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
- namespace strings

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 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
- 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, 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 isValidUsername (const std::string &username) -> bool
- auto isValidPassword (const std::string &password) -> bool
- auto isValidName (const std::string &name) -> bool
- auto isValidEmail (const std::string &email) -> bool
- auto isValidPrivilege (int privilege) -> bool
- auto isValidAddUser (const command::AddUser &cmd) -> bool
- auto makeUser (const command::AddUser &cmd) -> User
- auto insufficientPrivileges (const User &op, const User &target) -> bool
- 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, Order >
```

5.1.3 Function Documentation

5.1.3.1 copyStrings()

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

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

5.1.3.2 declval()

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

declare value, used in type annotations.

5.1.3.3 insufficientPrivileges()

```
auto ticket::insufficientPrivileges (
        const User & op,
        const User & target ) -> bool   [inline]
```

5.1.3.4 isValidAddUser()

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

5.1.3.5 isValidEmail()

```
auto ticket::isValidEmail (
        const std::string & email ) -> bool   [inline]
```

5.1.3.6 isValidName()

```
auto ticket::isValidName (
        const std::string & name ) -> bool   [inline]
```

5.1.3.7 isValidPassword()

```
auto ticket::isValidPassword (
        const std::string & password ) -> bool   [inline]
```

5.1.3.8 isValidPrivilege()

```
auto ticket::isValidPrivilege (
        int privilege ) -> bool   [inline]
```

5.1.3.9 isValidUsername()

```
auto ticket::isValidUsername (
        const std::string & username ) -> bool   [inline]
```

5.1.3.10 isVisibleChar()

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

5.1.3.11 makeUser()

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

5.1.3.12 move()

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

forcefully make an rvalue.

5.1.3.13 setTimestamp() `auto ticket::setTimestamp ( int timestamp ) -> void`

sets the current timestamp.

5.1.3.14 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 dispatch (const AddUser &cmd) -> Result< Response, Exception >

  Visitor for the commands.
- auto dispatch (const Login &cmd) -> Result< Response, Exception >
- auto dispatch (const Logout &cmd) -> Result< Response, Exception >
- auto dispatch (const QueryProfile &cmd) -> Result< Response, Exception >
- auto dispatch (const ModifyProfile &cmd) -> Result< Response, Exception >
- auto dispatch (const AddTrain &cmd) -> Result< Response, Exception >
- auto dispatch (const DeleteTrain &cmd) -> Result< Response, Exception >
- auto dispatch (const ReleaseTrain &cmd) -> Result< Response, Exception >
- auto dispatch (const QueryTrain &cmd) -> Result< Response, Exception >
- auto dispatch (const QueryTicket &cmd) -> Result< Response, Exception >
- auto dispatch (const QueryTransfer &cmd) -> Result< Response, Exception >
- auto dispatch (const BuyTicket &cmd) -> Result< Response, Exception >
- auto dispatch (const QueryOrder &cmd) -> Result< Response, Exception >
- auto dispatch (const RefundTicket &cmd) -> Result< Response, Exception >
- auto dispatch (const Rollback &cmd) -> Result< Response, Exception >
- auto dispatch (const Clean &cmd) -> Result< Response, Exception >
- auto dispatch (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 dispatch() [1/17]

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

5.2.4.2 dispatch() [2/17]

```
auto ticket::command::dispatch (
        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.

5.2.4.3 dispatch() [3/17]

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

5.2.4.4 dispatch() [4/17]

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

5.2.4.5 dispatch() [5/17]

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

5.2.4.6 dispatch() [6/17]

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

5.2.4.7 dispatch() [7/17]

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

5.2.4.8 dispatch() [8/17]

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

5.2.4.9 dispatch() [9/17]

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

5.2.4.10 dispatch() [10/17]

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

5.2.4.11 dispatch() [11/17]

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

5.2.4.12 dispatch() [12/17]

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

5.2.4.13 dispatch() [13/17]

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

5.2.4.14 dispatch() [14/17]

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

5.2.4.15 dispatch() [15/17]

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

5.2.4.16 dispatch() [16/17]

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

5.2.4.17 dispatch() [17/17]

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

5.2.4.18 parse() [1/2]

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

5.2.4.19 parse() [2/2]

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

parses the command stored in str.

this function is autogenerated.

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 Order &order) -> 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 Order & order ) -> void
```

5.4.1.3 cout() [3/6]

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

5.4.1.4 cout() [4/6]

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

5.4.1.5 cout() [5/6]

```
auto ticket::response::cout (
        const User & user ) -> 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 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 dispatch (const AddUser &log) -> Result< Unit, Exception >

  Visitor for the log entries.
- auto dispatch (const ModifyProfile &log) -> Result< Unit, Exception >
- auto dispatch (const AddTrain &log) -> Result< Unit, Exception >
- auto dispatch (const DeleteTrain &log) -> Result< Unit, Exception >
- auto dispatch (const ReleaseTrain &log) -> Result< Unit, Exception >
- auto dispatch (const BuyTicket &log) -> Result< Unit, Exception >
- auto dispatch (const RefundTicket &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 dispatch() `[1/7]`

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

5.5.2.2 dispatch() [2/7]

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

Visitor for the log entries.

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

5.5.2.3 dispatch() [3/7]

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

5.5.2.4 dispatch() [4/7]

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

5.5.2.5 dispatch() [5/7]

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

5.5.2.6 dispatch() [6/7]

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

5.5.2.7 dispatch() [7/7]

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

5.5.2.8 log()

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

inserts a log entry.

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>`

5.7 ticket::strings Namespace Reference

Variables

- constexpr const char * kSuccess = "0\n"
- constexpr const char * kFail = "-1\n"

5.7.1 Variable Documentation

5.7.1.1 kFail `constexpr const char* ticket::strings::kFail = "-1\n" [constexpr]`

5.7.1.2 kSuccess `constexpr const char* ticket::strings::kSuccess = "0\n" [constexpr]`

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.

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.

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::output_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 -> const_iterator
- auto operator++ () -> const_iterator &
- auto operator-- (int) const -> 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::output_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 -> 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 -> 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:

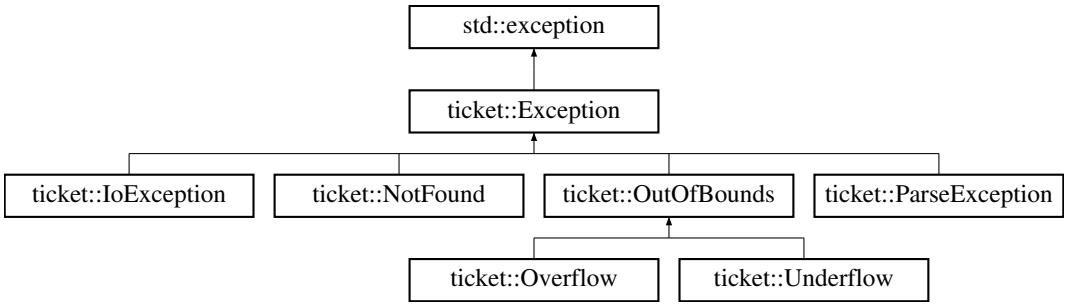
- src/train.h

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)
- virtual ~Exception ()=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() `virtual ticket::Exception::~Exception ( ) [virtual], [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.

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.

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

- lib/file/file.h

6.23 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

6.23.1 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.

6.23.2 Member Typedef Documentation

6.23.2.1 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>
```

6.23.3 Constructor & Destructor Documentation

6.23.3.1 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]
```

6.23.3.2 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]
```

6.23.3.3 ~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]
```

6.23.4 Member Function Documentation

6.23.4.1 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'

6.23.4.2 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]
```

6.23.4.3 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

6.23.4.4 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]
```

6.23.4.5 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]
```

6.23.4.6 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

6.23.4.7 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.

6.23.4.8 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.

6.23.4.9 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

6.23.4.10 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

6.23.4.11 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)

6.23.4.12 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.

6.23.4.13 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]
```

6.23.4.14 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.

6.23.4.15 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]
```

6.23.4.16 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.

6.23.4.17 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.

6.23.4.18 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

6.24 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.

6.24.1 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.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 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.

6.24.3 Member Function Documentation

6.24.3.1 empty() `template<typename Key , typename Model >`

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

checks if the index is empty.

6.24.3.2 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.

6.24.3.3 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.

6.24.3.4 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.

6.24.3.5 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.

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

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

inserts an object into the index.

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

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

removes an object from the index.

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

- lib/file/index.h

6.25 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.

6.25.1 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.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 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.

6.25.3 Member Function Documentation

6.25.3.1 empty()

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

checks if the index is empty.

6.25.3.2 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.

6.25.3.3 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.

6.25.3.4 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.

6.25.3.5 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.

6.25.3.6 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.

6.25.3.7 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.

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

- lib/file/index.h

6.26 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

6.26.1 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.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Instant() [1/3] `ticket::Instant::Instant ( )  [default]`

6.26.2.2 Instant() [2/3] ticket::Instant::Instant (

```
int hour,
int minute )
```

6.26.2.3 Instant() [3/3] ticket::Instant::Instant (

```
const char * str )  [explicit]
```

constructs an Instant from an HH:MM format string.

6.26.3 Member Function Documentation

6.26.3.1 daysOverflow() auto ticket::Instant::daysOverflow ( ) const -> int

6.26.3.2 hour() auto ticket::Instant::hour ( ) const -> int

6.26.3.3 minute() auto ticket::Instant::minute ( ) const -> int

6.26.3.4 operator std::string() ticket::Instant::operator std::string ( ) const

gets an HH:MM representation of the Instant.

6.26.3.5 operator+() auto ticket::Instant::operator+ (

```
Duration dt ) const -> Instant
```

6.26.3.6 operator-() [1/2] auto ticket::Instant::operator- (

```
Duration dt ) const -> Instant
```

6.26.3.7 operator-() [2/2] auto ticket::Instant::operator- (

```
Instant rhs ) const -> Duration
```

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

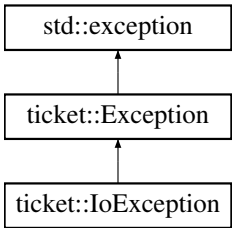
The documentation for this class was generated from the following files:

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

6.27 ticket::IoException Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::IoException:



Public Member Functions

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

6.27.1 Constructor & Destructor Documentation

6.27.1.1 IoException() [1/2] `ticket::IoException::IoException ( ) [inline]`

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

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

- lib/exception.h

6.28 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

6.28.1 Member Typedef Documentation

6.28.1.1 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
```

6.28.1.2 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
```

6.28.1.3 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 *
```

6.28.1.4 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 &
```

6.28.1.5 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
```

6.28.2 Constructor & Destructor Documentation

6.28.2.1 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]
```

6.28.2.2 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]
```

6.28.3 Member Function Documentation

6.28.3.1 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]
```

6.28.3.2 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]
```

6.28.3.3 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]
```

6.28.3.4 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]
```

6.28.3.5 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]
```

6.28.3.6 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]
```

6.28.3.7 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]
```

6.28.3.8 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]
```

6.28.3.9 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]
```

6.28.3.10 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]
```

6.28.4 Friends And Related Function Documentation

6.28.4.1 const_iterator

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

6.28.4.2 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

6.29 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::output_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) const -> iterator
- auto operator++ () -> iterator &
- auto operator-- (int) const -> 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

6.29.1 Member Typedef Documentation

6.29.1.1 difference_type template<typename T >

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

6.29.1.2 iterator_category template<typename T >

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

6.29.1.3 pointer template<typename T >

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

6.29.1.4 reference template<typename T >

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

6.29.1.5 value_type template<typename T >

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

6.29.2 Member Function Documentation

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

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

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

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

some other operator for iterator.

