UnmanagedObject<Node, Meta, szChunk>(tree.file_),
     type(type) {
188      if (type == kRecord) {
189        new(&payload.record) RecordPayload;
190      } else {
191        new(&payload.index) IndexPayload;
192      }
193    }
194    ~Node () {
195      if (type == kRecord) {
196        payload.record.~RecordPayload();
197      } else {
198        payload.index.~IndexPayload();
199      }
200    }
201
202    static auto root (BpTree &tree) -> Node { return Node::get(tree.file_, 0); }
203
204    auto halfLimit () -> size_t {
205      return type == kRecord ? RecordPayload::l : IndexPayload::k;
206    }
207    auto length () -> size_t {
208      return type == kRecord ? payload.record.entries.length : payload.index.children.length;
209    }
210    auto shouldSplit () -> bool { return length() == 2 * halfLimit(); }
211    auto shouldMerge () -> bool { return length() < halfLimit(); }
212    auto lowerBound () -> Pair {
213      return type == kRecord ? payload.record.entries[0] : payload.index.splits[0];
214    }
215  };
216
217  // helper functions
218  auto ixInsert_ (const Pair &entry, Node &node) -> size_t {
219    TICKET_ASSERT(node.type != kRecord);
220    auto &splits = node.splits();
221    size_t ix = upperBound(splits.content, splits.content + splits.length, entry) - splits.content;
222    return ix == 0 ? ix : ix - 1;
223  }
224  auto splitRoot_ (Node &node) -> void {
225    Node left(*this, kIntermediate), right(*this, kIntermediate);
226
227    // copy children and splits
228    left.children().copyFrom(node.children(), 0, 0, IndexPayload::k);
229    left.splits().copyFrom(node.splits(), 0, 0, IndexPayload::k);
230    right.children().copyFrom(node.children(), IndexPayload::k, 0, IndexPayload::k);
231    right.splits().copyFrom(node.splits(), IndexPayload::k, 0, IndexPayload::k);
232    left.children().length = left.splits().length = right.children().length = right.splits().length =
     IndexPayload::k;
233
234    // set misc properties and save
235    left.leaf() = right.leaf() = node.leaf();
236    node.leaf() = false;
237    left.save();
238    right.save();
239
240    // initiate the new root node
241    node.children().clear();
242    node.children().insert(left.id(), 0);
243    node.children().insert(right.id(), 1);
244    node.splits().clear();
245    node.splits().insert(left.lowerBound());
246    node.splits().insert(right.lowerBound());
247  }
248  auto split_ (Node &node, Node &parent, size_t ixChild) -> void {
249    TICKET_ASSERT(node.shouldSplit());
250 #ifdef TICKET_DEBUG_BPTREE
251    std::cerr « "[Split] " « node.id() « " (parent " « parent.id() « ")" « std::endl;
252 #endif
253    if (node.type == kRoot) {
254      // the split of the root node is a bit different from other nodes. it produces two extra subnodes.
255      splitRoot_(node);
256      return;
257    }
258    TICKET_ASSERT(node.type != kRoot);
259
260    // create a new next node
261    Node next(*this, node.type);
262    if (node.type == kIntermediate) {
263      next.children().copyFrom(node.children(), IndexPayload::k, 0, IndexPayload::k);
```

```
264       next.splits().copyFrom(node.splits(), IndexPayload::k, 0, IndexPayload::k);
265       node.children().length = node.splits().length = next.children().length = next.splits().length =
      IndexPayload::k;
266       next.leaf() = node.leaf();
267       next.save();
268     } else {
269       TICKET_ASSERT(node.type == kRecord);
270       next.next() = node.next();
271       next.prev() = node.id();
272       memmove(
273         next.entries().content,
274         &node.entries().content[RecordPayload::l],
275         RecordPayload::l * sizeof(node.entries()[0])
276       );
277       next.entries().length = node.entries().length = RecordPayload::l;
278       next.save();
279       if (next.next() != 0) {
280         Node nextnext = Node::get(file_, next.next());
281         nextnext.prev() = next.id();
282         nextnext.update();
283       }
284       node.next() = next.id();
285     }
286
287     // update the parent node
288     parent.children().insert(next.id(), ixChild + 1);
289     parent.splits().insert(next.lowerBound());
290   }
291
292   template <typename A, typename B>
293   static auto unshift_ (A &to, B &from, size_t k) -> void {
294     // we now have [b[0],...,b[k-1]] and [a[0]...a[k-2]], want a -> [b[0],...,b[k-1],a[0],...,a[k-2]]
295     to.copyFrom(to, 0, k, k - 1);
296     to.copyFrom(from, 0, 0, k);
297     to.length += from.length;
298     from.length = 0;
299   }
300   template <typename A, typename B>
301   static auto push_ (A &to, B &from, size_t k) -> void {
302     to.copyFrom(from, 0, k - 1, k);
303     to.length += from.length;
304     from.length = 0;
305   }
306   auto merge_ (Node &node, Node &parent, size_t ixChild) -> void {
307     TICKET_ASSERT(node.shouldMerge());
308 #ifdef TICKET_DEBUG_BPTREE
309     std::cerr « "[Merge] " « node.id() « " (parent " « parent.id() « ")" « std::endl;
310 #endif
311     if (node.type == kRoot) {
312       if (node.length() > 1 || node.leaf()) return;
313       Node onlyChild = Node::get(file_, node.children()[0]);
314       memcpy(&node, &onlyChild, sizeof(node));
315       node.type = kRoot;
316       return;
317     }
318     const bool hasPrev = ixChild != 0;
319     const bool hasNext = ixChild != parent.children().length - 1;
320     if (!hasNext) {
321       // don't do anything to the only data node.
322       if (!hasPrev && node.type == kRecord) return;
323       // all index nodes has at least 2 child nodes, except for the root node.
324       TICKET_ASSERT(hasPrev);
325       Node prev = Node::get(file_, parent.children()[ixChild - 1]);
326       if (prev.length() > prev.halfLimit()) {
327         if (node.type == kRecord) {
328           node.entries().insert(prev.entries().pop());
329         } else {
330           node.children().unshift(prev.children().pop());
331           node.splits().insert(prev.splits().pop());
332         }
333         prev.update();
334         parent.splits()[ixChild] = node.lowerBound();
335         return;
336       }
337       TICKET_ASSERT(prev.length() == prev.halfLimit());
338
339       if (node.type == kRecord) {
340         unshift_(node.entries(), prev.entries(), RecordPayload::l);
341         if (prev.prev() != 0) {
342           Node prevprev = Node::get(file_, prev.prev());
343           prevprev.next() = node.id();
344           prevprev.update();
345         }
346         node.prev() = prev.prev();
347       } else {
348         TICKET_ASSERT(node.type == kIntermediate);
349         unshift_(node.children(), prev.children(), IndexPayload::k);
```

```
350         unshift_(node.splits(), prev.splits(), IndexPayload::k);
351       }
352       parent.splits()[ixChild] = node.lowerBound();
353       parent.children().removeAt(ixChild - 1);
354       parent.splits().removeAt(ixChild - 1);
355       prev.destroy();
356       return;
357     }
358     TICKET_ASSERT(hasNext);
359
360     // FIXME: remove dupe code here
361     Node next = Node::get(file_, parent.children()[ixChild + 1]);
362     if (next.length() > next.halfLimit()) {
363       if (node.type == kRecord) {
364         node.entries().insert(next.entries().shift());
365       } else {
366         node.children().push(next.children().shift());
367         node.splits().insert(next.splits().shift());
368       }
369       next.update();
370       parent.splits()[ixChild + 1] = next.lowerBound();
371       return;
372     }
373     TICKET_ASSERT(next.length() == next.halfLimit());
374
375     if (node.type == kRecord) {
376       push_(node.entries(), next.entries(), RecordPayload::l);
377       if (next.next() != 0) {
378         Node nextnext = Node::get(file_, next.next());
379         nextnext.prev() = node.id();
380         nextnext.update();
381       }
382       node.next() = next.next();
383     } else {
384       TICKET_ASSERT(node.type == kIntermediate);
385       push_(node.children(), next.children(), IndexPayload::k);
386       push_(node.splits(), next.splits(), IndexPayload::k);
387     }
388
389     parent.children().removeAt(ixChild + 1);
390     parent.splits().removeAt(ixChild + 1);
391     next.destroy();
392   }
393
394   // FIXME: lengthy function name
395   auto addValuesToVectorForAllKeyFrom_ (Vector<ValueType> &vec, const KeyType &key, Node node, int
       first) -> void {
396     // we need to declare i outside to see if we have advanced to the last element
397     int i = first;
398     for (; i < node.length() && cmpKey_.equals(node.entries()[i].key, key); ++i)
       vec.push_back(node.entries()[i].value);
399     if (i == node.length() && node.next() != 0) addValuesToVectorForAllKeyFrom_(vec, key,
       Node::get(file_, node.next()), 0);
400   }
401   auto addEntriesToVector_ (Vector<ticket::Pair<KeyType, ValueType>> &vec, Node node) -> void {
402     for (int i = 0; i < node.length(); ++i) vec.emplace_back(node.entries()[i].key,
       node.entries()[i].value);
403     if (node.next() != 0) addEntriesToVector_(vec, Node::get(file_, node.next()));
404   }
405   auto findFirstChildWithKey_ (const KeyType &key, Node &node) -> ticket::Pair<Node, Optional<Node>> {
406     TICKET_ASSERT(node.type != kRecord);
407     size_t ixGreater = upperBound(
408       node.splits().content,
409       node.splits().content + node.length(),
410       key,
411       Less<KeyComparatorLess_>()
412     ) - node.splits().content;
413     bool hasCdr = ixGreater < node.length() && cmpKey_.equals(node.splits()[ixGreater].key, key);
414     auto cdr = hasCdr ? Optional<Node>(Node::get(file_, node.children()[ixGreater])) :
       Optional<Node>(unit);
415     size_t ix = ixGreater == 0 ? ixGreater : ixGreater - 1;
416     return { Node::get(file_, node.children()[ix]), cdr };
417   }
418
419   // operation functions
420   auto insert_ (const Pair &entry, Node &node) -> void {
421     if (node.type == kRecord) {
422       node.entries().insert(entry);
423       TICKET_ASSERT(node.entries().length <= 2 * RecordPayload::l);
424       return;
425     }
426     // if this is the first entry of the root, go create a record node.
427     if (node.children().length == 0) {
428       TICKET_ASSERT(node.type == kRoot);
429       TICKET_ASSERT(node.leaf());
430       Node child(*this, kRecord);
431       child.entries().insert(entry);
```

```
432       child.save();
433       node.children().push(child.id());
434       node.splits().insert(entry);
435       return;
436     }
437     size_t ix = ixInsert_(entry, node);
438     if (entry < node.splits()[ix]) node.splits()[ix] = entry;
439     Node nodeToInsert = Node::get(file_, node.children()[ix]);
440     insert_(entry, nodeToInsert);
441     node.splits()[ix] = nodeToInsert.lowerBound();
442     if (nodeToInsert.shouldSplit()) split_(nodeToInsert, node, ix);
443     nodeToInsert.update();
444   }
445   auto remove_ (const Pair &entry, Node &node) -> void {
446     if (node.type == kRecord) {
447       node.entries().remove(entry);
448       return;
449     }
450     size_t ix = ixInsert_(entry, node);
451     Node child = Node::get(file_, node.children()[ix]);
452     remove_(entry, child);
453     if (child.length() == 0) {
454       TICKET_ASSERT(node.type == kRoot);
455       TICKET_ASSERT(child.type == kRecord);
456       child.destroy();
457       node.children().clear();
458       node.splits().clear();
459       return;
460     }
461     node.splits()[ix] = child.lowerBound();
462     if (child.shouldMerge()) merge_(child, node, ix);
463     child.update();
464   }
465   auto findOne_ (const KeyType &key, Node node) -> Optional<ValueType> {
466     if (node.type != kRecord) {
467       if (node.length() == 0) return unit;
468       auto [ car, cdr ] = findFirstChildWithKey_(key, node);
469       if (!cdr) return findOne_(key, car);
470       auto res = findOne_(key, car);
471       if (res) return res;
472       return findOne_(key, *cdr);
473     }
474     size_t ix = upperBound(
475       node.entries().content,
476       node.entries().content + node.length(),
477       key,
478       Less<KeyComparatorLess_>()
479     ) - node.entries().content;
480     if (ix >= node.length()) return unit;
481     Pair entry = node.entries()[ix];
482     if (!cmpKey_.equals(entry.key, key)) return unit;
483     return entry.value;
484   }
485   auto includes_ (const Pair &entry, Node node) -> bool {
486     if (node.type == kRecord) return node.entries().includes(entry);
487     if (node.length() == 0) return false;
488     return includes_(entry, Node::get(file_, node.children()[ixInsert_(entry, node)]));
489   }
490   auto findMany_ (const KeyType &key, Node node) -> Vector<ValueType> {
491     if (node.type != kRecord) {
492       if (node.length() == 0) return {};
493       auto [ car, cdr ] = findFirstChildWithKey_(key, node);
494       if (!cdr) return findMany_(key, car);
495       Vector<ValueType> res = findMany_(key, car);
496       if (!res.empty()) return res;
497       return findMany_(key, *cdr);
498     }
499     size_t ix = upperBound(
500       node.entries().content,
501       node.entries().content + node.length(),
502       key,
503       Less<KeyComparatorLess_>()
504     ) - node.entries().content;
505     if (ix >= node.length()) return {};
506     Vector<ValueType> res;
507     addValuesToVectorForAllKeyFrom_(res, key, node, ix);
508     return res;
509   }
510   auto findAll_ (Node node) -> Vector<ticket::Pair<KeyType, ValueType>> {
511     if (node.type != kRecord) {
512       if (node.length() == 0) return {};
513       return findAll_(Node::get(file_, node.children()[0]));
514     }
515     Vector<ticket::Pair<KeyType, ValueType>> res;
516     addEntriesToVector_(res, node);
517     return res;
518   }
```

```
519   auto init_ () -> void {
520     Node root(*this, kRoot);
521     root.leaf() = true;
522     root.save();
523     TICKET_ASSERT(root.id() == 0);
524   }
525 #ifdef TICKET_DEBUG
526   auto print_ (Node node) -> void {
527     if (node.type == kRecord) {
528       std::cerr « "[Record " « node.id() « " (" « node.length() « "/" « 2 * RecordPayload::l - 1 « ")]";
529       for (int i = 0; i < node.length(); ++i) std::cerr « " (" « std::string(node.entries()[i].key) « ",
      " « node.entries()[i].value « ")";
530       std::cerr « std::endl;
531       return;
532     }
533     std::cerr « "[Node " « node.id() « " (" « node.length() « "/" « 2 * IndexPayload::k - 1 « ")" «
     (node.leaf() ? " leaf" : "") « "]";
534     for (int i = 0; i < node.length(); ++i) std::cerr « " (" « std::string(node.splits()[i].key) « ", "
     « node.splits()[i].value « ") " « node.children()[i];
535     std::cerr « std::endl;
536     for (int i = 0; i < node.length(); ++i) print_(Node::get(file_, node.children()[i]));
537   }
538 #endif
539 };
540
541 } // namespace ticket::file
542
543 #endif // TICKET_LIB_FILE_BPTREE_H_
```

7.12 lib/file/file.h File Reference

```
#include <cstring>
#include <fstream>
#include "hashmap.h"
#include "lru-cache.h"
#include "utility.h"
#include "exception.h"
```

Classes

- class ticket::file::File< Meta, szChunk >

  A chunked file storage with manual garbage collection.
- class ticket::file::Managed< T, Meta >

  an opinionated utility class wrapper for the objects to be stored.

Namespaces

- namespace ticket
- namespace ticket::file

  File utilities.

Variables

- constexpr size_t ticket::file::kDefaultSzChunk = 4096

7.13 file.h

Go to the documentation of this file.

```
1 // This file defines several basic file-based utilities.
2 #ifndef TICKET_LIB_FILE_FILE_H_
3 #define TICKET_LIB_FILE_FILE_H_
4
5 #include <cstring>
6 #include <fstream>
7
8 #include "hashmap.h"
9 #include "lru-cache.h"
10 #include "utility.h"
11 #include "exception.h"
12
14 namespace ticket::file {
15
16 constexpr size_t kDefaultSzChunk = 4096;
17
25 template <typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
26 class File {
27  private:
28   class Metadata;
29  public:
39   template <typename Functor>
40   File (const char *filename, const Functor &initializer) {
41     init_(filename, initializer);
42   }
43   File (const char *filename) {
44     init_(filename, [] {});
45   }
46   ~File () { clearCache(); }
47
49   auto get (void *buf, size_t index, size_t n) -> void {
50     if (auto cached = cache_.get(index)) {
51       memcpy(buf, *cached, n);
52       return;
53     }
54     file_.seekg(offset_(index));
55     file_.read((char *) buf, n);
56     TICKET_ASSERT(file_.good());
57     // TODO(perf): memcmp overhead
58     cache_.upsert(index, buf, n);
59   }
61   auto set (const void *buf, size_t index, size_t n) -> void {
62     if (!cache_.upsert(index, buf, n)) return;
63     file_.seekp(offset_(index));
64     file_.write((const char *) buf, n);
65     TICKET_ASSERT(file_.good());
66   }
68   auto push (const void *buf, size_t n) -> size_t {
69     Metadata meta = meta_();
70     size_t id = meta.next;
71     if (meta.hasNext) {
72       Metadata nextMeta;
73       get(&nextMeta, meta.next, sizeof(nextMeta));
74       set(&nextMeta, -1, sizeof(nextMeta));
75     } else {
76       ++meta.next;
77       set(&meta, -1, sizeof(meta));
78     }
79     set(buf, id, n);
80     return id;
81   }
82   auto remove (size_t index) -> void {
83     Metadata meta = meta_();
84     set(&meta, index, sizeof(meta));
85     Metadata newMeta(index, true);
86     set(&newMeta, -1, sizeof(newMeta));
87     cache_.remove(index);
88   }
89
91   auto getMeta () -> Meta {
92     return meta_().user;
93   }
95   auto setMeta (const Meta &user) -> void {
96     Metadata meta = meta_();
97     meta.user = user;
98     set(&meta, -1, sizeof(meta));
99   }
100
102   auto clearCache () -> void {
103     cache_.clear();
104   }
105
107   auto truncate () -> void {
```

```
108     Metadata meta(0, false);
109     set(&meta, -1, sizeof(meta));
110   }
111
112  private:
113   struct Metadata {
114     size_t next;
115     bool hasNext;
116     Meta user;
117     Metadata () = default;
118     Metadata (size_t next, bool hasNext) : next(next), hasNext(hasNext) {}
119   };
120   static_assert(szChunk > sizeof(Metadata));
121
122   template <typename Functor>
123   auto init_ (const char *filename, const Functor &initializer) -> void {
124     bool shouldCreate = false;
125     auto testFile = fopen(filename, "r");
126     if (testFile == nullptr) {
127       if (errno != ENOENT) {
128         throw IoException("Unable to open file");
129       }
130       shouldCreate = true;
131     } else if (fclose(testFile)) {
132       throw IoException("Unable to close file");
133     }
134     if (shouldCreate) {
135       auto file = fopen(filename, "w+");
136       if (file == nullptr) {
137         throw IoException("Unable to create file");
138       }
139       if (fclose(file)) {
140         throw IoException("Unable to close file when creating file");
141       }
142     }
143     file_.open(filename);
144     if (!file_.is_open() || !file_.good()) {
145       throw IoException("Unable to open file");
146     }
147     if (shouldCreate) {
148       truncate();
149       initializer();
150     }
151   }
152
153   auto meta_ () -> Metadata {
154     Metadata retval;
155     get(&retval, -1, sizeof(retval));
156     return retval;
157   }
158   auto offset_ (size_t index) -> size_t {
159     return (index + 1) * szChunk;
160   }
161   std::fstream file_;
162   constexpr static int kSzCache_ = 1024;
163   LruCache<size_t, kSzCache_> cache_;
164 };
165
174 template <typename T, typename Meta = Unit>
175 class Managed : public T {
176  public:
178   static File<Meta, sizeof(T)> file;
179
187   auto id () const -> int { return (int) id_; }
188
190   static auto get (size_t id) -> Managed {
191     char buf[sizeof(Managed)];
192     auto managed = reinterpret_cast<Managed *>(buf);
193     auto unmanaged = static_cast<T *>(managed);
194     file.get(unmanaged, id, sizeof(T));
195     managed->id_ = id;
196     return *managed;
197   }
199   static auto truncate () -> void {
200     file.truncate();
201   }
202
209   auto save () -> void {
210     TICKET_ASSERT(id_ == -1);
211     id_ = file.push(static_cast<T *>(this), sizeof(T));
212   }
214   auto update () -> void {
215     TICKET_ASSERT(id_ != -1);
216     file.set(static_cast<T *>(this), id_, sizeof(T));
217   }
219   auto destroy () -> void {
220     TICKET_ASSERT(id_ != -1);
```

```
221     file.remove(id_);
222     id_ = -1;
223   }
224  private:
225   size_t id_ = -1;
226 };
227
228 template <typename T, typename Meta>
229 File<Meta, sizeof(T)> Managed<T, Meta>::file { T::filename };
230
231 } // namespace ticket::file
232
233 #endif // TICKET_LIB_FILE_FILE_H_
```

7.14 lib/file/index.h File Reference

```
#include "file/bptree.h"
#include "file/varchar.h"
#include "optional.h"
#include "vector.h"
```

Classes

- class ticket::file::Index< Key, Model >

  Class representing an index file.
- class ticket::file::Index< Varchar< maxLength >, Model >

  Specialization of Index on Varchar.

Namespaces

- namespace ticket
- namespace ticket::file

  File utilities.

7.15 index.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_FILE_INDEX_H_
2 #define TICKET_LIB_FILE_INDEX_H_
3
4 #include "file/bptree.h"
5 #include "file/varchar.h"
6 #include "optional.h"
7 #include "vector.h"
8
9 namespace ticket::file {
10
20 template <typename Key, typename Model>
21 class Index {
22  public:
29   Index (Key Model::*ptr, const char *filename)
30     : ptr_(ptr), tree_(filename) {}
32   auto insert (const Model &model) -> void {
33     tree_.insert(model.*ptr_, model.id());
34   }
36   auto remove (const Model &model) -> void {
37     tree_.remove(model.*ptr_, model.id());
38   }
40   auto findOne (const Key &key) -> Optional<Model> {
41     auto id = tree_.findOne(key);
42     if (!id) return unit;
43     return Model::get(*id);
44   }
```

```
46   auto findOneId (const Key &key) -> Optional<int> {
47     return tree_.findOne(key);
48   }
50   auto findMany (const Key &key) -> Vector<Model> {
51     Vector<Model> res;
52     auto ids = tree_.findMany(key);
53     if (ids.size() > 0) res.reserve(ids.size());
54     for (auto id : ids) {
55       res.push_back(Model::get(id));
56     }
57     return res;
58   }
60   auto findManyId (const Key &key) -> Vector<int> {
61     return tree_.findMany(key);
62   }
64   auto empty () -> bool {
65     return tree_.empty();
66   }
67
69   auto truncate () -> void {
70     tree_.truncate();
71   }
72  private:
73   Key Model::*ptr_;
74   BpTree<Key, int> tree_;
75 };
76
82 template <size_t maxLength, typename Model>
83 class Index<Varchar<maxLength>, Model> {
84  private:
85   using Key = Varchar<maxLength>;
86  public:
93   Index (Key Model::*ptr, const char *filename)
94     : ptr_(ptr), tree_(filename) {}
96   auto insert (const Model &model) -> void {
97     tree_.insert(model.*ptr_.hash(), model.id());
98   }
100   auto remove (const Model &model) -> void {
101     tree_.remove(model.*ptr_.hash(), model.id());
102   }
104   auto findOne (const Key &key) -> Optional<Model> {
105     auto id = tree_.findOne(key.hash());
106     if (!id) return unit;
107     return Model::get(*id);
108   }
110   auto findOneId (const Key &key) -> Optional<int> {
111     return tree_.findOne(key.hash());
112   }
114   auto findMany (const Key &key) -> Vector<Model> {
115     Vector<Model> res;
116     auto ids = tree_.findMany(key.hash());
117     return ids.map(Model::get);
118   }
120   auto findManyId (const Key &key) -> Vector<int> {
121     return tree_.findMany(key.hash());
122   }
124   auto empty () -> bool {
125     return tree_.empty();
126   }
127
129   auto truncate () -> void {
130     tree_.truncate();
131   }
132  private:
133   Key Model::*ptr_;
134   BpTree<size_t, int> tree_;
135 };
136
137 } // namespace ticket::file
138
139 #endif // TICKET_LIB_FILE_INDEX_H_
```

7.16 lib/file/set.h File Reference

```
#include <cstring>
#include "algorithm.h"
#include "exception.h"
#include "utility.h"
```

Classes

- struct ticket::file::Set< T, maxLength, Cmp >

  A sorted array with utility functions and bound checks.