6.29.2.3 operator*() template<typename T >

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

6.29.2.4 operator+() template<typename T >

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

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

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

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

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

6.29.2.7 operator+=() template<typename T >

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

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

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

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

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

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

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

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

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

6.29.2.12 operator-=() template<typename T >

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

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

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

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

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

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

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

6.29.2.16 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).

6.29.3 Friends And Related Function Documentation

6.29.3.1 const_iterator template<typename T >

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

6.29.3.2 Vector template<typename T >

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

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

- lib/vector.h

6.30 ticket::rollback::LogEntryBase Struct Reference

`#include <rollback.h>`

Public Types

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

Public Attributes

- int timestamp
- Content content

Static Public Attributes

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

6.30.1 Member Typedef Documentation

6.30.1.1 Content

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

6.30.2 Member Data Documentation

6.30.2.1 content

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

6.30.2.2 filename

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

6.30.2.3 timestamp

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

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

- src/rollback.h

6.31 ticket::command::Login Struct Reference

`#include <parser.h>`

Public Attributes

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

6.31.1 Member Data Documentation

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

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

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

- src/parser.h

6.32 ticket::command::Logout Struct Reference

`#include <parser.h>`

Public Attributes

- std::string username

6.32.1 Member Data Documentation

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

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

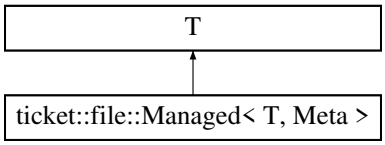
- src/parser.h

6.33 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 -> size_t
  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 Public Attributes

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

6.33.1 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.

6.33.2 Member Function Documentation

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

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

removes the object from the file.

6.33.2.2 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.

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

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

the unique immutable numeral identifier of the object.

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

6.33.2.4 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().

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

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

updates a modified object.

6.33.3 Member Data Documentation

6.33.3.1 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

6.34 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

6.34.1 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.

6.34.2 Member Typedef Documentation

6.34.2.1 const_iterator

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

6.34.2.2 iterator

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

6.34.2.3 value_type

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

6.34.3 Constructor & Destructor Documentation

6.34.3.1 Map()

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

6.34.4 Member Function Documentation

6.34.4.1 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'

6.34.4.2 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]
```

6.34.4.3 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

6.34.4.4 cbegin()

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

6.34.4.5 cend()

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

6.34.4.6 clear()

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

clears the contents

6.34.4.7 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)

6.34.4.8 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.

6.34.4.9 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.

6.34.4.10 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)

6.34.4.11 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.

6.34.4.12 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]
```

6.34.4.13 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.

6.34.4.14 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.

6.34.4.15 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.

6.34.4.16 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

6.35 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

6.35.1 Member Data Documentation

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

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

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

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

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

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

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

- src/parser.h

6.36 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

6.36.1 Member Data Documentation

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

6.36.1.2 id `int ticket::rollback::ModifyProfile::id`

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

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

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

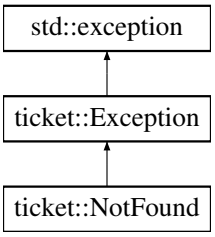
The documentation for this struct was generated from the following file:

- src/rollback.h

6.37 ticket::NotFound Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::NotFound:



Public Member Functions

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

6.37.1 Constructor & Destructor Documentation

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

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

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

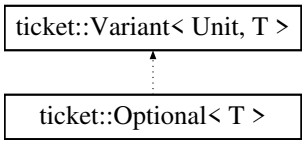
- lib/exception.h

6.38 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 *

6.38.1 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.

6.38.2 Constructor & Destructor Documentation

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

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

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

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

constructs a empty optional.

6.38.2.3 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.

6.38.3 Member Function Documentation

6.38.3.1 operator bool() template<typename T >

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

true if the optional has value.

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

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

provides access to the actual object.

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

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

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

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

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

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

6.38.3.6 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]
```

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

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

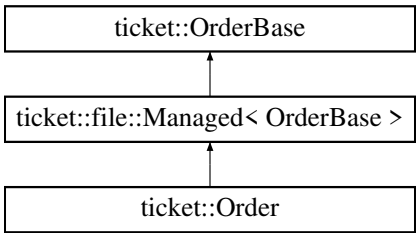
The documentation for this class was generated from the following file:

- lib/optional.h

6.39 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

6.39.1 Constructor & Destructor Documentation

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

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

6.39.2 Member Data Documentation

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

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

Initial value:

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

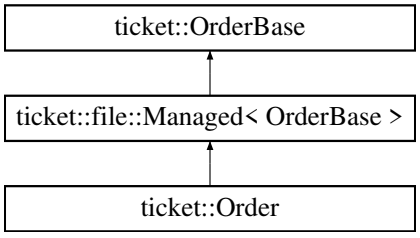
The documentation for this struct was generated from the following files:

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

6.40 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.
- auto satisfiable () -> bool

  checks if the pending order is satisfiable.

Public Attributes

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

Static Public Attributes

- static constexpr const char * filename = "orders"

6.40.1 Member Typedef Documentation

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

6.40.2 Member Enumeration Documentation

6.40.2.1 Status `enum ticket::OrderBase::Status`

Enumerator

kSuccess	
kPending	
kRefunded	

6.40.3 Member Function Documentation

6.40.3.1 getTrain() `auto ticket::OrderBase::getTrain ( ) -> Train`

gets the corresponding train object.

6.40.3.2 satisfiable() `auto ticket::OrderBase::satisfiable ( ) -> bool`

checks if the pending order is satisfiable.

6.40.4 Member Data Documentation

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

6.40.4.2 ixFrom `int ticket::OrderBase::ixFrom`

6.40.4.3 ixTo `int ticket::OrderBase::ixTo`

6.40.4.4 ride `Ride ticket::OrderBase::ride`

6.40.4.5 seats `int ticket::OrderBase::seats`

6.40.4.6 status `Status ticket::OrderBase::status`

6.40.4.7 user `User::Id ticket::OrderBase::user`

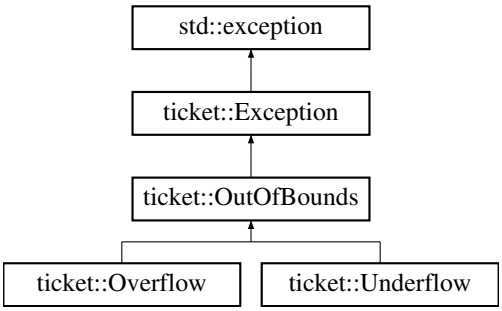
The documentation for this struct was generated from the following file:

- src/order.h

6.41 ticket::OutOfBounds Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::OutOfBounds:



Public Member Functions

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

6.41.1 Constructor & Destructor Documentation

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

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

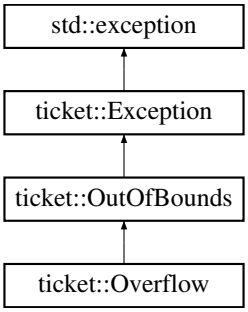
The documentation for this class was generated from the following file:

- lib/exception.h

6.42 ticket::Overflow Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::Overflow:



Public Member Functions

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

6.42.1 Constructor & Destructor Documentation

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

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

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

- lib/exception.h

6.43 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

6.43.1 Detailed Description

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

A pair of objects.

6.43.2 Constructor & Destructor Documentation

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.43.3 Member Data Documentation

6.43.3.1 first template<typename T1 , typename T2 >

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

6.43.3.2 second template<typename T1 , typename T2 >

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

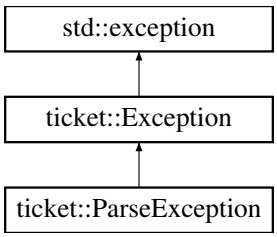
The documentation for this class was generated from the following file:

- lib/utility.h

6.44 ticket::ParseException Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::ParseException:



Public Member Functions

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

6.44.1 Constructor & Destructor Documentation

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

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

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

- lib/exception.h

6.45 ticket::command::QueryOrder Struct Reference

`#include <parser.h>`

Public Attributes

- std::string currentUser

6.45.1 Member Data Documentation

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

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

- src/parser.h

6.46 ticket::command::QueryProfile Struct Reference

`#include <parser.h>`

Public Attributes

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

6.46.1 Member Data Documentation

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

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

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

- src/parser.h

6.47 ticket::command::QueryTicket Struct Reference

`#include <parser.h>`

Public Attributes

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

6.47.1 Member Data Documentation

6.47.1.1 date `Date ticket::command::QueryTicket::date`

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

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

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

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

- src/parser.h

6.48 ticket::command::QueryTrain Struct Reference

`#include <parser.h>`

Public Attributes

- std::string id
- Date date

6.48.1 Member Data Documentation

6.48.1.1 date `Date ticket::command::QueryTrain::date`

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

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

- src/parser.h

6.49 ticket::command::QueryTransfer Struct Reference

`#include <parser.h>`

Public Attributes

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

6.49.1 Member Data Documentation

6.49.1.1 date

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

6.49.1.2 from

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

6.49.1.3 sort

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

6.49.1.4 to

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

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

- src/parser.h

6.50 ticket::command::RefundTicket Struct Reference

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

Public Attributes

- std::string currentUser
- int index = 1

6.50.1 Member Data Documentation

6.50.1.1 currentUser

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

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

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

- src/parser.h

6.51 ticket::rollback::RefundTicket Struct Reference

`#include <rollback.h>`

Public Attributes

- int id
- Order::Status status

6.51.1 Member Data Documentation

6.51.1.1 id `int ticket::rollback::RefundTicket::id`

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

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

- src/rollback.h

6.52 ticket::command::ReleaseTrain Struct Reference

`#include <parser.h>`

Public Attributes

- std::string id

6.52.1 Member Data Documentation

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

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

- src/parser.h

6.53 ticket::rollback::ReleaseTrain Struct Reference

`#include <rollback.h>`

Public Attributes

- int id

6.53.1 Member Data Documentation

6.53.1.1 id `int ticket::rollback::ReleaseTrain::id`

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

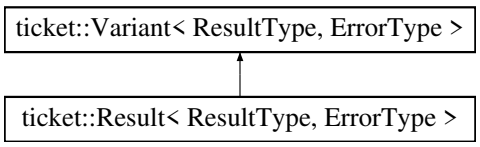
- src/rollback.h

6.54 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.

6.54.1 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.

6.54.2 Constructor & Destructor Documentation

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

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

6.54.2.2 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]
```