Namespaces

- namespace ticket
- namespace ticket::file

  File utilities.

7.17 set.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_FILE_SET_H_
2 #define TICKET_LIB_FILE_SET_H_
3
4 #include <cstring>
5
6 #include "algorithm.h"
7 #include "exception.h"
8 #include "utility.h"
9
10 // FIXME: remove dupe code of Set and Array. does C++ support mixins?
11 namespace ticket::file {
12
14 template <typename T, size_t maxLength, typename Cmp = Less<>>
15 struct Set {
16  private:
17   auto boundsCheck_ (size_t index) const -> void {
18     if (index >= length) {
19       throw OutOfBounds("Set: overflow or underflow");
20     }
21   }
22   Cmp cmp_;
23  public:
24   Set () = default;
25   size_t length = 0;
26   T content[maxLength];
27   auto indexOfInsert (const T &element) -> size_t {
28     return lowerBound(content, content + length, element) - content;
29   }
31   auto indexOf (const T &element) -> size_t {
32     size_t index = indexOfInsert(element);
33     if (index >= length || !cmp_.equals(content[index], element)) {
34       throw NotFound("Set::indexOf: element not found");
35     }
36     return index;
37   }
39   auto includes (const T &element) -> bool {
40     size_t ix = indexOfInsert(element);
41     return ix < length && cmp_.equals(content[ix], element);
42   }
44   auto insert (const T &element) -> void {
45     if (length == maxLength) {
46       throw Overflow("Set::insert: overflow");
47     }
48     size_t offset = indexOfInsert(element);
49     if (offset != length) {
50       memmove(
51         &content[offset + 1],
52         &content[offset],
53         (length - offset) * sizeof(content[0])
54       );
55     }
56     content[offset] = element;
57     ++length;
58   }
59
61   auto remove (const T &element) -> void {
62     removeAt(indexOf(element));
63   }
65   auto removeAt (size_t offset) -> void {
66     boundsCheck_(offset);
67     if (offset != length - 1) {
```

```
68        memmove(
69          &content[offset],
70          &content[offset + 1],
71          (length - offset - 1) * sizeof(content[0])
72        );
73      }
74      --length;
75    }
77    auto clear () -> void { length = 0; }
78
80    void copyFrom (const Set &other, size_t ixFrom, size_t ixTo, size_t count) {
81      if (this == &other) {
82        memmove(
83          &content[ixTo],
84          &content[ixFrom],
85          count * sizeof(content[0])
86        );
87      } else {
88        memcpy(
89          &content[ixTo],
90          &other.content[ixFrom],
91          count * sizeof(content[0])
92        );
93      }
94    }
95
96    auto operator[] (size_t index) -> T & {
97      boundsCheck_(index);
98      return content[index];
99    }
100    auto operator[] (size_t index) const -> const T & {
101      boundsCheck_(index);
102      return content[index];
103    }
104
106    auto pop () -> T {
107      if (length == 0) throw Underflow("Set::pop: underflow");
108      return content[--length];
109    }
111    auto shift () -> T {
112      if (length == 0) throw Underflow("Set::pop: underflow");
113      T result = content[0];
114      removeAt(0);
115      return result;
116    }
117
119    template <typename Functor>
120    auto forEach (const Functor &callback) -> void {
121      for (int i = 0; i < length; ++i) callback(content[i]);
122    }
123  };
124
125  } // namespace ticket::file
126
127  #endif // TICKET_LIB_FILE_SET_H_
```

7.18 lib/file/varchar.h File Reference

```
#include <cstring>
#include <iostream>
#include "exception.h"
```

Classes

- struct ticket::file::Varchar< maxLength >

  *A wrapper for const char * with utility functions and type conversions.*

Namespaces

- namespace ticket
- namespace ticket::file

  File utilities.

7.19 varchar.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_FILE_VARCHAR_H_
2 #define TICKET_LIB_FILE_VARCHAR_H_
3
4 #include <cstring>
5 #include <iostream>
6
7 #include "exception.h"
8
9 namespace ticket::file {
10
21 template <int maxLength>
22 struct Varchar {
23  public:
24   static constexpr int kMaxLength = maxLength;
25   Varchar () { content[0] = '\0'; }
26   Varchar (const std::string &s) {
27     if (s.length() > maxLength) {
28       throw Overflow("Varchar length overflow");
29     }
30     strncpy(content, s.c_str(), maxLength + 1);
31   }
32   Varchar (const char *cstr) : Varchar(std::string(cstr)) {
33     if (strlen(cstr) > maxLength) {
34       throw Overflow("Varchar length overflow");
35     }
36     strncpy(content, cstr, maxLength + 1);
37   }
38
39   template<int A>
40   Varchar (const Varchar<A> &that) { *this = that; }
41   operator std::string () const {
42     return std::string(content);
43   }
44   [[nodiscard]] auto str () const -> std::string {
45     return std::string(*this);
46   }
47
48   auto length () const -> int {
49     return strlen(content);
50   }
51
52   template <int A>
53   auto operator= (const Varchar<A> &that) -> Varchar & {
54     if (that.length() > maxLength) {
55       throw Overflow("Varchar length overflow");
56     }
57     strcpy(content, that.content);
58     hash_ = that.hash_;
59     return *this;
60   }
61
62   template <int A>
63   auto operator< (const Varchar<A> &that) const -> bool {
64     return hash() < that.hash();
65   }
66   template <int A>
67   auto operator== (const Varchar<A> &that) const -> bool {
68     return hash() == that.hash();
69   }
70   template <int A>
71   auto operator!= (const Varchar<A> &that) const -> bool {
72     return hash() != that.hash();
73   }
74
75   auto hash () const -> size_t {
76     if (hash_ != 0) return hash_;
77     return hash_ = std::hash<std::string_view>()(content);
78   }
79
80  private:
81   template <int A>
82   friend class Varchar;
83   char content[maxLength + 1];
84   mutable size_t hash_ = 0;
85 };
86
87 } // namespace ticket::file
88
89 #endif // TICKET_LIB_FILE_VARCHAR_H_
```

7.20 lib/hashmap.h File Reference

```
#include <functional>
#include <cstddef>
#include "exception.h"
#include "utility.h"
#include "internal/rehash.inc"
```

Classes

- class ticket::HashMap< Key, Value, Hash, Equal >

  An unordered hash-based map.
- class ticket::HashMap< Key, Value, Hash, Equal >::iterator
- class ticket::HashMap< Key, Value, Hash, Equal >::const_iterator

Namespaces

- namespace ticket

7.21 hashmap.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_HASHMAP_H_
2 #define TICKET_LIB_HASHMAP_H_
3
4 // only for std::equal_to<T> and std::hash<T>
5 #include <functional>
6 #include <cstddef>
7
8 #include "exception.h"
9 #include "utility.h"
10
11 #ifdef DEBUG
12 #include <iostream>
13 #endif
14
15 namespace ticket {
16
17 #include "internal/rehash.inc"
18
30 template <
31   typename Key,
32   typename Value,
33   typename Hash = std::hash<Key>,
34   typename Equal = std::equal_to<Key>
35 > class HashMap {
36  private:
37   struct ListNode;
38   struct Node;
39  public:
40   using value_type = Pair<const Key, Value>;
41
42   class const_iterator;
43   class iterator {
44    public:
45     using difference_type = std::ptrdiff_t;
46     using value_type = HashMap::value_type;
47     using pointer = value_type *;
48     using reference = value_type &;
49     using iterator_category = std::output_iterator_tag;
50
51     iterator () = default;
52     iterator (ListNode *node, HashMap *home) : node_(node), home_(home) {}
53     auto operator++ (int) -> iterator {
54       if (node_ == &home_->pivot_) throw Exception("invalid state");
55       auto node = node_;
56       node_ = node_->next_;
```

```
57        return { node, home_ };
58      }
59      auto operator++ () -> iterator & {
60        if (node_ == &home_->pivot_) throw Exception("invalid state");
61        node_ = node_->next_;
62        return *this;
63      }
64      auto operator-- (int) -> iterator {
65        if (node_ == home_->pivot_.next_) throw Exception("invalid state");
66        auto node = node_;
67        node_ = node_->prev_;
68        return { node, home_ };
69      }
70      auto operator-- () -> iterator & {
71        if (node_ == home_->pivot_.next_) throw Exception("invalid state");
72        node_ = node_->prev_;
73        return *this;
74      }
75      auto operator* () const -> reference {
76        return node_->self->value;
77      }
78      auto operator== (const iterator &rhs) const -> bool {
79        return node_ == rhs.node_;
80      }
81      auto operator== (const const_iterator &rhs) const -> bool {
82        return node_ == rhs.node_;
83      }
84      auto operator!= (const iterator &rhs) const -> bool {
85        return !(*this == rhs);
86      }
87      auto operator!= (const const_iterator &rhs) const -> bool {
88        return !(*this == rhs);
89      }
90      auto operator-> () const noexcept -> pointer {
91        return &**this;
92      }
93    private:
94      ListNode *node_;
95      HashMap *home_;
96      friend class const_iterator;
97      friend class HashMap;
98    };
99
100   class const_iterator {
101    public:
102     using difference_type = std::ptrdiff_t;
103     using value_type = const HashMap::value_type;
104     using pointer = value_type *;
105     using reference = value_type &;
106     using iterator_category = std::output_iterator_tag;
107
108     const_iterator () = default;
109     const_iterator (const ListNode *node, const HashMap *home) : node_(node), home_(home) {}
110     const_iterator (const iterator &other) : node_(other.node_), home_(other.home_) {}
111     auto operator++ (int) -> const_iterator {
112       if (node_ == &home_->pivot_) throw Exception("invalid state");
113       auto node = node_;
114       node_ = node_->next_;
115       return { node, home_ };
116     }
117     auto operator++ () -> const_iterator & {
118       if (node_ == &home_->pivot_) throw Exception("invalid state");
119       node_ = node_->next_;
120       return *this;
121     }
122     auto operator-- (int) -> const_iterator {
123       if (node_ == home_->pivot_.next_) throw Exception("invalid state");
124       auto node = node_;
125       node_ = node_->prev_;
126       return { node, home_ };
127     }
128     auto operator-- () -> const_iterator & {
129       if (node_ == home_->pivot_.next_) throw Exception("invalid state");
130       node_ = node_->prev_;
131       return *this;
132     }
133     auto operator* () const -> reference {
134       return node_->self->value;
135     }
136     auto operator== (const iterator &rhs) const -> bool {
137       return node_ == rhs.node_;
138     }
139     auto operator== (const const_iterator &rhs) const -> bool {
140       return node_ == rhs.node_;
141     }
142     auto operator!= (const iterator &rhs) const -> bool {
143       return !(*this == rhs);
```