6.54.2.3 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]
```

6.54.3 Member Function Documentation

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

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

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

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

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

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

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

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

6.54.3.5 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

6.55 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

6.55.1 Member Function Documentation

6.55.1.1 operator<() auto ticket::Ride::operator< (const Ride & *rhs*) const -> bool

6.55.2 Member Data Documentation

6.55.2.1 date `Date ticket::Ride::date`

6.55.2.2 train `int ticket::Ride::train`

the numerical id of the train.

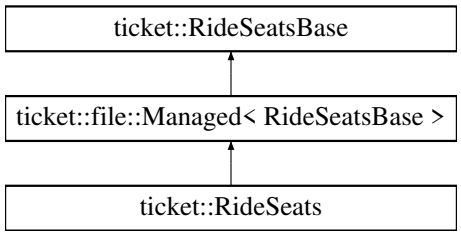
The documentation for this struct was generated from the following file:

- src/train.h

6.56 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

6.56.1 Constructor & Destructor Documentation

6.56.1.1 RideSeats() [1/2] ticket::RideSeats::RideSeats ( ) [default]

6.56.1.2 RideSeats() [2/2] ticket::RideSeats::RideSeats (const file::Managed< RideSeatsBase > & *rideSeats*) [inline]

6.56.2 Member Data Documentation

6.56.2.1 ixRide file::Index< Ride, RideSeats > ticket::RideSeats::ixRide {&RideSeats::ride, "ride-seats.ride.ix"} [static]

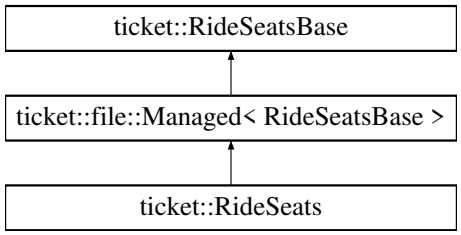
The documentation for this struct was generated from the following files:

- src/train.h
- src/train.cpp

6.57 ticket::RideSeatsBase Struct Reference

`#include <train.h>`

Inheritance diagram for ticket::RideSeatsBase:



Public Member Functions

- auto ticketsAvailable (int ixFrom, int ixTo) -> int
  calculates how many tickets are still available.

Public Attributes

- Ride ride
- file::Array< int, 99 > seatsRemaining

Static Public Attributes

- static constexpr const char * filename = "ride-seats"

6.57.1 Member Function Documentation

6.57.1.1 ticketsAvailable() auto ticket::RideSeatsBase::ticketsAvailable (int *ixFrom*, int *ixTo*) -> int

calculates how many tickets are still available.

Parameters

ixFrom	index of the departing stop
ixTo	index of the arriving stop

6.57.2 Member Data Documentation

6.57.2.1 filename `constexpr const char* ticket::RideSeatsBase::filename = "ride-seats" [static], [constexpr]`

6.57.2.2 ride `Ride ticket::RideSeatsBase::ride`

6.57.2.3 seatsRemaining `file::Array<int, 99> ticket::RideSeatsBase::seatsRemaining`

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

- src/train.h

6.58 ticket::command::Rollback Struct Reference

`#include <parser.h>`

Public Attributes

- int timestamp

6.58.1 Member Data Documentation

6.58.1.1 timestamp `int ticket::command::Rollback::timestamp`

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

- src/parser.h

6.59 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]

6.59.1 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.

6.59.2 Constructor & Destructor Documentation

6.59.2.1 Set() template<typename T , size_t maxLength, typename Cmp = Less<>> ticket::file::Set< T, maxLength, Cmp >::Set ( ) [default]

6.59.3 Member Function Documentation

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

clears the set.

6.59.3.2 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.

6.59.3.3 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.

6.59.3.4 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.

6.59.3.5 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.

6.59.3.6 indexOfInsert()

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

6.59.3.7 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.

6.59.3.8 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]
```

6.59.3.9 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]
```

6.59.3.10 pop()

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

pops the greatest element.

6.59.3.11 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.

6.59.3.12 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.

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

pops the least element.

6.59.4 Member Data Documentation

6.59.4.1 content template<typename T , size_t maxLength, typename Cmp = Less<>> T ticket::file::Set< T, maxLength, Cmp >::content[maxLength]

6.59.4.2 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

6.60 ticket::TrainBase::Stop Struct Reference

#include <train.h>

Public Attributes

- Station::Id name

6.60.1 Member Data Documentation

6.60.1.1 name Station::Id ticket::TrainBase::Stop::name

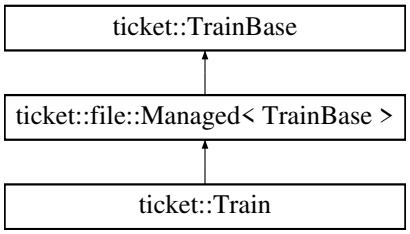
The documentation for this struct was generated from the following file:

- src/train.h

6.61 ticket::Train Struct Reference

`#include <train.h>`

Inheritance diagram for ticket::Train:



Public Member Functions

- Train ()=default
- Train (const file::Managed< TrainBase > &train)

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

6.61.1 Constructor & Destructor Documentation

6.61.1.1 Train() [1/2] ticket::Train::Train ( ) [default]

6.61.1.2 Train() [2/2] ticket::Train::Train (const file::Managed< TrainBase > & *train*) [inline]

6.61.2 Member Data Documentation

6.61.2.1 ixId file::Index< Train::Id, Train > ticket::Train::ixId {&Train::trainId, "trains.↩ train-id.ix"} [static]

6.61.2.2 ixStop `file::BpTree< size_t, int > ticket::Train::ixStop {"trains.stop.ix"} [static]`

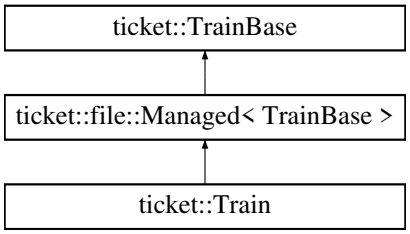
The documentation for this struct was generated from the following files:

- src/train.h
- src/train.cpp

6.62 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 Member Functions

- auto indexOfStop (const std::string &name) -> Result< int, NotFound >

  finds the index of the station of the given name.
- auto totalPrice (int ixDeparture, int ixArrival) -> int

  calculates the total price of a trip.
- auto getRide (Date date) -> RideSeats

  gets the remaining seats object on a given date.
- auto getRide (Date date, int ixDeparture) -> RideSeats

  gets the remaining seats object on a given date at a given stop.
- auto runsOnDate (Date date) -> bool

  checks if the train has a ride departing from the first station on the given date.
- auto runsOnDate (Date date, int ixDeparture) -> bool

  checks if the train has a ride departing from the given station on the given date.

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"

6.62.1 Member Typedef Documentation

6.62.1.1 Id `using ticket::TrainBase::Id = file::Varchar<20>`

6.62.1.2 Type `using ticket::TrainBase::Type = char`

6.62.2 Member Function Documentation

6.62.2.1 getRide() [1/2]

```
auto ticket::TrainBase::getRide (
        Date date ) -> 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).

6.62.2.2 getRide() [2/2]

```
auto ticket::TrainBase::getRide (
        Date date,
        int ixDeparture ) -> 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.

6.62.2.3 indexOfStop()

```
auto ticket::TrainBase::indexOfStop (
        const std::string & name ) -> Result< int, NotFound >
```

finds the index of the station of the given name.

6.62.2.4 runsOnDate() [1/2]

```
auto ticket::TrainBase::runsOnDate (
        Date date ) -> bool
```

checks if the train has a ride departing from the first station on the given date.

Parameters

date	the departure date of the first station.

6.62.2.5 runsOnDate() [2/2]

```
auto ticket::TrainBase::runsOnDate (
        Date date,
        int ixDeparture ) -> bool
```

checks if the train has a ride departing from the given station on the given date.

Parameters

date	the departure date of the given station.
ixDeparture	the index of the departing stop.

6.62.2.6 totalPrice()

```
auto ticket::TrainBase::totalPrice (
        int ixDeparture,
        int ixArrival ) -> int
```

calculates the total price of a trip.

6.62.3 Member Data Documentation

6.62.3.1 begin

```
Date ticket::TrainBase::begin
```

6.62.3.2 deleted

```
bool ticket::TrainBase::deleted = false
```

6.62.3.3 edges

```
file::Array<Edge, 99> ticket::TrainBase::edges
```

6.62.3.4 end

```
Date ticket::TrainBase::end
```

6.62.3.5 filename

```
constexpr const char* ticket::TrainBase::filename = "trains"  [static], [constexpr]
```

6.62.3.6 released

```
bool ticket::TrainBase::released = false
```

6.62.3.7 seats

```
int ticket::TrainBase::seats
```

6.62.3.8 stops

```
file::Array<Stop, 100> ticket::TrainBase::stops
```

6.62.3.9 trainId

```
Id ticket::TrainBase::trainId
```

6.62.3.10 type

```
Type ticket::TrainBase::type
```

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

- src/train.h

6.63 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

6.63.1 Detailed Description

template<typename T1, typename T2, typename T3>
class ticket::Triple< T1, T2, T3 >

A triplet of objects.

6.63.2 Constructor & Destructor Documentation

6.63.2.1 Triple() [1/4]

```
template<typename T1 , typename T2 , typename T3 >
constexpr ticket::Triple< T1, T2, T3 >::Triple ( )  [inline], [constexpr]
```

6.63.2.2 Triple() [2/4]

```
template<typename T1 , typename T2 , typename T3 >
ticket::Triple< T1, T2, T3 >::Triple (
            const Triple< T1, T2, T3 > & other )  [default]
```

6.63.2.3 Triple() [3/4]

```
template<typename T1 , typename T2 , typename T3 >
ticket::Triple< T1, T2, T3 >::Triple (
            Triple< T1, T2, T3 > && other )  [default], [noexcept]
```

6.63.2.4 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]
```

6.63.3 Member Data Documentation

6.63.3.1 first `template<typename T1 , typename T2 , typename T3 >`

`T1 ticket::Triple< T1, T2, T3 >::first`

6.63.3.2 second `template<typename T1 , typename T2 , typename T3 >`

`T2 ticket::Triple< T1, T2, T3 >::second`

6.63.3.3 third `template<typename T1 , typename T2 , typename T3 >`

`T3 ticket::Triple< T1, T2, T3 >::third`

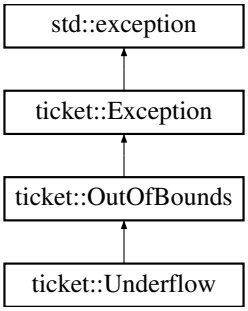
The documentation for this class was generated from the following file:

- lib/utility.h

6.64 ticket::Underflow Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::Underflow:



Public Member Functions

- Underflow ()
- Underflow (const char ∗what)

6.64.1 Constructor & Destructor Documentation

6.64.1.1 Underflow() [1/2] ticket::Underflow::Underflow ( ) [inline]

6.64.1.2 Underflow() [2/2] ticket::Underflow::Underflow (const char * what) [inline]

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

- lib/exception.h

6.65 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

6.65.1 Detailed Description

An empty class, used at various places.

6.65.2 Constructor & Destructor Documentation

6.65.2.1 Unit() [1/2] constexpr ticket::Unit::Unit ( ) [constexpr], [default]

6.65.2.2 Unit() [2/2] template<typename T > constexpr ticket::Unit::Unit (const T &) [inline], [constexpr]

6.65.3 Member Function Documentation

6.65.3.1 operator<() auto ticket::Unit::operator< (const Unit &) -> bool [inline]

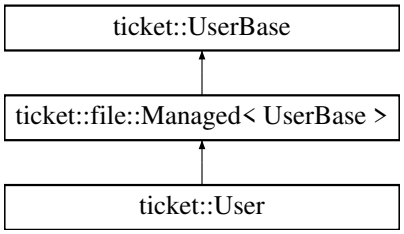
The documentation for this struct was generated from the following file:

- lib/utility.h

6.66 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

6.66.1 Constructor & Destructor Documentation

6.66.1.1 User() [1/2] ticket::User::User ( ) [default]

6.66.1.2 User() [2/2] ticket::User::User (const file::Managed< UserBase > & *user*) [inline]

6.66.2 Member Data Documentation

6.66.2.1 ixUsername `file::Index< User::Id, User > ticket::User::ixUsername {&User::username, "users.username.ix"} [static]`

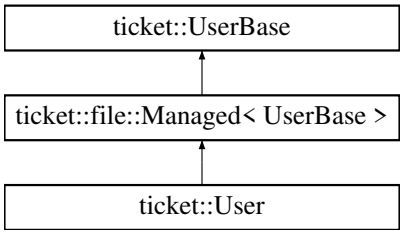
The documentation for this struct was generated from the following files:

- src/user.h
- src/user.cpp

6.67 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.

Public Attributes

- Id username
- Password password
- Name name
- Email email
- Privilege privilege

Static Public Attributes

- static constexpr const char * filename = "users"

6.67.1 Member Typedef Documentation

6.67.1.1 Email

```
using ticket::UserBase::Email = file::Varchar<30>
```

6.67.1.2 Id

```
using ticket::UserBase::Id = file::Varchar<20>
```

6.67.1.3 Name

```
using ticket::UserBase::Name = file::Varchar<15>
```

6.67.1.4 Password

```
using ticket::UserBase::Password = file::Varchar<30>
```

6.67.1.5 Privilege

```
using ticket::UserBase::Privilege = int
```

6.67.2 Member Function Documentation

6.67.2.1 has()

```
auto ticket::UserBase::has (
        const char * username ) -> bool  [static]
```

checks if there is a user with the given username.

6.67.3 Member Data Documentation

6.67.3.1 email

```
Email ticket::UserBase::email
```

6.67.3.2 filename

```
constexpr const char* ticket::UserBase::filename = "users"  [static], [constexpr]
```

6.67.3.3 name

`Name ticket::UserBase::name`

6.67.3.4 password

`Password ticket::UserBase::password`

6.67.3.5 privilege

`Privilege ticket::UserBase::privilege`

6.67.3.6 username

`Id ticket::UserBase::username`

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

- src/user.h
- src/user.cpp

6.68 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

6.68.1 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.

6.68.2 Constructor & Destructor Documentation

6.68.2.1 Varchar() [1/4] template<int maxLength>

```
ticket::file::Varchar< maxLength >::Varchar ( )  [inline]
```

6.68.2.2 Varchar() [2/4] template<int maxLength>

```
ticket::file::Varchar< maxLength >::Varchar (
            const std::string & s )  [inline]
```

6.68.2.3 Varchar() [3/4] template<int maxLength>

```
ticket::file::Varchar< maxLength >::Varchar (
            const char * cstr )  [inline]
```

6.68.2.4 Varchar() [4/4] template<int maxLength>

```
template<int A>
ticket::file::Varchar< maxLength >::Varchar (
            const Varchar< A > & that )  [inline]
```

6.68.3 Member Function Documentation

6.68.3.1 hash() template<int maxLength>

```
auto ticket::file::Varchar< maxLength >::hash ( ) const -> size_t   [inline]
```

6.68.3.2 length() template<int maxLength>

```
auto ticket::file::Varchar< maxLength >::length ( ) const -> int   [inline]
```

6.68.3.3 operator std::string() template<int maxLength>

```
ticket::file::Varchar< maxLength >::operator std::string ( ) const  [inline]
```

6.68.3.4 operator"!=() template<int maxLength>

```
template<int A>
auto ticket::file::Varchar< maxLength >::operator!= (
            const Varchar< A > & that ) const -> bool   [inline]
```

6.68.3.5 operator<() template<int maxLength>

```
template<int A>
auto ticket::file::Varchar< maxLength >::operator< (
            const Varchar< A > & that ) const -> bool   [inline]
```

6.68.3.6 operator=() template<int maxLength>

```
template<int A>
auto ticket::file::Varchar< maxLength >::operator= (
            const Varchar< A > & that ) -> Varchar &   [inline]
```

6.68.3.7 operator==() template<int maxLength>

```
template<int A>
auto ticket::file::Varchar< maxLength >::operator== (
            const Varchar< A > & that ) const -> bool   [inline]
```

6.68.3.8 str() template<int maxLength>

```
auto ticket::file::Varchar< maxLength >::str ( ) const -> std::string   [inline]
```

6.68.4 Friends And Related Function Documentation

6.68.4.1 Varchar template<int maxLength>

```
template<int A>
friend class Varchar  [friend]
```

6.68.5 Member Data Documentation

6.68.5.1 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

6.69 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.

6.69.1 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.

6.69.2 Constructor & Destructor Documentation

6.69.2.1 Variant() [1/4] template<typename ... Ts>

```
ticket::Variant< Ts >::Variant ( )  [inline]
```

6.69.2.2 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.

6.69.2.3 Variant() [3/4] template<typename ... Ts>

```
ticket::Variant< Ts >::Variant (
            const Variant< Ts > & other )  [inline]
```

6.69.2.4 Variant() [4/4] template<typename ... Ts>

```
ticket::Variant< Ts >::Variant (
            Variant< Ts > && other )  [inline], [noexcept]
```

6.69.2.5 ~Variant() template<typename ... Ts>

```
virtual ticket::Variant< Ts >::~Variant ( )  [inline], [virtual]
```

6.69.3 Member Function Documentation

6.69.3.1 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.

6.69.3.2 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.

6.69.3.3 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.

6.69.3.4 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.

6.69.3.5 index() template<typename ... Ts>

```
auto ticket::Variant< Ts >::index ( ) const -> int   [inline]
```

returns the current index of the current state.

6.69.3.6 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.

6.69.3.7 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.

6.69.3.8 operator=() [2/3] template<typename ... Ts>

```
auto ticket::Variant< Ts >::operator= (
            const Variant< Ts > & other ) -> Variant &   [inline]
```

6.69.3.9 operator=() [3/3] template<typename ... Ts>

```
auto ticket::Variant< Ts >::operator= (
            Variant< Ts > && other ) -> Variant &   [inline], [noexcept]
```

6.69.3.10 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

6.70 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

6.70.1 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.

6.70.2 Constructor & Destructor Documentation

6.70.2.1 Vector() [1/3] `template<typename T >`

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

6.70.2.2 Vector() [2/3] template<typename T >

```
ticket::Vector< T >::Vector (
            const Vector< T > & other )  [inline]
```

6.70.2.3 Vector() [3/3] template<typename T >

```
ticket::Vector< T >::Vector (
            Vector< T > && other )  [inline], [noexcept]
```

6.70.2.4 ~Vector() template<typename T >

```
ticket::Vector< T >::~Vector ( )  [inline]
```

6.70.3 Member Function Documentation

6.70.3.1 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)

6.70.3.2 at() [2/2] template<typename T >

```
auto ticket::Vector< T >::at (
            const size_t & pos ) const -> const T &   [inline]
```

6.70.3.3 back() template<typename T >

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

access the last element. throw container_is_empty if size == 0

6.70.3.4 begin() [1/2] template<typename T >

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

returns an iterator to the beginning.