```
144     }
145     auto operator!= (const const_iterator &rhs) const -> bool {
146       return !(*this == rhs);
147     }
148     auto operator-> () const noexcept -> pointer {
149       return &**this;
150     }
151    private:
152     const ListNode *node_;
153     const HashMap *home_;
154     friend class iterator;
155     friend class HashMap;
156   };
157
158   HashMap () = default;
159   HashMap (const HashMap &other) { *this = other; }
160   auto operator= (const HashMap &other) -> HashMap & {
161     if (this == &other) return *this;
162     clear();
163     capacity_ = other.capacity_;
164     size_ = other.size_;
165     store_ = new ListNode[internal::pow2[capacity_]];
166     const ListNode *node = &other.pivot_;
167     for (int i = 0; i < size_; ++i) {
168       node = node->next_;
169       Node *newNode = new Node(*(node->self));
170       int ix = newNode->hash & internal::mask[capacity_];
171       newNode->hashList.insertBefore(&store_[ix]);
172       newNode->iteratorList.insertBefore(&pivot_);
173     }
174     return *this;
175   }
176   ~HashMap () {
177     destroy_();
178   }
179
185   auto at (const Key &key) -> Value & {
186     auto it = find(key);
187     if (it == end()) throw OutOfBounds();
188     return it->second;
189   }
190   auto at (const Key &key) const -> const Value & {
191     return const_cast<HashMap *>(this)->at(key);
192   }
193
199   auto operator[] (const Key &key) -> Value & {
200     return insert({ key, Value() }).first->second;
201   }
202
204   auto operator[] (const Key &key) const -> const Value & { return at(key); }
205
207   auto begin () -> iterator { return { pivot_.next_, this }; }
208   auto cbegin () const -> const_iterator { return { pivot_.next_, this }; }
209
211   auto end () -> iterator { return { &pivot_, this }; }
212   auto cend () const -> const_iterator { return { &pivot_, this }; }
213
215   auto empty () const -> bool {
216     return size_ == 0;
217   }
219   auto size () const -> size_t {
220     return size_;
221   }
222
224   auto clear () -> void {
225     destroy_();
226   }
227
234   auto insert (const value_type &value) -> Pair<iterator, bool> {
235     auto &[ k, _ ] = value;
236     auto hash = hash_(k);
237     if (capacity_ > 0) {
238       int ix = hash & internal::mask[capacity_];
239       if (store_[ix].next() != nullptr) {
240         Node *node = store_[ix].next()->find(k);
241         if (node != nullptr) return { { &node->iteratorList, this }, false };
242       }
243     }
244     growIfNeeded_();
245     int ix = hash & internal::mask[capacity_];
246     Node *node = new Node(value, hash);
247     node->hashList.insertBefore(&store_[ix]);
248     node->iteratorList.insertBefore(&pivot_);
249     ++size_;
250     return { { &node->iteratorList, this }, true };
251   }
252
```

```
257   auto erase (iterator pos) -> void {
258     if (pos == end() || pos.home_ != this) throw Exception("invalid state");
259     pos.node_->self->hashList.remove();
260     pos.node_->self->iteratorList.remove();
261     delete pos.node_->self;
262     pos.node_ = &pivot_;
263     --size_;
264   }
265
272   auto count (const Key &key) const -> size_t {
273     return find(key) == cend() ? 0 : 1;
274   }
276   auto contains (const Key &key) const -> bool {
277     return find(key) != cend();
278   }
279
286   auto find (const Key &key) -> iterator {
287     if (empty()) return end();
288     auto ix = hash_(key) & internal::mask[capacity_];
289     if (store_[ix].next() == nullptr) return end();
290     Node *node = store_[ix].next()->find(key);
291     if (node == nullptr) return end();
292     return { &node->iteratorList, this };
293   }
294   auto find (const Key &key) const -> const_iterator {
295     return const_cast<HashMap *>(this)->find(key);
296   }
297
298  private:
299   struct ListNode {
300     ListNode *prev_ = this;
301     ListNode *next_ = this;
302     auto next () -> Node * { return next_->self; }
303     auto prev () -> Node * { return prev_->self; }
304     Node *self = nullptr;
305     ListNode () = default;
306     ListNode (Node *node) : self(node) {}
307
308     auto insertBefore (ListNode *pivot) -> void {
309       prev_ = pivot->prev_;
310       next_ = pivot;
311       pivot->prev_ = prev_->next_ = this;
312     }
313     auto remove () -> void {
314       prev_->next_ = next_;
315       next_->prev_ = prev_;
316     }
317     auto init () -> void {
318       prev_ = next_ = this;
319     }
320   };
321
322   struct Node {
323     value_type value;
324     unsigned hash;
325     ListNode iteratorList = this, hashList = this;
326     Node () = default;
327     Node (const Node &node) : value(node.value), hash(node.hash) {}
328     Node (const value_type &value, unsigned hash) : value(value), hash(hash) {}
329     auto find (const Key &key) -> Node * {
330       if (Equal()(key, value.first)) return this;
331       if (hashList.next() == nullptr) return nullptr;
332       return hashList.next()->find(key);
333     }
334   };
335   ListNode pivot_;
336   ListNode *store_ = nullptr;
337   int size_ = 0;
338   int capacity_ = 0;
339   constexpr static int kThreshold_ = 2;
340   Hash hash0_;
341   auto hash_ (const Key &key) const -> unsigned {
342     return internal::rehash(hash0_(key));
343   }
344   auto growIfNeeded_ () -> void {
345     auto capacityNeeded = static_cast<unsigned long long>((size_ + 1) * kThreshold_);
346     if (capacityNeeded > internal::pow2[capacity_]) grow_();
347   }
348   auto grow_ () -> void {
349     if (capacity_ == 0) {
350       capacity_ = 2;
351       store_ = new ListNode[4];
352       return;
353     }
354     int newCapacity = capacity_ + 1;
355     auto prospective = new ListNode[internal::pow2[newCapacity]];
356     auto node = &pivot_;
```

```
357     for (int i = 0; i < size_; ++i) {
358       node = node->next_;
359       int ix = node->self->hash & internal::mask[newCapacity];
360       node->self->hashList.insertBefore(&prospective[ix]);
361     }
362     capacity_ = newCapacity;
363     delete[] store_;
364     store_ = prospective;
365   }
366
367   auto destroy_ () -> void {
368     ListNode *node = pivot_.next_;
369     for (int i = 0; i < size_; ++i) {
370       ListNode *next = node->next_;
371       delete node->self;
372       node = next;
373     }
374     capacity_ = 0;
375     size_ = 0;
376     delete[] store_;
377     store_ = nullptr;
378     pivot_.init();
379   }
380 };
381
382 } // namespace ticket
383
384 #endif // TICKET_LIB_HASHMAP_H_
```

7.22 lib/lru-cache.h File Reference

```
#include <cstring>
#include "hashmap.h"
#include "map.h"
#include "optional.h"
#include "utility.h"
```

Classes

- class ticket::LruCache< Key, kSize >

  A fixed-size cache with a least recently used policy.

Namespaces

- namespace ticket

7.23 lru-cache.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_LRU_CACHE_H_
2 #define TICKET_LIB_LRU_CACHE_H_
3
4 #include <cstring>
5
6 #include "hashmap.h"
7 #include "map.h"
8 #include "optional.h"
9 #include "utility.h"
10
11 namespace ticket {
12
14 template <typename Key, int kSize>
15 class LruCache {
16  private:
17   struct WeightedKey;
```

```
18  public:
19   ~LruCache () {
20     clear();
21   }
23   auto get (const Key &key) -> Optional<void *> {
24     auto it = storage_.find(key);
25     if (it == storage_.end()) return unit;
26     touch_(it);
27     return it->second.value;
28   }
36   auto upsert (const Key &key, const void *buf, int length)
37     -> bool {
38     auto it = storage_.find(key);
39     if (it == storage_.end()) {
40       // the key is not in the cache; insert.
41
42       // is there enough space for a new entry?
43       TICKET_ASSERT(index_.size() <= kSize);
44       TICKET_ASSERT(index_.size() == storage_.size());
45       if (index_.size() == kSize) {
46         // not enough space; clean the lru entry.
47         auto willDelete = index_.begin();
48         auto &key = willDelete->second;
49
50         auto willDeleteStorage = storage_.find(key);
51         auto value = willDeleteStorage->second.value;
52         delete[] value;
53
54         storage_.erase(willDeleteStorage);
55         index_.erase(willDelete);
56       }
57       TICKET_ASSERT(index_.size() < kSize);
58       TICKET_ASSERT(index_.size() == storage_.size());
59
60       // okay, we must have enough space here.
61       ++currentTime_;
62       index_[currentTime_] = key;
63       storage_[key] =
64         { currentTime_, (char *) allocate_(buf, length) };
65
66       // the cache has changed.
67       return true;
68     } // if (it == storage_.end())
69
70     // the key is in the cache. check, then update if needed
71     touch_(it);
72     auto &value = it->second.value;
73     if (memcmp(buf, value, length) == 0) {
74       // content is identical, no update needed.
75       return false;
76     }
77     // content is different, update needed.
78     delete[] value;
79     value = (char *) allocate_(buf, length);
80     return true;
81   }
82
84   auto remove (const Key &key) -> void {
85     auto it = storage_.find(key);
86     if (it == storage_.end()) return;
87     delete[] it->second.value;
88     index_.erase(index_.find(it->second.accessTime));
89     storage_.erase(it);
90   }
91
93   auto clear () -> void {
94     for (auto &pair : storage_) delete[] pair.second.value;
95     index_.clear();
96     storage_.clear();
97   }
98  private:
99   static_assert(kSize >= 2);
100    struct WeightedValue {
101      int accessTime;
102      char *value;
103    };
104    int currentTime_ = 0;
105    Map<int, Key> index_;
106    HashMap<Key, WeightedValue> storage_;
107
108    template <typename Iterator>
109    auto touch_ (Iterator it) -> void {
110      const auto &key = it->first;
111      auto &value = it->second;
112      index_.erase(index_.find(value.accessTime));
113      value.accessTime = ++currentTime_;
114      index_[value.accessTime] = key;
```

```
115   }
116
117   auto allocate_ (const void *buf, int length) -> void * {
118     char *copy = new char[length];
119     memcpy(copy, buf, length);
120     return copy;
121   }
122 };
123
124 } // namespace ticket
125
126 #endif // TICKET_LIB_LRU_CACHE_H_
```

7.24 lib/map.h File Reference

```
#include <cstddef>
#include "internal/tree.h"
#include "utility.h"
#include "exception.h"
#include "internal/map-value-compare.inc"
```

Classes

- class ticket::Map< KeyType, ValueType, Compare >

  A sorted key-value map backed by a red-black tree.

Namespaces

- namespace ticket

7.25 map.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_MAP_H_
2 #define TICKET_LIB_MAP_H_
3
4 #include <cstddef>
5
6 #include "internal/tree.h"
7 #include "utility.h"
8 #include "exception.h"
9
10 #ifdef DEBUG
11 #include <iostream>
12 #endif
13
14 namespace ticket {
15
16 #include "internal/map-value-compare.inc"
17
19 template <typename KeyType, typename ValueType, typename Compare = internal::LessOp>
20 class Map {
21  public:
22   using value_type = Pair<const KeyType, ValueType>;
23  private:
24   using TreeType = typename internal::RbTree<value_type, internal::MapValueCompare<KeyType, ValueType,
      Compare»;
25  public:
26   using iterator = typename TreeType::iterator;
27   using const_iterator = typename TreeType::const_iterator;
28
29   Map () = default;
35   auto at (const KeyType &key) -> ValueType & {
36     auto it = tree_.find(key);
37     if (it == tree_.end()) throw OutOfBounds();
```

```
38      return it->second;
39    }
40    auto at (const KeyType &key) const -> const ValueType & {
41      auto it = tree_.find(key);
42      if (it == tree_.cend()) throw OutOfBounds();
43      return it->second;
44    }
50    auto operator[] (const KeyType &key) -> ValueType & {
51      // we need to use the default constructor here. Too bad we have no choice.
52      auto p = tree_.insert({ key, ValueType() });
53      return p.first->second;
54    }
58    auto operator[] (const KeyType &key) const -> const ValueType & {
59      return at(key);
60    }
64    auto begin () -> iterator {
65      return tree_.begin();
66    }
67    auto cbegin () const -> const_iterator {
68      return tree_.cbegin();
69    }
74    auto end () -> iterator {
75      return tree_.end();
76    }
77    auto cend () const -> const_iterator {
78      return tree_.cend();
79    }
84    auto empty () const -> bool {
85      return tree_.empty();
86    }
90    auto size () const -> size_t {
91      return tree_.size();
92    }
96    auto clear () -> void {
97      tree_.clear();
98    }
105    auto insert (const value_type &value) -> Pair<iterator, bool> {
106      return tree_.insert(value);
107    }
112    auto erase (iterator pos) -> void {
113      return tree_.erase(pos);
114    }
122    auto count (const KeyType &key) const -> size_t {
123      auto it = tree_.find(key);
124      return it == tree_.cend() ? 0 : 1;
125    }
132    auto find (const KeyType &key) -> iterator {
133      return tree_.find(key);
134    }
135    auto find (const KeyType &key) const -> const_iterator {
136      return tree_.find(key);
137    }
138
139  #ifdef DEBUG
140    auto print () -> void {
141      std::cout « "s=" « size() « " ";
142      for (const auto &p : *this) {
143        std::cout « "(" « p.first.print() « ", " « p.second.print() « ") ";
144      }
145      std::cout « std::endl;
146    }
147  #endif
148
149   private:
150    TreeType tree_;
151  };
152
153  } // namespace ticket
154
155  #endif // TICKET_LIB_MAP_H_
```

7.26 lib/optional.h File Reference

```
#include "utility.h"
#include "variant.h"
```

Classes

- class ticket::Optional< T >

A resemblence of std::optional.

Namespaces

- namespace ticket

7.27 optional.h

Go to the documentation of this file.