6.70.3.5 begin() [2/2] template<typename T >

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

6.70.3.6 cbegin() template<typename T >

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

6.70.3.7 cend() template<typename T >

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

6.70.3.8 clear() template<typename T >

```
auto ticket::Vector< T >::clear ( ) -> void   [inline]
```

clears the contents

6.70.3.9 empty() template<typename T >

```
auto ticket::Vector< T >::empty ( ) const -> bool   [inline]
```

checks whether the container is empty

6.70.3.10 end() [1/2] template<typename T >

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

returns an iterator to the end.

6.70.3.11 end() [2/2] template<typename T >

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

6.70.3.12 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

6.70.3.13 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.

6.70.3.14 front() template<typename T >

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

access the first element. throw container_is_empty if size == 0

6.70.3.15 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.)

6.70.3.16 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.

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

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

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

```
auto ticket::Vector< T >::operator= (
            Vector< T > && other ) -> Vector &   [inline], [noexcept]
```

6.70.3.19 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.

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

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

6.70.3.21 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

6.70.3.22 push_back() template<typename T >

```
auto ticket::Vector< T >::push_back (
            const T & value ) -> void   [inline]
```

adds an element to the end.

6.70.3.23 reserve() template<typename T >

```
auto ticket::Vector< T >::reserve (
            size_t capacity ) -> void   [inline]
```

6.70.3.24 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 "utility.h"
```

Namespaces

- namespace ticket

Macros

- #define TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(name, cf)

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
7 #include "utility.h"
8
9 namespace ticket {
10
11 using std::distance, std::advance;
12
13 #define TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(name, cf) \
14 template<class Iterator, class T, class Compare = Less<>> \
15 auto name (Iterator first, Iterator last, const T &value, Compare cmp = {}) -> Iterator { \
16   int length = distance(first, last); \
17   while (length != 0) { \
18     auto it = first; \
19     int mid = length / 2; \
20     advance(it, mid); \
21     if (cmp.cf(value, *it)) { \
22       first = ++it; \
23       length -= mid + 1; \
24     } else { \
25       length = mid; \
26     } \
27   } \
28   return first; \
29 }
30 TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(upperBound, geq)
31 TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(lowerBound, gt)
32 #undef TICKET_ALGORIGHM_DEFINE_BOUND_FUNC
33
34 } // namespace ticket
35
36 #endif // TICKET_LIB_ALGORITHM_H_
```

7.3 lib/datetime.cpp File Reference

```
#include "datetime.h"
#include "utility.h"
```

Namespaces

- namespace ticket

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

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 } // namespace ticket
107
108 #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   virtual ~Exception () = 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) -> 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(); }
106
107 #ifdef TICKET_DEBUG
108    auto print () -> void { print_(Node::root(*this)); }
109 #endif
110
111   private:
112    File<Meta, szChunk> file_;
113    CmpKey cmpKey_;
114    CmpValue cmpValue_;
115
116    // data structures
118    struct Pair {
119      KeyType key;
120      ValueType value;
121      auto operator< (const Pair &that) const -> bool {
122        CmpKey cmpKey_;
123        CmpValue cmpValue_;
124        if (!cmpKey_.equals(key, that.key)) return cmpKey_.lt(key, that.key);
125        return cmpValue_.lt(value, that.value);
126      }
127    };
129    class KeyComparatorLess_ {
130     public:
131      auto operator() (const Pair &lhs, const KeyType &rhs) -> bool {
132        return cmpKey_.lt(lhs.key, rhs);
133      }
134      auto operator() (const KeyType &lhs, const Pair &rhs) -> bool {
135        return cmpKey_.geq(rhs.key, lhs);
136      }
137     private:
138      CmpKey cmpKey_;
139      CmpValue cmpValue_;
140    };
141
142    using NodeId = unsigned int;
143    // ROOT and INTERMEDIATE nodes are index nodes
144    enum NodeType { kRoot, kIntermediate, kRecord };
145    // if k > kLengthMax, there must be an overflow.
146    static constexpr size_t kLengthMax = 18446744073709000000ULL;
147    struct IndexPayload {
148      static constexpr size_t k = (szChunk - 2 * sizeof(NodeId)) / (sizeof(NodeId) + sizeof(Pair)) / 2 -
       1;
149      static_assert(k >= 2 && k < kLengthMax);
150      bool leaf = false;
152      Array<NodeId, 2 * k> children;
153      Set<Pair, 2 * k> splits;
154    };
155    struct RecordPayload {
156      static constexpr size_t l = (szChunk - 3 * sizeof(NodeId)) / sizeof(Pair) / 2 - 1;
157      static_assert(l >= 2 && l < kLengthMax);
158      NodeId prev = 0;
159      NodeId next = 0;
160      Set<Pair, 2 * l> entries;
161    };
162    union NodePayload {
163      IndexPayload index;
164      RecordPayload record;
165      NodePayload () {} // NOLINT
166    };
167    struct Node : public internal::UnmanagedObject<Node, Meta, szChunk> {
168      char _start[0];
169      NodeType type;
170      NodePayload payload;
171      char _end[0];
172      static_assert(sizeof(NodeType) + sizeof(NodePayload) <= szChunk);
173
174      // dynamically type-safe accessors
175      auto leaf () -> bool & { TICKET_ASSERT(type != kRecord); return payload.index.leaf; }
176      auto children () -> Array<NodeId, 2 * IndexPayload::k> & { TICKET_ASSERT(type != kRecord); return
       payload.index.children; }
177      auto splits () -> Set<Pair, 2 * IndexPayload::k> & { TICKET_ASSERT(type != kRecord); return
       payload.index.splits; }
178      auto prev () -> NodeId & { TICKET_ASSERT(type == kRecord); return payload.record.prev; }
```

```
179     auto next () -> NodeId & { TICKET_ASSERT(type == kRecord); return payload.record.next; }
180     auto entries () -> Set<Pair, 2 * RecordPayload::l> & { TICKET_ASSERT(type == kRecord); return
      payload.record.entries; }
181
182     Node (BpTree &tree, NodeType type) : internal::UnmanagedObject<Node, Meta, szChunk>(tree.file_),
      type(type) {
183       if (type == kRecord) {
184         new(&payload.record) RecordPayload;
185       } else {
186         new(&payload.index) IndexPayload;
187       }
188     }
189     ~Node () {
190       if (type == kRecord) {
191         payload.record.~RecordPayload();
192       } else {
193         payload.index.~IndexPayload();
194       }
195     }
196
197     static auto root (BpTree &tree) -> Node { return Node::get(tree.file_, 0); }
198
199     auto halfLimit () -> size_t {
200       return type == kRecord ? RecordPayload::l : IndexPayload::k;
201     }
202     auto length () -> size_t {
203       return type == kRecord ? payload.record.entries.length : payload.index.children.length;
204     }
205     auto shouldSplit () -> bool { return length() == 2 * halfLimit(); }
206     auto shouldMerge () -> bool { return length() < halfLimit(); }
207     auto lowerBound () -> Pair {
208       return type == kRecord ? payload.record.entries[0] : payload.index.splits[0];
209     }
210   };
211
212   // helper functions
213   auto ixInsert_ (const Pair &entry, Node &node) -> size_t {
214     TICKET_ASSERT(node.type != kRecord);
215     auto &splits = node.splits();
216     size_t ix = upperBound(splits.content, splits.content + splits.length, entry) - splits.content;
217     return ix == 0 ? ix : ix - 1;
218   }
219   auto splitRoot_ (Node &node) -> void {
220     Node left(*this, kIntermediate), right(*this, kIntermediate);
221
222     // copy children and splits
223     left.children().copyFrom(node.children(), 0, 0, IndexPayload::k);
224     left.splits().copyFrom(node.splits(), 0, 0, IndexPayload::k);
225     right.children().copyFrom(node.children(), IndexPayload::k, 0, IndexPayload::k);
226     right.splits().copyFrom(node.splits(), IndexPayload::k, 0, IndexPayload::k);
227     left.children().length = left.splits().length = right.children().length = right.splits().length =
      IndexPayload::k;
228
229     // set misc properties and save
230     left.leaf() = right.leaf() = node.leaf();
231     node.leaf() = false;
232     left.save();
233     right.save();
234
235     // initiate the new root node
236     node.children().clear();
237     node.children().insert(left.id(), 0);
238     node.children().insert(right.id(), 1);
239     node.splits().clear();
240     node.splits().insert(left.lowerBound());
241     node.splits().insert(right.lowerBound());
242   }
243   auto split_ (Node &node, Node &parent, size_t ixChild) -> void {
244     TICKET_ASSERT(node.shouldSplit());
245 #ifdef TICKET_DEBUG_BPTREE
246     std::cerr « "[Split] " « node.id() « " (parent " « parent.id() « ")" « std::endl;
247 #endif
248     if (node.type == kRoot) {
249       // the split of the root node is a bit different from other nodes. it produces two extra subnodes.
250       splitRoot_(node);
251       return;
252     }
253     TICKET_ASSERT(node.type != kRoot);
254
255     // create a new next node
256     Node next(*this, node.type);
257     if (node.type == kIntermediate) {
258       next.children().copyFrom(node.children(), IndexPayload::k, 0, IndexPayload::k);
259       next.splits().copyFrom(node.splits(), IndexPayload::k, 0, IndexPayload::k);
260       node.children().length = node.splits().length = next.children().length = next.splits().length =
      IndexPayload::k;
261       next.leaf() = node.leaf();
```