```
1
2 #ifndef TICKET_LIB_OPTIONAL_H_
3 #define TICKET_LIB_OPTIONAL_H_
4
5 #include "utility.h"
6 #include "variant.h"
7
8 namespace ticket {
9
19 template <typename T>
20 class Optional : Variant<Unit, T> {
21  private:
22   using VarT = Variant<Unit, T>;
23  public:
24   Optional () = default;
26   Optional (Unit /* unused */) : VarT(unit) {}
28   template <
29     typename Init,
30     typename = std::enable_if_t<!std::is_same_v<Init, Unit»
31   >
32   Optional (const Init &value) : VarT(T(value)) {}
33   auto operator= (Unit unit) -> Optional & {
34     VarT::operator=(unit);
35     return *this;
36   }
37   template <
38     typename Init,
39     typename = std::enable_if_t<!std::is_same_v<Init, Unit»
40   >
41   auto operator= (const Init &value) -> Optional & {
42     VarT::operator=(T(value));
43     return *this;
44   }
46   operator bool () const {
47     return this->template is<T>();
48   }
50   auto operator* () -> T & {
51     return *this->template get<T>();
52   }
53   auto operator* () const -> const T & {
54     return *this->template get<T>();
55   }
56   auto operator-> () -> T * {
57     return this->template get<T>();
58   }
59   auto operator-> () const -> const T * {
60     return this->template get<T>();
61   }
62 };
63
64 } // namespace ticket
65
66 #endif // TICKET_LIB_OPTIONAL_H_
```

7.28 lib/result.h File Reference

```
#include "utility.h"
#include "variant.h"
```

Classes

- class ticket::Result< ResultType, ErrorType >

  Result<Res, Err> = Res | Err.

Namespaces

- namespace ticket

7.29 result.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_RESULT_H_
2 #define TICKET_LIB_RESULT_H_
3
4 #include "utility.h"
5 #include "variant.h"
6
7 namespace ticket {
8
27 template <typename ResultType, typename ErrorType>
28 class Result : public Variant<ResultType, ErrorType> {
29  public:
30   Result () = delete;
31   template <
32     typename T,
33     typename = std::enable_if_t<
34       std::is_constructible_v<ResultType, const T &> &&
35       !std::is_constructible_v<ErrorType, const T &>
36     >
37   >
38   Result (const T &value) : Variant<ResultType, ErrorType>(ResultType(value)) {}
39   template <
40     typename T,
41     typename = std::enable_if_t<
42       !std::is_constructible_v<ResultType, const T &> ||
43       std::is_same_v<ErrorType, T>
44     >,
45     typename = std::enable_if_t<std::is_constructible_v<ErrorType, const T &»
46   >
47   Result (const T &value) : Variant<ResultType, ErrorType>(ErrorType(value)) {}
48   auto result () -> ResultType & {
49     return *this->template get<ResultType>();
50   }
51   auto result () const -> const ResultType & {
52     return *this->template get<ResultType>();
53   }
54   auto error () -> ErrorType * {
55     return this->template get<ErrorType>();
56   }
57   auto error () const -> const ErrorType * {
58     return this->template get<ErrorType>();
59   }
60
62   auto success () const -> bool {
63     return this->index() == 0;
64   }
65 };
66
67 } // namespace ticket
68
69 #endif // TICKET_LIB_RESULT_H_
```

7.30 lib/utility.cpp File Reference

```
#include "utility.h"
```

Namespaces

- namespace ticket

Functions

- auto ticket::split (std::string &str, char sep) -> Vector< std::string_view >

  splits the string with sep into several substrings.
- auto ticket::copyStrings (const Vector< std::string_view > &vec) -> Vector< std::string >

  copies the strings in vec into an array of real strings.

7.31 lib/utility.h File Reference

```
#include <iostream>
#include "vector.h"
#include "internal/cmp.inc"
```

Classes

- struct ticket::Unit

  An empty class, used at various places.
- class ticket::Pair< T1, T2 >

  A pair of objects.
- class ticket::Triple< T1, T2, T3 >

  A triplet of objects.
- class ticket::Cmp< Lt >

  Comparison utilities.

Namespaces

- namespace ticket

Macros

- #define TICKET_ASSERT(x)

Typedefs

- template<typename Lt = internal::LessOp>
  using ticket::Less = Cmp< Lt >
- template<typename Lt = internal::LessOp>
  using ticket::Greater = Cmp< internal::GreaterOp< Lt > >

Functions

- auto ticket::split (std::string &str, char sep) -> Vector< std::string_view >

  splits the string with sep into several substrings.
- auto ticket::copyStrings (const Vector< std::string_view > &vec) -> Vector< std::string >

  copies the strings in vec into an array of real strings.
- template<typename T >
  auto ticket::declval () -> T

  declare value, used in type annotations.
- template<typename T >
  auto ticket::move (T &val) -> T &&

  forcefully make an rvalue.
- auto ticket::isVisibleChar (char ch) -> bool

Variables

- constexpr Unit ticket::unit

7.31.1 Macro Definition Documentation

7.31.1.1 TICKET_ASSERT `#define TICKET_ASSERT( x )`

7.32 utility.h

Go to the documentation of this file.

```
1 // This file defines several common utilities.
2 #ifndef TICKET_LIB_UTILITY_H_
3 #define TICKET_LIB_UTILITY_H_
4
5 // place this macro at the top to avoid cross-dep messing up
6 // the macro definition
7 #ifdef TICKET_DEBUG
8 #include <cassert>
9 #define TICKET_ASSERT(x) assert(x)
10 #else
11 #define TICKET_ASSERT(x)
12 #endif // TICKET_DEBUG
13
14 #include <iostream>
15
16 #include "vector.h"
17
18 namespace ticket {
19
31 auto split (std::string &str, char sep)
32   -> Vector<std::string_view>;
33
35 auto copyStrings (const Vector<std::string_view> &vec)
36   -> Vector<std::string>;
37
39 struct Unit {
40   constexpr Unit () = default;
41   template <typename T>
42   constexpr Unit (const T & /* unused */) {}
43   auto operator< (const Unit & /* unused */) -> bool {
44     return false;
45   }
46 };
47 inline constexpr Unit unit;
48
```

```
50 template <typename T>
51 auto declval () -> T;
52
54 template <typename T>
55 auto move (T &val) -> T && {
56   return reinterpret_cast<T &&>(val);
57 }
58
60 template <typename T1, typename T2>
61 class Pair {
62  public:
63   T1 first;
64   T2 second;
65   constexpr Pair () : first(), second() {}
66   Pair (const Pair &other) = default;
67   Pair (Pair &&other) noexcept = default;
68   Pair (const T1 &x, const T2 &y) : first(x), second(y) {}
69   template <class U1, class U2>
70   Pair (U1 &&x, U2 &&y) : first(x), second(y) {}
71   template <class U1, class U2>
72   Pair (const Pair<U1, U2> &other) : first(other.first), second(other.second) {}
73   template <class U1, class U2>
74   Pair (Pair<U1, U2> &&other) : first(other.first), second(other.second) {}
75 };
77 template <typename T1, typename T2, typename T3>
78 class Triple {
79  public:
80   T1 first;
81   T2 second;
82   T3 third;
83   constexpr Triple () : first(), second(), third() {}
84   Triple (const Triple &other) = default;
85   Triple (Triple &&other) noexcept = default;
86   Triple (const T1 &x, const T2 &y, const T3 &z) : first(x), second(y), third(z) {}
87 };
88
90 template <typename Lt>
91 class Cmp {
92  public:
93   template <typename T, typename U>
94   auto equals (const T &lhs, const U &rhs) -> bool {
95     return !lt_(lhs, rhs) && !lt_(rhs, lhs);
96   }
97   template <typename T, typename U>
98   auto ne (const T &lhs, const U &rhs) -> bool {
99     return !equals(lhs, rhs);
100   }
101   template <typename T, typename U>
102   auto lt (const T &lhs, const U &rhs) -> bool {
103     return lt_(lhs, rhs);
104   }
105   template <typename T, typename U>
106   auto gt (const T &lhs, const U &rhs) -> bool {
107     return lt_(rhs, lhs);
108   }
109   template <typename T, typename U>
110   auto leq (const T &lhs, const U &rhs) -> bool {
111     return !gt(lhs, rhs);
112   }
113   template <typename T, typename U>
114   auto geq (const T &lhs, const U &rhs) -> bool {
115     return !lt(lhs, rhs);
116   }
117  private:
118   Lt lt_;
119 };
120
121 #include "internal/cmp.inc"
122
123 template <typename Lt = internal::LessOp>
124 using Less = Cmp<Lt>;
125 template <typename Lt = internal::LessOp>
126 using Greater = Cmp<internal::GreaterOp<Lt>>;
127
128 inline auto isVisibleChar (char ch) -> bool {
129   return ch >= '\x21' && ch <= '\x7E';
130 }
131
132 } // namespace ticket
133
134 #endif // TICKET_LIB_UTILITY_H_
```

7.33 lib/variant.h File Reference

```
#include "internal/variant-impl.h"
#include "utility.h"
```

Classes

- class ticket::Variant< Ts >

  A tagged union, aka sum type.

Namespaces

- namespace ticket

7.34 variant.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_VARIANT_H_
2 #define TICKET_LIB_VARIANT_H_
3
4 #include "internal/variant-impl.h"
5 #include "utility.h"
6
7 namespace ticket {
8
18 template <typename ...Ts>
19 class Variant {
20  private:
21   using Traits = internal::VariantTraits<Ts...>;
22   using First = typename Traits::template NthType<0>;
23   using Second = typename Traits::template NthType<1>;
24   static constexpr size_t length = sizeof...(Ts);
25   static_assert(length >= 2);
26   static_assert(!Traits::hasDuplicates());
27  public:
28   Variant () : ix_(0), store_(internal::ctorIndex<0>) {}
32   template <typename T, int ix = Traits::template indexOf<T>()>
33   Variant (const T &value) :
34     ix_(ix),
35     store_(internal::ctorIndex<ix>, value) {
36     static_assert(Traits::template includes<T>());
37   }
38   Variant (const Variant &other) {
39     *this = other;
40   }
41   Variant (Variant &&other) noexcept { *this = move(other); }
42   // this class may be extended, so let it be virtual.
43   virtual ~Variant () {
44     destroy_();
45   }
46   auto operator= (const Variant &other) -> Variant & {
47     if (this == &other) return *this;
48     destroy_();
49     ix_ = other.ix_;
50     if constexpr (length == 2) {
51       if (ix_ == 0) new(&get_<First>()) First(other.get_<First>());
52       else new(&get_<Second>()) Second(other.get_<Second>());
53     } else {
54       other.visit([this] (auto &value) {
55         using T = std::remove_cvref_t<decltype(value)>;
56         new(&get_<T>()) T(value);
57       });
58     }
59     return *this;
60   }
61   auto operator= (Variant &&other) noexcept -> Variant & {
62     if (this == &other) return *this;
63     destroy_();
64     ix_ = other.ix_;
65     if constexpr (length == 2) {
```

```
66        if (ix_ == 0) new(&get_<First>()) First(move(other.get_<First>()));
67        else new(&get_<Second>()) Second(move(other.get_<Second>()));
68      } else {
69        other.visit([this] (auto &value) {
70          using T = decltype(value);
71          new(&get_<T>()) T(move(value));
72        });
73      }
74      return *this;
75    }
76
78    template <typename T, int ix = Traits::template indexOf<T>()>
79    auto operator= (const T &value) -> Variant & {
80      static_assert(Traits::template includes<T>());
81      destroy_();
82      ix_ = ix;
83      new(&get_<T>()) T(value);
84      return *this;
85    }
86
88    template <typename T>
89    auto is () const -> bool {
90      static_assert(Traits::template includes<T>());
91      return ix_ == Traits::template indexOf<T>();
92    }
94    auto index () const -> int {
95      return ix_;
96    }
97
99    template <typename T>
100    auto get () -> T * {
101      if (is<T>()) return &get_<T>();
102      return nullptr;
103    }
105    template <typename T>
106    auto get () const -> const T * {
107      if (is<T>()) return &get_<T>();
108      return nullptr;
109    }
111    template <int ix>
112    auto get () -> typename Traits::template NthType<ix> * {
113      if (ix_ != ix) return nullptr;
114      return &get_<typename Traits::template NthType<ix»();
115    }
117    template <int ix>
118    auto get () const -> const typename Traits::template NthType<ix> * {
119      if (ix_ != ix) return nullptr;
120      return &get_<typename Traits::template NthType<ix»();
121    }
122
132    template <typename Visitor>
133    auto visit (const Visitor &f) const -> void {
134      using Vt = typename Traits::template Vtable<Visitor>;
135      // sorry about the C-style cast here... it casts away const.
136      Vt::visit(ix_, f, (void *) &store_);
137    }
138
139  private:
140    int ix_ = -1;
141    typename Traits::Impl store_{internal::ctorValueless};
142
143    template <typename T = void>
144    auto get_ () -> T & {
145      return *reinterpret_cast<T *>(&store_);
146    }
147    template <typename T = void>
148    auto get_ () const -> const T & {
149      return *reinterpret_cast<const T *>(&store_);
150    }
151
152    auto destroy_ () -> void {
153      if (ix_ == -1) return;
154      if constexpr (length == 2) {
155        if (ix_ == 0) get_<First>().~First();
156        else get_<Second>().~Second();
157      } else {
158        visit([] (auto &value) {
159          using T = std::remove_reference_t<decltype(value)>;
160          value.~T();
161        });
162      }
163      ix_ = -1;
164    }
165  };
166
167  } // namespace ticket
168
```

```
169 #endif // TICKET_LIB_VARIANT_H_
```

7.35 lib/vector.h File Reference

```
#include <climits>
#include <cstddef>
#include <iterator>
#include "exception.h"
#include "utility.h"
```

Classes

- class ticket::Vector< T >

  A data container like std::vector.
- class ticket::Vector< T >::iterator
- class ticket::Vector< T >::const_iterator

Namespaces

- namespace ticket

7.36 vector.h

Go to the documentation of this file.

```
1 #ifndef TICKET_LIB_VECTOR_H_
2 #define TICKET_LIB_VECTOR_H_
3
4 #include <climits>
5 #include <cstddef>
6 #include <iterator>
7
8 #include "exception.h"
9 #include "utility.h"
10
11 namespace ticket {
12
18 template<typename T>
19 class Vector {
20  public:
21   class const_iterator;
22   class iterator {
23    public:
24     using difference_type = std::ptrdiff_t;
25     using value_type = T;
26     using pointer = T *;
27     using reference = T &;
28     using iterator_category = std::random_access_iterator_tag;
29
30    private:
31     Vector *home_;
32     pointer ptr_;
33     iterator (Vector *home, pointer ptr) : home_(home), ptr_(ptr) {}
34    public:
35     auto operator+ (const int &n) const -> iterator {
36       return iterator(home_, ptr_ + n);
37     }
38     auto operator- (const int &n) const -> iterator {
39       return iterator(home_, ptr_ - n);
40     }
41     // return the distance between two iterators,
42     // if these two iterators point to different vectors, throw invaild_iterator.
43     auto operator- (const iterator &rhs) const -> int {
44       if (home_ != rhs.home_) throw Exception("invalid operation");
45       return ptr_ - rhs.ptr_;
```

```
46      }
47      auto operator+= (const int &n) -> iterator & {
48        ptr_ += n;
49        return *this;
50      }
51      auto operator-= (const int &n) -> iterator & { return (*this += -n); }
52      auto operator++ (int) -> iterator {
53        auto res = *this;
54        ++*this;
55        return res;
56      }
57      auto operator++ () -> iterator & { return (*this += 1); }
58      auto operator-- (int) -> iterator {
59        auto res = *this;
60        --*this;
61        return res;
62      }
63      auto operator-- () -> iterator & { return (*this -= 1); }
64      auto operator* () const -> T & { return *ptr_; }
68      auto operator== (const iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
69      auto operator== (const const_iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
73      auto operator!= (const iterator &rhs) const -> bool { return !(*this == rhs); }
74      auto operator!= (const const_iterator &rhs) const -> bool { return !(*this == rhs); }
75      auto operator< (const iterator &rhs) const -> bool {
76        return **this < *rhs;
77      }
78      auto operator< (const const_iterator &rhs) const -> bool {
79        return **this < *rhs;
80      }
81      friend class const_iterator;
82      friend class Vector;
83    };
84    class const_iterator {
85     public:
86      using difference_type = std::ptrdiff_t;
87      using value_type = T;
88      using pointer = T *;
89      using reference = T &;
90      using iterator_category = std::random_access_iterator_tag;
91
92     private:
93      const Vector *home_;
94      const T *ptr_;
95      const_iterator (const Vector *home, pointer ptr) : home_(home), ptr_(ptr) {}
96     public:
97      auto operator+ (const int &n) const -> const_iterator {
98        return const_iterator(home_, ptr_ + n);
99      }
100     auto operator- (const int &n) const -> const_iterator {
101       return const_iterator(home_, ptr_ - n);
102     }
103     auto operator- (const const_iterator &rhs) const -> int {
104       if (home_ != rhs.home_) throw Exception("invalid operation");
105       return ptr_ - rhs.ptr_;
106     }
107     auto operator+= (const int &n) -> const_iterator & {
108       ptr_ += n;
109       return *this;
110     }
111     auto operator-= (const int &n) -> const_iterator & { return (*this += -n); }
112     auto operator++ (int) -> const_iterator {
113       auto res = *this;
114       ++*this;
115       return res;
116     }
117     auto operator++ () -> const_iterator & { return (*this += 1); }
118     auto operator-- (int) -> const_iterator {
119       auto res = *this;
120       --*this;
121       return res;
122     }
123     auto operator-- () -> const_iterator & { return (*this -= 1); }
124     auto operator* () const -> const T & { return *ptr_; }
125     auto operator== (const iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
126     auto operator== (const const_iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
127     auto operator!= (const iterator &rhs) const -> bool { return !(*this == rhs); }
128     auto operator!= (const const_iterator &rhs) const -> bool { return !(*this == rhs); }
129     auto operator< (const iterator &rhs) const -> bool {
130       return **this < *rhs;
131     }
132     auto operator< (const const_iterator &rhs) const -> bool {
133       return **this < *rhs;
134     }
135     friend class iterator;
136     friend class Vector;
137   };
138   Vector () = default;
```

```
139   Vector (const Vector &other) { *this = other; }
140   Vector (Vector &&other) noexcept { *this = move(other); }
141   ~Vector () {
142     destroyContents_();
143     delete[] reinterpret_cast<char *>(storage_);
144   }
145   auto operator= (const Vector &other) -> Vector & {
146     if (this == &other) return *this;
147     clear();
148     grow_(other.capacity_);
149     size_ = other.size_;
150     copyContents_(storage_, other.storage_, size_);
151     return *this;
152   }
153   auto operator= (Vector &&other) noexcept -> Vector & {
154     if (this == &other) return *this;
155     clear();
156     storage_ = other.storage_;
157     size_ = other.size_;
158     capacity_ = other.capacity_;
159     other.size_ = other.capacity_ = 0;
160     other.storage_ = nullptr;
161     return *this;
162   }
163
168   auto at (const size_t &pos) -> T & {
169     checkPosition_(pos);
170     return storage_[pos];
171   }
172   auto at (const size_t &pos) const -> const T & {
173     return const_cast<Vector *>(this)->at(pos);
174   }
181   auto operator[] (const size_t &pos) -> T & { return at(pos); }
182   auto operator[] (const size_t &pos) const -> const T & { return at(pos); }
187   auto front () const -> const T & {
188     checkNonEmpty_();
189     return at(0);
190   }
195   auto back () const -> const T & {
196     checkNonEmpty_();
197     return at(size_ - 1);
198   }
202   auto begin () -> iterator {
203     return iterator(this, storage_);
204   }
205   auto begin () const -> const_iterator { return cbegin(); }
206   auto cbegin () const -> const_iterator {
207     return const_iterator(this, storage_);
208   }
212   auto end () -> iterator {
213     return iterator(this, storage_ + size_);
214   }
215   auto end () const -> const_iterator { return cend(); }
216   auto cend () const -> const_iterator {
217     return const_iterator(this, storage_ + size_);
218   }
222   auto empty () const -> bool {
223     return size_ == 0;
224   }
228   auto size () const -> size_t {
229     return size_;
230   }
234   auto clear () -> void {
235     destroyContents_();
236     delete[] reinterpret_cast<char *>(storage_);
237     storage_ = nullptr;
238     capacity_ = 0;
239     size_ = 0;
240   }
245   auto insert (iterator pos, const T &value) -> iterator { return insert(pos.ptr_ - storage_, value); }
252   auto insert (const size_t &ix, const T &value) -> iterator {
253     if (ix > size_) throw OutOfBounds();
254     if (size_ == capacity_) grow_();
255     for (size_t i = size_; i > ix; --i) {
256       storage_[i] = move_(storage_[i - 1]);
257     }
258     storage_[ix] = value;
259     ++size_;
260     return iterator(this, storage_ + ix);
261   }
267   auto erase (iterator pos) -> iterator { return erase(pos.ptr_ - storage_); }
273   auto erase (const size_t &ix) -> iterator {
274     checkPosition_(ix);
275     for (size_t i = ix; i + 1 < size_; ++i) {
276       storage_[i] = move_(storage_[i + 1]);
277     }
278     (storage_ + size_ - 1)->~T();
```

```
279     --size_;
280     return iterator(this, storage_ + ix);
281   }
285   auto push_back (const T &value) -> void {
286     if (size_ == capacity_) grow_();
287     new(storage_ + size_) T(value);
288     ++size_;
289   }
294   auto pop_back () -> void {
295     checkNonEmpty_();
296     (storage_ + size_ - 1)->~T();
297     --size_;
298   }
299
300   auto reserve (size_t capacity) -> void {
301     if (capacity_ < capacity) grow_(capacity);
302   }
303
304   // TODO: docs
305   template <typename Functor>
306   auto map (const Functor &fn) const
307     -> Vector<decltype(fn(at(0)))> {
308     Vector<decltype(fn(at(0)))> res;
309     res.reserve(capacity_);
310     for (int i = 0; i < size_; ++i) {
311       res.push_back(fn(storage_[i]));
312     }
313     return res;
314   }
315
316   template <typename Functor>
317   auto reduce (const Functor &fn) const -> T {
318     TICKET_ASSERT(size_ > 0);
319     auto prev = storage_[0];
320     for (int i = 1; i < size_; ++i) {
321       prev = fn(prev, storage_[i]);
322     }
323     return prev;
324   }
325   template <typename Functor, typename Res>
326   auto reduce (const Functor &fn, const Res &init) const
327     -> Res {
328     auto prev = init;
329     for (int i = 0; i < size_; ++i) {
330       prev = fn(prev, storage_[i]);
331     }
332     return prev;
333   }
334
335  private:
336   static constexpr size_t kSzDefault_ = 4;
337   static constexpr size_t kSzT_ = sizeof(T);
338   T *storage_ = nullptr;
339   size_t capacity_ = 0;
340   size_t size_ = 0;
341
342   static auto move_ (T &el) -> T && { return reinterpret_cast<T &&>(el); }
343   static auto copyContents_ (T *to, T *from, size_t n) -> void {
344     for (size_t i = 0; i < n; ++i) {
345       to[i] = from[i];
346     }
347   }
348   static auto moveContents_ (T *to, T *from, size_t n) -> void {
349     for (size_t i = 0; i < n; ++i) {
350       new(to + i) T(move_(from[i]));
351       from[i].~T();
352     }
353   }
354   static auto destroyContents_ (T *array, size_t n) -> void {
355     for (size_t i = 0; i < n; ++i) {
356       (array + i)->~T();
357     }
358   }
359   auto destroyContents_ () -> void { destroyContents_(storage_, size_); }
360   auto grow_ (size_t capNew) -> void {
361     T *storeNew = reinterpret_cast<T *>(new char[capNew * kSzT_]);
362     if (storage_ != nullptr) {
363       moveContents_(storeNew, storage_, size_);
364       delete[] reinterpret_cast<char *>(storage_);
365     }
366     storage_ = storeNew;
367     capacity_ = capNew;
368   }
369   auto grow_ () -> void {
370     grow_(storage_ == nullptr ? kSzDefault_ : 2 * capacity_);
371   }
372   auto checkPosition_ (size_t pos) const -> void {
```