```
262       next.save();
263     } else {
264       TICKET_ASSERT(node.type == kRecord);
265       next.next() = node.next();
266       next.prev() = node.id();
267       memmove(
268         next.entries().content,
269         &node.entries().content[RecordPayload::l],
270         RecordPayload::l * sizeof(node.entries()[0])
271       );
272       next.entries().length = node.entries().length = RecordPayload::l;
273       next.save();
274       if (next.next() != 0) {
275         Node nextnext = Node::get(file_, next.next());
276         nextnext.prev() = next.id();
277         nextnext.update();
278       }
279       node.next() = next.id();
280     }
281
282     // update the parent node
283     parent.children().insert(next.id(), ixChild + 1);
284     parent.splits().insert(next.lowerBound());
285   }
286
287   template <typename A, typename B>
288   static auto unshift_ (A &to, B &from, size_t k) -> void {
289     // 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]]
290     to.copyFrom(to, 0, k, k - 1);
291     to.copyFrom(from, 0, 0, k);
292     to.length += from.length;
293     from.length = 0;
294   }
295   template <typename A, typename B>
296   static auto push_ (A &to, B &from, size_t k) -> void {
297     to.copyFrom(from, 0, k - 1, k);
298     to.length += from.length;
299     from.length = 0;
300   }
301   auto merge_ (Node &node, Node &parent, size_t ixChild) -> void {
302     TICKET_ASSERT(node.shouldMerge());
303 #ifdef TICKET_DEBUG_BPTREE
304     std::cerr « "[Merge] " « node.id() « " (parent " « parent.id() « ")" « std::endl;
305 #endif
306     if (node.type == kRoot) {
307       if (node.length() > 1 || node.leaf()) return;
308       Node onlyChild = Node::get(file_, node.children()[0]);
309       memcpy(&node, &onlyChild, sizeof(node));
310       node.type = kRoot;
311       return;
312     }
313     const bool hasPrev = ixChild != 0;
314     const bool hasNext = ixChild != parent.children().length - 1;
315     if (!hasNext) {
316       // don't do anything to the only data node.
317       if (!hasPrev && node.type == kRecord) return;
318       // all index nodes has at least 2 child nodes, except for the root node.
319       TICKET_ASSERT(hasPrev);
320       Node prev = Node::get(file_, parent.children()[ixChild - 1]);
321       if (prev.length() > prev.halfLimit()) {
322         if (node.type == kRecord) {
323           node.entries().insert(prev.entries().pop());
324         } else {
325           node.children().unshift(prev.children().pop());
326           node.splits().insert(prev.splits().pop());
327         }
328         prev.update();
329         parent.splits()[ixChild] = node.lowerBound();
330         return;
331       }
332       TICKET_ASSERT(prev.length() == prev.halfLimit());
333
334       if (node.type == kRecord) {
335         unshift_(node.entries(), prev.entries(), RecordPayload::l);
336         if (prev.prev() != 0) {
337           Node prevprev = Node::get(file_, prev.prev());
338           prevprev.next() = node.id();
339           prevprev.update();
340         }
341         node.prev() = prev.prev();
342       } else {
343         TICKET_ASSERT(node.type == kIntermediate);
344         unshift_(node.children(), prev.children(), IndexPayload::k);
345         unshift_(node.splits(), prev.splits(), IndexPayload::k);
346       }
347       parent.splits()[ixChild] = node.lowerBound();
348       parent.children().removeAt(ixChild - 1);
```

```
349       parent.splits().removeAt(ixChild - 1);
350       prev.destroy();
351       return;
352     }
353     TICKET_ASSERT(hasNext);
354
355     // FIXME: remove dupe code here
356     Node next = Node::get(file_, parent.children()[ixChild + 1]);
357     if (next.length() > next.halfLimit()) {
358       if (node.type == kRecord) {
359         node.entries().insert(next.entries().shift());
360       } else {
361         node.children().push(next.children().shift());
362         node.splits().insert(next.splits().shift());
363       }
364       next.update();
365       parent.splits()[ixChild + 1] = next.lowerBound();
366       return;
367     }
368     TICKET_ASSERT(next.length() == next.halfLimit());
369
370     if (node.type == kRecord) {
371       push_(node.entries(), next.entries(), RecordPayload::1);
372       if (next.next() != 0) {
373         Node nextnext = Node::get(file_, next.next());
374         nextnext.prev() = node.id();
375         nextnext.update();
376       }
377       node.next() = next.next();
378     } else {
379       TICKET_ASSERT(node.type == kIntermediate);
380       push_(node.children(), next.children(), IndexPayload::k);
381       push_(node.splits(), next.splits(), IndexPayload::k);
382     }
383
384     parent.children().removeAt(ixChild + 1);
385     parent.splits().removeAt(ixChild + 1);
386     next.destroy();
387   }
388
389   // FIXME: lengthy function name
390   auto addValuesToVectorForAllKeyFrom_ (Vector<ValueType> &vec, const KeyType &key, Node node, int
      first) -> void {
391     // we need to declare i outside to see if we have advanced to the last element
392     int i = first;
393     for (; i < node.length() && cmpKey_.equals(node.entries()[i].key, key); ++i)
      vec.push_back(node.entries()[i].value);
394     if (i == node.length() && node.next() != 0) addValuesToVectorForAllKeyFrom_(vec, key,
      Node::get(file_, node.next()), 0);
395   }
396   auto addEntriesToVector_ (Vector<ticket::Pair<KeyType, ValueType>> &vec, Node node) -> void {
397     for (int i = 0; i < node.length(); ++i) vec.emplace_back(node.entries()[i].key,
      node.entries()[i].value);
398     if (node.next() != 0) addEntriesToVector_(vec, Node::get(file_, node.next()));
399   }
400   auto findFirstChildWithKey_ (const KeyType &key, Node &node) -> ticket::Pair<Node, Optional<Node>> {
401     TICKET_ASSERT(node.type != kRecord);
402     size_t ixGreater = upperBound(
403       node.splits().content,
404       node.splits().content + node.length(),
405       key,
406       Less<KeyComparatorLess_>()
407     ) - node.splits().content;
408     bool hasCdr = ixGreater < node.length() && cmpKey_.equals(node.splits()[ixGreater].key, key);
409     auto cdr = hasCdr ? Optional<Node>(Node::get(file_, node.children()[ixGreater])) :
      Optional<Node>(unit);
410     size_t ix = ixGreater == 0 ? ixGreater : ixGreater - 1;
411     return { Node::get(file_, node.children()[ix]), cdr };
412   }
413
414   // operation functions
415   auto insert_ (const Pair &entry, Node &node) -> void {
416     if (node.type == kRecord) {
417       node.entries().insert(entry);
418       TICKET_ASSERT(node.entries().length <= 2 * RecordPayload::1);
419       return;
420     }
421     // if this is the first entry of the root, go create a record node.
422     if (node.children().length == 0) {
423       TICKET_ASSERT(node.type == kRoot);
424       TICKET_ASSERT(node.leaf());
425       Node child(*this, kRecord);
426       child.entries().insert(entry);
427       child.save();
428       node.children().push(child.id());
429       node.splits().insert(entry);
430       return;
```

```
431     }
432     size_t ix = ixInsert_(entry, node);
433     if (entry < node.splits()[ix]) node.splits()[ix] = entry;
434     Node nodeToInsert = Node::get(file_, node.children()[ix]);
435     insert_(entry, nodeToInsert);
436     node.splits()[ix] = nodeToInsert.lowerBound();
437     if (nodeToInsert.shouldSplit()) split_(nodeToInsert, node, ix);
438     nodeToInsert.update();
439   }
440   auto remove_ (const Pair &entry, Node &node) -> void {
441     if (node.type == kRecord) {
442       node.entries().remove(entry);
443       return;
444     }
445     size_t ix = ixInsert_(entry, node);
446     Node child = Node::get(file_, node.children()[ix]);
447     remove_(entry, child);
448     if (child.length() == 0) {
449       TICKET_ASSERT(node.type == kRoot);
450       TICKET_ASSERT(child.type == kRecord);
451       child.destroy();
452       node.children().clear();
453       node.splits().clear();
454       return;
455     }
456     node.splits()[ix] = child.lowerBound();
457     if (child.shouldMerge()) merge_(child, node, ix);
458     child.update();
459   }
460   auto findOne_ (const KeyType &key, Node node) -> Optional<ValueType> {
461     if (node.type != kRecord) {
462       if (node.length() == 0) return unit;
463       auto [ car, cdr ] = findFirstChildWithKey_(key, node);
464       if (!cdr) return findOne_(key, car);
465       auto res = findOne_(key, car);
466       if (res) return res;
467       return findOne_(key, *cdr);
468     }
469     size_t ix = upperBound(
470       node.entries().content,
471       node.entries().content + node.length(),
472       key,
473       Less<KeyComparatorLess_>()
474     ) - node.entries().content;
475     if (ix >= node.length()) return unit;
476     Pair entry = node.entries()[ix];
477     if (!cmpKey_.equals(entry.key, key)) return unit;
478     return entry.value;
479   }
480   auto includes_ (const Pair &entry, Node node) -> bool {
481     if (node.type == kRecord) return node.entries().includes(entry);
482     if (node.length() == 0) return false;
483     return includes_(entry, Node::get(file_, node.children()[ixInsert_(entry, node)]));
484   }
485   auto findMany_ (const KeyType &key, Node node) -> Vector<ValueType> {
486     if (node.type != kRecord) {
487       if (node.length() == 0) return {};
488       auto [ car, cdr ] = findFirstChildWithKey_(key, node);
489       if (!cdr) return findMany_(key, car);
490       Vector<ValueType> res = findMany_(key, car);
491       if (!res.empty()) return res;
492       return findMany_(key, *cdr);
493     }
494     size_t ix = upperBound(
495       node.entries().content,
496       node.entries().content + node.length(),
497       key,
498       Less<KeyComparatorLess_>()
499     ) - node.entries().content;
500     if (ix >= node.length()) return {};
501     Vector<ValueType> res;
502     addValuesToVectorForAllKeyFrom_(res, key, node, ix);
503     return res;
504   }
505   auto findAll_ (Node node) -> Vector<ticket::Pair<KeyType, ValueType>> {
506     if (node.type != kRecord) {
507       if (node.length() == 0) return {};
508       return findAll_(Node::get(file_, node.children()[0]));
509     }
510     Vector<ticket::Pair<KeyType, ValueType>> res;
511     addEntriesToVector_(res, node);
512     return res;
513   }
514   auto init_ () -> void {
515     Node root(*this, kRoot);
516     root.leaf() = true;
517     root.save();
```

```
518     TICKET_ASSERT(root.id() == 0);
519   }
520 #ifdef TICKET_DEBUG
521   auto print_ (Node node) -> void {
522     if (node.type == kRecord) {
523       std::cerr « "[Record " « node.id() « " (" « node.length() « "/" « 2 * RecordPayload::l - 1 « ")]";
524       for (int i = 0; i < node.length(); ++i) std::cerr « " (" « std::string(node.entries()[i].key) « ",
      " « node.entries()[i].value « ")";
525       std::cerr « std::endl;
526       return;
527     }
528     std::cerr « "[Node " « node.id() « " (" « node.length() « "/" « 2 * IndexPayload::k - 1 « ")" «
     (node.leaf() ? " leaf" : "") « "]";
529     for (int i = 0; i < node.length(); ++i) std::cerr « " (" « std::string(node.splits()[i].key) « ", "
     « node.splits()[i].value « ") " « node.children()[i];
530     std::cerr « std::endl;
531     for (int i = 0; i < node.length(); ++i) print_(Node::get(file_, node.children()[i]));
532   }
533 #endif
534 };
535
536 } // namespace ticket::file
537
538 #endif // TICKET_LIB_FILE_BPTREE_H_
```

7.12 lib/file/file.h File Reference

```
#include <cstring>
#include <fstream>
#include "hashmap.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 "utility.h"
10 #include "exception.h"
11
13 namespace ticket::file {
14
15 constexpr size_t kDefaultSzChunk = 4096;
16
24 template <typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
25 class File {
26  private:
27   class Metadata;
28  public:
38   template <typename Functor>
39   File (const char *filename, const Functor &initializer) {
40     init_(filename, initializer);
41   }
42   File (const char *filename) {
43     init_(filename, [] {});
44   }
45   ~File () { clearCache(); }
46
48   auto get (void *buf, size_t index, size_t n) -> void {
49     if (index != -1 && cache_.count(index) > 0) {
50       memcpy(buf, cache_[index], n);
51       return;
52     }
53     file_.seekg(offset_(index));
54     file_.read((char *) buf, n);
55     TICKET_ASSERT(file_.good());
56     if (index != -1) putCache_(buf, index, n);
57   }
59   auto set (const void *buf, size_t index, size_t n) -> void {
60     if (index != -1) {
61       // dirty check
62       if (cache_.count(index) > 0 && memcmp(buf, cache_[index], n) == 0) return;
63       putCache_(buf, index, n);
64     }
65     file_.seekp(offset_(index));
66     file_.write((const char *) buf, n);
67     TICKET_ASSERT(file_.good());
68   }
70   auto push (const void *buf, size_t n) -> size_t {
71     Metadata meta = meta_();
72     size_t id = meta.next;
73     if (meta.hasNext) {
74       Metadata nextMeta;
75       get(&nextMeta, meta.next, sizeof(nextMeta));
76       set(&nextMeta, -1, sizeof(nextMeta));
77     } else {
78       ++meta.next;
79       set(&meta, -1, sizeof(meta));
80     }
81     set(buf, id, n);
82     return id;
83   }
84   auto remove (size_t index) -> void {
85     Metadata meta = meta_();
86     set(&meta, index, sizeof(meta));
87     Metadata newMeta(index, true);
88     set(&newMeta, -1, sizeof(newMeta));
89     if (cache_.count(index) > 0) delete[] cache_[index];
90     cache_.erase(cache_.find(index));
91   }
92
94   auto getMeta () -> Meta {
95     return meta_().user;
96   }
98   auto setMeta (const Meta &user) -> void {
99     Metadata meta = meta_();
100     meta.user = user;
101     set(&meta, -1, sizeof(meta));
102   }
103
105   auto clearCache () -> void {
106     for (const auto &[ _, ptr ] : cache_) delete[] ptr;
```

```
107     cache_.clear();
108   }
109
110  private:
111   struct Metadata {
112     size_t next;
113     bool hasNext;
114     Meta user;
115     Metadata () = default;
116     Metadata (size_t next, bool hasNext) : next(next), hasNext(hasNext) {}
117   };
118   static_assert(szChunk > sizeof(Metadata));
119
120   template <typename Functor>
121   auto init_ (const char *filename, const Functor &initializer) -> void {
122     bool shouldCreate = false;
123     auto testFile = fopen(filename, "r");
124     if (testFile == nullptr) {
125       if (errno != ENOENT) {
126         throw IoException("Unable to open file");
127       }
128       shouldCreate = true;
129     } else if (fclose(testFile)) {
130       throw IoException("Unable to close file");
131     }
132     if (shouldCreate) {
133       auto file = fopen(filename, "w+");
134       if (file == nullptr) {
135         throw IoException("Unable to create file");
136       }
137       if (fclose(file)) {
138         throw IoException("Unable to close file when creating file");
139       }
140     }
141     file_.open(filename);
142     if (!file_.is_open() || !file_.good()) {
143       throw IoException("Unable to open file");
144     }
145     if (shouldCreate) {
146       Metadata meta(0, false);
147       set(&meta, -1, sizeof(meta));
148       initializer();
149     }
150   }
151
152   auto meta_ () -> Metadata {
153     Metadata retval;
154     get(&retval, -1, sizeof(retval));
155     return retval;
156   }
157   auto offset_ (size_t index) -> size_t {
158     return (index + 1) * szChunk;
159   }
160   std::fstream file_;
161   HashMap<size_t, char *> cache_;
162   auto putCache_ (const void *buf, size_t index, size_t n) -> void {
163     char *cache = new char[n];
164     memcpy(cache, buf, n);
165     if (cache_.count(index) > 0) delete[] cache_[index];
166     cache_[index] = cache;
167   }
168 };
169
178 template <typename T, typename Meta = Unit>
179 class Managed : public T {
180  public:
182   static File<Meta, sizeof(T)> file;
183
191   auto id () const -> size_t { return id_; }
192
194   static auto get (size_t id) -> Managed {
195     char buf[sizeof(Managed)];
196     auto managed = reinterpret_cast<Managed *>(buf);
197     auto unmanaged = static_cast<T *>(managed);
198     file.get(unmanaged, id, sizeof(T));
199     managed->id_ = id;
200     return *managed;
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   private:
68    Key Model::*ptr_;
69    BpTree<Key, int> tree_;
70  };
71
77  template <size_t maxLength, typename Model>
78  class Index<Varchar<maxLength>, Model> {
79   private:
80    using Key = Varchar<maxLength>;
81   public:
88    Index (Key Model::*ptr, const char *filename)
89      : ptr_(ptr), tree_(filename) {}
91    auto insert (const Model &model) -> void {
92      tree_.insert(model.*ptr_.hash(), model.id());
93    }
95    auto remove (const Model &model) -> void {
96      tree_.remove(model.*ptr_.hash(), model.id());
97    }
99    auto findOne (const Key &key) -> Optional<Model> {
100     auto id = tree_.findOne(key.hash());
101     if (!id) return unit;
102     return Model::get(*id);
103   }
105   auto findOneId (const Key &key) -> Optional<int> {
106     return tree_.findOne(key.hash());
107   }
109   auto findMany (const Key &key) -> Vector<Model> {
110     Vector<Model> res;
111     auto ids = tree_.findMany(key.hash());
112     if (ids.size() > 0) res.reserve(ids.size());
113     for (auto id : ids) {
114       res.push_back(Model::get(id));
115     }
116     return res;
117   }
119   auto findManyId (const Key &key) -> Vector<int> {
120     return tree_.findMany(key.hash());
121   }
123   auto empty () -> bool {
124     return tree_.empty();
125   }
126  private:
127   Key Model::*ptr_;
128   BpTree<size_t, int> tree_;
129 };
130
131 } // namespace ticket::file
132
133 #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) -> 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() ? false : true;
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/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

7.23 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_
```

7.24 lib/optional.h File Reference

```
#include "utility.h"
#include "variant.h"
```

Classes

- class ticket::Optional< T >

  A resemblence of std::optional.

Namespaces

- namespace ticket

7.25 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_
```

7.26 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

7.27 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_
```

7.28 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.

7.29 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

7.29.1 Macro Definition Documentation

7.29.1.1 TICKET_ASSERT `#define TICKET_ASSERT(`

x)

7.30 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 #include <iostream>
6
7 #include "vector.h"
8
9 #ifdef TICKET_DEBUG
10 #include <cassert>
11 #define TICKET_ASSERT(x) assert(x)
12 #else
13 #define TICKET_ASSERT(x)
14 #endif // TICKET_DEBUG
15
16 namespace ticket {
17
29 auto split (std::string &str, char sep)
30   -> Vector<std::string_view>;
31
33 auto copyStrings (const Vector<std::string_view> &vec)
34   -> Vector<std::string>;
35
37 struct Unit {
38   constexpr Unit () = default;
39   template <typename T>
40   constexpr Unit (const T & /* unused */) {}
41   auto operator< (const Unit & /* unused */) -> bool {
42     return false;
43   }
44 };
45 inline constexpr Unit unit;
46
48 template <typename T>
49 auto declval () -> T;
```

```
50
52 template <typename T>
53 auto move (T &val) -> T && {
54   return reinterpret_cast<T &&>(val);
55 }
56
58 template <typename T1, typename T2>
59 class Pair {
60  public:
61   T1 first;
62   T2 second;
63   constexpr Pair () : first(), second() {}
64   Pair (const Pair &other) = default;
65   Pair (Pair &&other) noexcept = default;
66   Pair (const T1 &x, const T2 &y) : first(x), second(y) {}
67   template <class U1, class U2>
68   Pair (U1 &&x, U2 &&y) : first(x), second(y) {}
69   template <class U1, class U2>
70   Pair (const Pair<U1, U2> &other) : first(other.first), second(other.second) {}
71   template <class U1, class U2>
72   Pair (Pair<U1, U2> &&other) : first(other.first), second(other.second) {}
73 };
75 template <typename T1, typename T2, typename T3>
76 class Triple {
77  public:
78   T1 first;
79   T2 second;
80   T3 third;
81   constexpr Triple () : first(), second(), third() {}
82   Triple (const Triple &other) = default;
83   Triple (Triple &&other) noexcept = default;
84   Triple (const T1 &x, const T2 &y, const T3 &z) : first(x), second(y), third(z) {}
85 };
86
88 template <typename Lt>
89 class Cmp {
90  public:
91   template <typename T, typename U>
92   auto equals (const T &lhs, const U &rhs) -> bool {
93     return !lt_(lhs, rhs) && !lt_(rhs, lhs);
94   }
95   template <typename T, typename U>
96   auto ne (const T &lhs, const U &rhs) -> bool {
97     return !equals(lhs, rhs);
98   }
99   template <typename T, typename U>
100   auto lt (const T &lhs, const U &rhs) -> bool {
101     return lt_(lhs, rhs);
102   }
103   template <typename T, typename U>
104   auto gt (const T &lhs, const U &rhs) -> bool {
105     return lt_(rhs, lhs);
106   }
107   template <typename T, typename U>
108   auto leq (const T &lhs, const U &rhs) -> bool {
109     return !gt(lhs, rhs);
110   }
111   template <typename T, typename U>
112   auto geq (const T &lhs, const U &rhs) -> bool {
113     return !lt(lhs, rhs);
114   }
115  private:
116   Lt lt_;
117 };
118
119 #include "internal/cmp.inc"
120
121 template <typename Lt = internal::LessOp>
122 using Less = Cmp<Lt>;
123 template <typename Lt = internal::LessOp>
124 using Greater = Cmp<internal::GreaterOp<Lt>>;
125
126 inline auto isVisibleChar (char ch) -> bool {
127   return ch >= '\x21' && ch <= '\x7E';
128 }
129
130 } // namespace ticket
131
132 #endif // TICKET_LIB_UTILITY_H_
```