```
373     // since this is size_t which is unsigned, we could not have pos < 0.
374     if (pos >= size_) throw OutOfBounds();
375   }
376   auto checkNonEmpty_ () const -> void {
377     if (size_ == 0) throw OutOfBounds();
378   }
379 };
380
381 } // namespace ticket
382
383 #endif // TICKET_LIB_VECTOR_H_
```

7.37 src/main.cpp File Reference

```
#include <iostream>
#include "parser.h"
#include "response.h"
#include "rollback.h"
#include "utility.h"
```

Functions

- auto main () -> int

7.37.1 Function Documentation

7.37.1.1 main() `auto main ( ) -> int`

7.38 src/misc.cpp File Reference

```
#include "parser.h"
#include <iostream>
#include "order.h"
#include "rollback.h"
#include "train.h"
#include "user.h"
```

Namespaces

- namespace ticket

7.39 src/node.cpp File Reference

```
#include <napi.h>
#include "parser.h"
#include "response.h"
#include "rollback.h"
```

Functions

- template<typename T >
  auto execute (Napi::Env env, const T &cmd) -> Napi::Value
- auto handler (const Napi::CallbackInfo &info) -> Napi::Value
- auto init (Napi::Env env, Napi::Object exports) -> Napi::Object

7.39.1 Function Documentation

7.39.1.1 execute() template<typename T >

```
auto execute (
            Napi::Env env,
            const T & cmd ) -> Napi::Value
```

7.39.1.2 handler() auto handler (

```
            const Napi::CallbackInfo & info ) -> Napi::Value
```

7.39.1.3 init() auto init (

```
            Napi::Env env,
            Napi::Object exports ) -> Napi::Object
```

7.40 src/order.cpp File Reference

```
#include "order.h"
#include "algorithm.h"
#include "parser.h"
#include "rollback.h"
#include "user.h"
```

Namespaces

- namespace ticket

7.41 src/order.h File Reference

```
#include "file/file.h"
#include "file/index.h"
#include "train.h"
#include "user.h"
#include "variant.h"
```

Classes

- struct ticket::OrderCache
- struct ticket::OrderBase
- struct ticket::Order
- struct ticket::BuyTicketEnqueued

  Utility class to represent the result of a buy ticket request that a pending order has been created.
- struct ticket::BuyTicketSuccess

  Utility class to represent the result of a buy ticket request that the order has been processed.

Namespaces

- namespace ticket

Typedefs

- using ticket::BuyTicketResponse = Variant< BuyTicketSuccess, BuyTicketEnqueued >

7.42 order.h

Go to the documentation of this file.

```
1 #ifndef TICKET_ORDER_H_
2 #define TICKET_ORDER_H_
3
4 #include "file/file.h"
5 #include "file/index.h"
6 #include "train.h"
7 #include "user.h"
8 #include "variant.h"
9
10 namespace ticket {
11
12 struct OrderCache {
13   Train::Id trainId;
14   Station::Id from, to;
15   Instant timeDeparture, timeArrival;
16 };
17
18 struct OrderBase {
19   using Id = int;
20   enum Status { kSuccess, kPending, kRefunded };
22   inline static auto statusString (Status status)
23     -> const char * {
24     switch (status) {
25       case kSuccess: return "success";
26       case kPending: return "pending";
27       case kRefunded: return "refunded";
28     }
29   }
30
31   User::Id user;
32   Ride ride;
33   int ixFrom, ixTo;
34   int seats;
35   int price;
36   Status status;
37   OrderCache cache;
38
40   auto getTrain () -> Train;
41
42   static constexpr const char *filename = "orders";
43 };
44 struct Order : public file::Managed<OrderBase> {
45   Order () = default;
46   Order (const file::Managed<OrderBase> &order)
47     : file::Managed<OrderBase>(order) {}
48   static file::Index<User::Id, Order> ixUserId;
49   static file::Index<Ride, Order> pendingOrders;
50 };
51
```

```
58 struct BuyTicketEnqueued {};
65 struct BuyTicketSuccess {
66   int price;
67 };
68 using BuyTicketResponse = Variant<
69   BuyTicketSuccess,
70   BuyTicketEnqueued
71 >;
72
73 } // namespace ticket
74
75 #endif // TICKET_ORDER_H_
```

7.43 src/parser.cpp File Reference

```
#include "parser.h"
#include "utility.h"
```

Namespaces

- namespace ticket
- namespace ticket::command

  Classes and parsers for commands.

Functions

- auto ticket::command::parse (std::string &str) -> Result< Command, ParseException >

  parses the command stored in str.
- auto ticket::command::parse (const Vector< std::string_view > &argv) -> Result< Command, Parse↩ Exception >

7.44 src/parser.h File Reference

```
#include <iostream>
#include "datetime.h"
#include "exception.h"
#include "optional.h"
#include "variant.h"
#include "result.h"
#include "response.h"
```

Classes

- struct ticket::command::AddUser
- struct ticket::command::Login
- struct ticket::command::Logout
- struct ticket::command::QueryProfile
- struct ticket::command::ModifyProfile
- struct ticket::command::AddTrain
- struct ticket::command::DeleteTrain
- struct ticket::command::ReleaseTrain

- struct ticket::command::QueryTrain
- struct ticket::command::QueryTicket
- struct ticket::command::QueryTransfer
- struct ticket::command::BuyTicket
- struct ticket::command::QueryOrder
- struct ticket::command::RefundTicket
- struct ticket::command::Rollback
- struct ticket::command::Clean
- struct ticket::command::Exit

Namespaces

- namespace ticket
- namespace ticket::command

  Classes and parsers for commands.

Typedefs

- using ticket::command::Command = Variant< AddUser, Login, Logout, QueryProfile, ModifyProfile, AddTrain, DeleteTrain, ReleaseTrain, QueryTrain, QueryTicket, QueryTransfer, BuyTicket, QueryOrder, RefundTicket, Rollback, Clean, Exit >

Enumerations

- enum ticket::command::SortType { ticket::command::kTime , ticket::command::kCost }

Functions

- auto ticket::command::parse (std::string &str) -> Result< Command, ParseException >

  parses the command stored in str.
- auto ticket::command::parse (const Vector< std::string_view > &argv) -> Result< Command, ParseException >
- auto ticket::command::run (const AddUser &cmd) -> Result< Response, Exception >

  Visitor for the commands.
- auto ticket::command::run (const Login &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const Logout &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const QueryProfile &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const ModifyProfile &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const AddTrain &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const DeleteTrain &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const ReleaseTrain &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const QueryTrain &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const QueryTicket &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const QueryTransfer &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const BuyTicket &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const QueryOrder &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const RefundTicket &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const Rollback &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const Clean &cmd) -> Result< Response, Exception >
- auto ticket::command::run (const Exit &cmd) -> Result< Response, Exception >

7.45 parser.h

Go to the documentation of this file.

```
1 // This file is autogenerated. Do not modify.
2 #ifndef TICKET_PARSER_H_
3 #define TICKET_PARSER_H_
4
5 #include <iostream>
6
7 #include "datetime.h"
8 #include "exception.h"
9 #include "optional.h"
10 #include "variant.h"
11 #include "result.h"
12 #include "response.h"
13
15 namespace ticket::command {
16
17 enum SortType { kTime, kCost };
18
19 struct AddUser {
20   Optional<std::string> currentUser;
21   std::string username;
22   std::string password;
23   std::string name;
24   std::string email;
25   Optional<int> privilege;
26 };
27
28 struct Login {
29   std::string username;
30   std::string password;
31 };
32
33 struct Logout {
34   std::string username;
35 };
36
37 struct QueryProfile {
38   std::string currentUser;
39   std::string username;
40 };
41
42 struct ModifyProfile {
43   std::string currentUser;
44   std::string username;
45   Optional<std::string> password;
46   Optional<std::string> name;
47   Optional<std::string> email;
48   Optional<int> privilege;
49 };
50
51 struct AddTrain {
52   std::string id;
53   int stops;
54   int seats;
55   Vector<std::string> stations;
56   Vector<int> prices;
57   Instant departure;
58   Vector<Duration> durations;
59   Vector<Duration> stopoverTimes;
60   Vector<Date> dates;
61   char type;
62 };
63
64 struct DeleteTrain {
65   std::string id;
66 };
67
68 struct ReleaseTrain {
69   std::string id;
70 };
71
72 struct QueryTrain {
73   std::string id;
74   Date date;
75 };
76
77 struct QueryTicket {
78   std::string from;
79   std::string to;
80   Date date;
81   SortType sort = kTime;
82 };
83
84 struct QueryTransfer {
```

```
85   std::string from;
86   std::string to;
87   Date date;
88   SortType sort = kTime;
89 };
90
91 struct BuyTicket {
92   std::string currentUser;
93   std::string train;
94   Date date;
95   int seats;
96   std::string from;
97   std::string to;
98   bool queue = false;
99 };
100
101 struct QueryOrder {
102   std::string currentUser;
103 };
104
105 struct RefundTicket {
106   std::string currentUser;
107   int index = 1;
108 };
109
110 struct Rollback {
111   int timestamp;
112 };
113
114 struct Clean {
115
116 };
117
118 struct Exit {
119
120 };
121
122
123 using Command = Variant<
124   AddUser,
125   Login,
126   Logout,
127   QueryProfile,
128   ModifyProfile,
129   AddTrain,
130   DeleteTrain,
131   ReleaseTrain,
132   QueryTrain,
133   QueryTicket,
134   QueryTransfer,
135   BuyTicket,
136   QueryOrder,
137   RefundTicket,
138   Rollback,
139   Clean,
140   Exit
141 >;
142
148 auto parse (std::string &str)
149   -> Result<Command, ParseException>;
150 auto parse (const Vector<std::string_view> &argv)
151   -> Result<Command, ParseException>;
152
163 auto run (const AddUser &cmd) -> Result<Response, Exception>;
164 auto run (const Login &cmd) -> Result<Response, Exception>;
165 auto run (const Logout &cmd) -> Result<Response, Exception>;
166 auto run (const QueryProfile &cmd) -> Result<Response, Exception>;
167 auto run (const ModifyProfile &cmd) -> Result<Response, Exception>;
168 auto run (const AddTrain &cmd) -> Result<Response, Exception>;
169 auto run (const DeleteTrain &cmd) -> Result<Response, Exception>;
170 auto run (const ReleaseTrain &cmd) -> Result<Response, Exception>;
171 auto run (const QueryTrain &cmd) -> Result<Response, Exception>;
172 auto run (const QueryTicket &cmd) -> Result<Response, Exception>;
173 auto run (const QueryTransfer &cmd) -> Result<Response, Exception>;
174 auto run (const BuyTicket &cmd) -> Result<Response, Exception>;
175 auto run (const QueryOrder &cmd) -> Result<Response, Exception>;
176 auto run (const RefundTicket &cmd) -> Result<Response, Exception>;
177 auto run (const Rollback &cmd) -> Result<Response, Exception>;
178 auto run (const Clean &cmd) -> Result<Response, Exception>;
179 auto run (const Exit &cmd) -> Result<Response, Exception>;
180
181 } // namespace ticket::command
182
183 #endif // TICKET_PARSER_H_
```

7.46 src/response.cpp File Reference

```
#include "response.h"
#include <iostream>
```

Namespaces

- namespace ticket
- namespace ticket::response

Functions

- auto ticket::response::cout (const Unit &) -> void
- auto ticket::response::cout (const User &user) -> void
- auto ticket::response::cout (const Train &train) -> void
- auto ticket::response::cout (const Vector< Train > &trains) -> void
- auto ticket::response::cout (const BuyTicketResponse &ticket) -> void
- auto ticket::response::cout (const Vector< Order > &orders) -> void

7.47 src/response.h File Reference

```
#include "order.h"
#include "train.h"
#include "user.h"
#include "utility.h"
#include "variant.h"
```

Namespaces

- namespace ticket
- namespace ticket::response

Typedefs

- using ticket::Response = Variant< Unit, User, Train, Vector< Train >, BuyTicketResponse, Vector< Order > >

Functions

- auto ticket::response::cout (const Unit &) -> void
- auto ticket::response::cout (const User &user) -> void
- auto ticket::response::cout (const Train &train) -> void
- auto ticket::response::cout (const Vector< Train > &trains) -> void
- auto ticket::response::cout (const BuyTicketResponse &ticket) -> void
- auto ticket::response::cout (const Vector< Order > &orders) -> void

7.48 response.h

Go to the documentation of this file.

```
1 // TODO: docs
2 #ifndef TICKET_RESPONSE_H_
3 #define TICKET_RESPONSE_H_
4
5 #ifdef BUILD_NODEJS
6 #include <napi.h>
7 #endif // BUILD_NODEJS
8
9 #include "order.h"
10 #include "train.h"
11 #include "user.h"
12 #include "utility.h"
13 #include "variant.h"
14
15 namespace ticket {
16
17 using Response = Variant<
18   Unit,
19   User,
20   Train,
21   Vector<Train>,
22   BuyTicketResponse,
23   Vector<Order>
24   // the exit command does not need a response object.
25 >;
26
27 namespace response {
28
29 auto cout (const Unit & /* unused */) -> void;
30 auto cout (const User &user) -> void;
31 auto cout (const Train &train) -> void;
32 auto cout (const Vector<Train> &trains) -> void;
33 auto cout (const BuyTicketResponse &ticket) -> void;
34 auto cout (const Vector<Order> &orders) -> void;
35
36 #ifdef BUILD_NODEJS
37
38 auto toJsObject (Napi::Env env, const Unit & /* unused */) -> Napi::Object;
39 auto toJsObject (Napi::Env env, const User &user) -> Napi::Object;
40 auto toJsObject (Napi::Env env, const Train &train) -> Napi::Object;
41 auto toJsObject (Napi::Env env, const Vector<Train> &trains) -> Napi::Object;
42 auto toJsObject (Napi::Env env, const BuyTicketResponse &ticket) -> Napi::Object;
43 auto toJsObject (Napi::Env env, const Vector<Order> &orders) -> Napi::Object;
44
45 #endif // BUILD_NODEJS
46
47 } // namespace response
48
49 } // namespace ticket
50
51 #endif // TICKET_RESPONSE_H_
```

7.49 src/rollback.cpp File Reference

```
#include "rollback.h"
#include "parser.h"
```

Namespaces

- namespace ticket

Functions

- auto ticket::setTimestamp (int timestamp) -> void

  sets the current timestamp.

7.50 src/rollback.h File Reference

```
#include "file/file.h"
#include "optional.h"
#include "order.h"
#include "result.h"
#include "train.h"
#include "user.h"
#include "variant.h"
```

Classes

- struct ticket::rollback::AddUser
- struct ticket::rollback::ModifyProfile
- struct ticket::rollback::AddTrain
- struct ticket::rollback::DeleteTrain
- struct ticket::rollback::ReleaseTrain
- struct ticket::rollback::BuyTicket
- struct ticket::rollback::RefundTicket
- struct ticket::rollback::FulfillOrder
- struct ticket::rollback::LogEntryBase

Namespaces

- namespace ticket
- namespace ticket::rollback

Typedefs

- using ticket::rollback::LogEntry = file::Managed< LogEntryBase >

Functions

- auto ticket::setTimestamp (int timestamp) -> void

  sets the current timestamp.
- auto ticket::rollback::log (const LogEntry::Content &content) -> void

  inserts a log entry.
- auto ticket::rollback::run (const AddUser &log) -> Result< Unit, Exception >

  Visitor for the log entries.
- auto ticket::rollback::run (const ModifyProfile &log) -> Result< Unit, Exception >
- auto ticket::rollback::run (const AddTrain &log) -> Result< Unit, Exception >
- auto ticket::rollback::run (const DeleteTrain &log) -> Result< Unit, Exception >
- auto ticket::rollback::run (const ReleaseTrain &log) -> Result< Unit, Exception >
- auto ticket::rollback::run (const BuyTicket &log) -> Result< Unit, Exception >
- auto ticket::rollback::run (const RefundTicket &log) -> Result< Unit, Exception >
- auto ticket::rollback::run (const FulfillOrder &log) -> Result< Unit, Exception >

7.51 rollback.h

Go to the documentation of this file.

```
1 #ifndef TICKET_BACKLOG_H_
2 #define TICKET_BACKLOG_H_
3
4 #include "file/file.h"
5 #include "optional.h"
6 #include "order.h"
7 #include "result.h"
8 #include "train.h"
9 #include "user.h"
10 #include "variant.h"
11
12 namespace ticket {
14 auto setTimestamp (int timestamp) -> void;
15 } // namespace ticket
16
17
18 namespace ticket::rollback {
19
20 struct AddUser {
21   int id;
22 };
23
24 struct ModifyProfile {
25   int id;
26   Optional<User::Password> password;
27   Optional<User::Name> name;
28   Optional<User::Email> email;
29   Optional<User::Privilege> privilege;
30 };
31
32 struct AddTrain {
33   int id;
34 };
35
36 struct DeleteTrain {
37   int id;
38 };
39
40 struct ReleaseTrain {
41   int id;
42 };
43
44 struct BuyTicket {
45   int id;
46 };
47
48 struct RefundTicket {
49   int id;
50   Order::Status status;
51 };
52
53 // this is not a command, but rather used in ticket refunds.
54 struct FulfillOrder {
55   int id;
56 };
57
58 struct LogEntryBase {
59   using Content = Variant<
60     AddUser,
61     ModifyProfile,
62     AddTrain,
63     DeleteTrain,
64     ReleaseTrain,
65     BuyTicket,
66     RefundTicket,
67     FulfillOrder
68   >;
69
70   int timestamp;
71   Content content;
72
73   static constexpr const char *filename = "rollback-log";
74 };
75 using LogEntry = file::Managed<LogEntryBase>;
76
78 auto log (const LogEntry::Content &content) -> void;
79
86 auto run (const AddUser &log) -> Result<Unit, Exception>;
87 auto run (const ModifyProfile &log) -> Result<Unit, Exception>;
88 auto run (const AddTrain &log) -> Result<Unit, Exception>;
89 auto run (const DeleteTrain &log) -> Result<Unit, Exception>;
90 auto run (const ReleaseTrain &log) -> Result<Unit, Exception>;
91 auto run (const BuyTicket &log) -> Result<Unit, Exception>;
```

```
92 auto run (const RefundTicket &log) -> Result<Unit, Exception>;
93 auto run (const FulfillOrder &log) -> Result<Unit, Exception>;
94
95 } // namespace ticket::rollback
96
97 #endif // TICKET_BACKLOG_H_
```

7.52 src/train.cpp File Reference

```
#include "train.h"
#include "parser.h"
#include "rollback.h"
#include "utility.h"
```

Namespaces

- namespace ticket

7.53 src/train.h File Reference

```
#include "datetime.h"
#include "exception.h"
#include "file/array.h"
#include "file/bptree.h"
#include "file/file.h"
#include "file/index.h"
#include "file/varchar.h"
#include "optional.h"
```

Classes

- struct ticket::TrainBase
- struct ticket::TrainBase::Stop
- struct ticket::TrainBase::Edge
- struct ticket::Train
- struct ticket::Ride
- struct ticket::RideSeatsBase
- struct ticket::RideSeats

Namespaces

- namespace ticket
- namespace ticket::Station

Typedefs

- using ticket::Station::Id = file::Varchar< 30 >

7.54 train.h

Go to the documentation of this file.

```
1 #ifndef TICKET_TRAIN_H_
2 #define TICKET_TRAIN_H_
3
4 #include "datetime.h"
5 #include "exception.h"
6 #include "file/array.h"
7 #include "file/bptree.h"
8 #include "file/file.h"
9 #include "file/index.h"
10 #include "file/varchar.h"
11 #include "optional.h"
12
13 namespace ticket {
14
15 namespace Station {
16 using Id = file::Varchar<30>;
17 } // namespace Station
18
19 struct RideSeats;
20
21 struct TrainBase {
22   using Id = file::Varchar<20>;
23   using Type = char;
24   struct Stop {
25     Station::Id name;
26   };
27   struct Edge {
28     int price;
29     Instant departure;
30     Instant arrival;
31   };
32
33   Id trainId;
34   file::Array<Stop, 100> stops;
35   file::Array<Edge, 99> edges;
36   int seats;
37   Date begin, end;
38   Type type;
39   bool released = false;
40   bool deleted = false;
41
42   static constexpr const char *filename = "trains";
43 };
44 struct Train : public file::Managed<TrainBase> {
45   Train () = default;
46   Train (const file::Managed<TrainBase> &train)
47     : file::Managed<TrainBase>(train) {}
48   static file::Index<Train::Id, Train> ixId;
49   static file::BpTree<size_t, int> ixStop;
50
52   auto indexOfStop (const std::string &name) const
53     -> Optional<int>;
55   auto totalPrice (int ixFrom, int ixTo) const -> int;
56
62   auto getRide (Date date) const -> Optional<RideSeats>;
69   auto getRide (Date date, int ixDeparture) const
70     -> Optional<RideSeats>;
71 };
72
73
74 struct Ride {
76   int train;
77   Date date;
78
79   auto operator< (const Ride &rhs) const -> bool;
80 };
81
82 struct RideSeatsBase {
83   Ride ride;
84   file::Array<int, 99> seatsRemaining;
85
91   auto ticketsAvailable (int ixFrom, int ixTo) const -> int;
93   auto rangeAdd (int dx, int ixFrom, int ixTo) -> void;
94
95   static constexpr const char *filename = "ride-seats";
96 };
97 struct RideSeats : public file::Managed<RideSeatsBase> {
98   RideSeats () = default;
99   RideSeats (const file::Managed<RideSeatsBase> &rideSeats)
100     : file::Managed<RideSeatsBase>(rideSeats) {}
101   static file::Index<Ride, RideSeats> ixRide;
102 };
103
```

```
104 } // namespace ticket
105
106 #endif // TICKET_TRAIN_H_
```

7.55 src/user.cpp File Reference

```
#include "user.h"
#include <iostream>
#include "hashmap.h"
#include "parser.h"
#include "rollback.h"
```

Namespaces

- namespace ticket

Macros

- #define TICKET_CHECK_FIELD(name)
- #define TICKET_CHECK_FIELD(name) if (log.name) user.name = ∗log.name;

Functions

- auto ticket::makeUser (const command::AddUser &cmd) -> User
- template<typename Cmd >
  auto ticket::checkUser (const Cmd &cmd) -> Result< Pair< User, User >, Exception >

Variables

- HashMap< std::string, Unit > ticket::usersLoggedIn
  a set of users that are logged in.

7.55.1 Macro Definition Documentation

7.55.1.1 TICKET_CHECK_FIELD [1/2] #define TICKET_CHECK_FIELD(*name*)

Value:

```
  if (cmd.name) { \
    log.name = target.name; \
    target.name = *cmd.name; \
  }
```

7.55.1.2 TICKET_CHECK_FIELD [2/2] #define TICKET_CHECK_FIELD(name) if (log.name) user.name = *log.name;

7.56 src/user.h File Reference

```
#include "file/file.h"
#include "file/index.h"
#include "file/varchar.h"
```

Classes

- struct ticket::UserBase
- struct ticket::User

Namespaces

- namespace ticket

7.57 user.h

Go to the documentation of this file.

```
1 #ifndef TICKET_USER_H_
2 #define TICKET_USER_H_
3
4 #include "file/file.h"
5 #include "file/index.h"
6 #include "file/varchar.h"
7
8 namespace ticket {
9
10 struct UserBase {
11   using Id = file::Varchar<20>;
12   using Password = file::Varchar<30>;
13   using Name = file::Varchar<15>;
14   using Email = file::Varchar<30>;
15   using Privilege = int;
16
17   Id username;
18   Password password;
19   Name name;
20   Email email;
21   Privilege privilege;
22
24   static auto has (const char *username) -> bool;
26   static auto isLoggedIn (const std::string &username)
27     -> bool;
28
29   static constexpr const char *filename = "users";
30 };
31 struct User : public file::Managed<UserBase> {
32   User () = default;
33   User (const file::Managed<UserBase> &user)
34     : file::Managed<UserBase>(user) {}
35   static file::Index<User::Id, User> ixUsername;
36 };
37
38 } // namespace ticket
39
40 #endif // TICKET_USER_H_
```

Index