7.31 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

7.32 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_
```

7.33 lib/vector.h File Reference

```
#include <climits>
#include <cstddef>
#include <iterator>
#include "exception.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

7.34 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
10 namespace ticket {
11
17 template<typename T>
18 class Vector {
19  public:
20   class const_iterator;
21   class iterator {
22    public:
23     using difference_type = std::ptrdiff_t;
24     using value_type = T;
25     using pointer = T *;
26     using reference = T &;
27     using iterator_category = std::output_iterator_tag;
28
29    private:
30     Vector *home_;
31     pointer ptr_;
32     iterator (Vector *home, pointer ptr) : home_(home), ptr_(ptr) {}
33    public:
34     auto operator+ (const int &n) const -> iterator {
35       return iterator(home_, ptr_ + n);
36     }
37     auto operator- (const int &n) const -> iterator {
38       return iterator(home_, ptr_ - n);
39     }
40     // return the distance between two iterators,
41     // if these two iterators point to different vectors, throw invaild_iterator.
42     auto operator- (const iterator &rhs) const -> int {
43       if (home_ != rhs.home_) throw Exception("invalid operation");
44       return ptr_ - rhs.ptr_;
45     }
46     auto operator+= (const int &n) -> iterator & {
47       ptr_ += n;
48       return *this;
49     }
50     auto operator-= (const int &n) -> iterator & { return (*this += -n); }
51     auto operator++ (int) const -> iterator { return operator+(1); }
52     auto operator++ () -> iterator & { return (*this += 1); }
```

```
53     auto operator-- (int) const -> iterator { return operator+(-1); }
54     auto operator-- () -> iterator & { return (*this -= 1); }
55     auto operator* () const -> T & { return *ptr_; }
59     auto operator== (const iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
60     auto operator== (const const_iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
64     auto operator!= (const iterator &rhs) const -> bool { return !(*this == rhs); }
65     auto operator!= (const const_iterator &rhs) const -> bool { return !(*this == rhs); }
66     auto operator< (const iterator &rhs) const -> bool {
67       return **this < *rhs;
68     }
69     auto operator< (const const_iterator &rhs) const -> bool {
70       return **this < *rhs;
71     }
72     friend class const_iterator;
73     friend class Vector;
74   };
75   class const_iterator {
76    public:
77     using difference_type = std::ptrdiff_t;
78     using value_type = T;
79     using pointer = T *;
80     using reference = T &;
81     using iterator_category = std::output_iterator_tag;
82
83    private:
84     const Vector *home_;
85     const T *ptr_;
86     const_iterator (const Vector *home, pointer ptr) : home_(home), ptr_(ptr) {}
87    public:
88     auto operator+ (const int &n) const -> const_iterator {
89       return const_iterator(home_, ptr_ + n);
90     }
91     auto operator- (const int &n) const -> const_iterator {
92       return const_iterator(home_, ptr_ - n);
93     }
94     auto operator- (const const_iterator &rhs) const -> int {
95       if (home_ != rhs.home_) throw Exception("invalid operation");
96       return ptr_ - rhs.ptr_;
97     }
98     auto operator+= (const int &n) -> const_iterator & {
99       ptr_ += n;
100       return *this;
101     }
102     auto operator-= (const int &n) -> const_iterator & { return (*this += -n); }
103     auto operator++ (int) const -> const_iterator { return operator+(1); }
104     auto operator++ () -> const_iterator & { return (*this += 1); }
105     auto operator-- (int) const -> const_iterator { return operator+(-1); }
106     auto operator-- () -> const_iterator & { return (*this -= 1); }
107     auto operator* () const -> const T & { return *ptr_; }
108     auto operator== (const iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
109     auto operator== (const const_iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
110     auto operator!= (const iterator &rhs) const -> bool { return !(*this == rhs); }
111     auto operator!= (const const_iterator &rhs) const -> bool { return !(*this == rhs); }
112     auto operator< (const iterator &rhs) const -> bool {
113       return **this < *rhs;
114     }
115     auto operator< (const const_iterator &rhs) const -> bool {
116       return **this < *rhs;
117     }
118     friend class iterator;
119     friend class Vector;
120   };
121   Vector () = default;
122   Vector (const Vector &other) { *this = other; }
123   Vector (Vector &&other) noexcept { *this = move(other); }
124   ~Vector () {
125     destroyContents_();
126     delete[] reinterpret_cast<char *>(storage_);
127   }
128   auto operator= (const Vector &other) -> Vector & {
129     if (this == &other) return *this;
130     clear();
131     grow_(other.capacity_);
132     size_ = other.size_;
133     copyContents_(storage_, other.storage_, size_);
134     return *this;
135   }
136   auto operator= (Vector &&other) noexcept -> Vector & {
137     if (this == &other) return *this;
138     clear();
139     storage_ = other.storage_;
140     size_ = other.size_;
141     capacity_ = other.capacity_;
142     other.size_ = other.capacity_ = 0;
143     other.storage_ = nullptr;
144     return *this;
145   }
```

```
146
151   auto at (const size_t &pos) -> T & {
152     checkPosition_(pos);
153     return storage_[pos];
154   }
155   auto at (const size_t &pos) const -> const T & {
156     return const_cast<Vector *>(this)->at(pos);
157   }
164   auto operator[] (const size_t &pos) -> T & { return at(pos); }
165   auto operator[] (const size_t &pos) const -> const T & { return at(pos); }
170   auto front () const -> const T & {
171     checkNonEmpty_();
172     return at(0);
173   }
178   auto back () const -> const T & {
179     checkNonEmpty_();
180     return at(size_ - 1);
181   }
185   auto begin () -> iterator {
186     return iterator(this, storage_);
187   }
188   auto begin () const -> const_iterator { return cbegin(); }
189   auto cbegin () const -> const_iterator {
190     return const_iterator(this, storage_);
191   }
195   auto end () -> iterator {
196     return iterator(this, storage_ + size_);
197   }
198   auto end () const -> const_iterator { return cend(); }
199   auto cend () const -> const_iterator {
200     return const_iterator(this, storage_ + size_);
201   }
205   auto empty () const -> bool {
206     return size_ == 0;
207   }
211   auto size () const -> size_t {
212     return size_;
213   }
217   auto clear () -> void {
218     destroyContents_();
219     delete[] reinterpret_cast<char *>(storage_);
220     storage_ = nullptr;
221     capacity_ = 0;
222     size_ = 0;
223   }
228   auto insert (iterator pos, const T &value) -> iterator { return insert(pos.ptr_ - storage_, value); }
235   auto insert (const size_t &ix, const T &value) -> iterator {
236     if (ix > size_) throw OutOfBounds();
237     if (size_ == capacity_) grow_();
238     for (size_t i = size_; i > ix; --i) {
239       storage_[i] = move_(storage_[i - 1]);
240     }
241     storage_[ix] = value;
242     ++size_;
243     return iterator(this, storage_ + ix);
244   }
250   auto erase (iterator pos) -> iterator { return erase(pos.ptr_ - storage_); }
256   auto erase (const size_t &ix) -> iterator {
257     checkPosition_(ix);
258     for (size_t i = ix; i + 1 < size_; ++i) {
259       storage_[i] = move_(storage_[i + 1]);
260     }
261     (storage_ + size_ - 1)->~T();
262     --size_;
263     return iterator(this, storage_ + ix);
264   }
268   auto push_back (const T &value) -> void {
269     if (size_ == capacity_) grow_();
270     new(storage_ + size_) T(value);
271     ++size_;
272   }
277   auto pop_back () -> void {
278     checkNonEmpty_();
279     (storage_ + size_ - 1)->~T();
280     --size_;
281   }
282
283   auto reserve (size_t capacity) -> void {
284     if (capacity_ < capacity) grow_(capacity);
285   }
286
287  private:
288   static constexpr size_t kSzDefault_ = 4;
289   static constexpr size_t kSzT_ = sizeof(T);
290   T *storage_ = nullptr;
291   size_t capacity_ = 0;
292   size_t size_ = 0;
```

```
293
294   static auto move_ (T &el) -> T && { return reinterpret_cast<T &&>(el); }
295   static auto copyContents_ (T *to, T *from, size_t n) -> void {
296     for (size_t i = 0; i < n; ++i) {
297       to[i] = from[i];
298     }
299   }
300   static auto moveContents_ (T *to, T *from, size_t n) -> void {
301     for (size_t i = 0; i < n; ++i) {
302       new(to + i) T(move_(from[i]));
303       from[i].~T();
304     }
305   }
306   static auto destroyContents_ (T *array, size_t n) -> void {
307     for (size_t i = 0; i < n; ++i) {
308       (array + i)->~T();
309     }
310   }
311   auto destroyContents_ () -> void { destroyContents_(storage_, size_); }
312   auto grow_ (size_t capNew) -> void {
313     T *storeNew = reinterpret_cast<T *>(new char[capNew * kSzT_]);
314     if (storage_ != nullptr) {
315       moveContents_(storeNew, storage_, size_);
316       delete[] reinterpret_cast<char *>(storage_);
317     }
318     storage_ = storeNew;
319     capacity_ = capNew;
320   }
321   auto grow_ () -> void {
322     grow_(storage_ == nullptr ? kSzDefault_ : 2 * capacity_);
323   }
324   auto checkPosition_ (size_t pos) const -> void {
325     // since this is size_t which is unsigned, we could not have pos < 0.
326     if (pos >= size_) throw OutOfBounds();
327   }
328   auto checkNonEmpty_ () const -> void {
329     if (size_ == 0) throw OutOfBounds();
330   }
331 };
332
333 } // namespace ticket
334
335 #endif // TICKET_LIB_VECTOR_H_
```

7.35 src/main.cpp File Reference

```
#include <iostream>
#include "parser.h"
#include "response.h"
#include "rollback.h"
#include "utility.h"
```

Functions

- auto main () -> int

7.35.1 Function Documentation

7.35.1.1 main() `auto main ( ) -> int`

7.36 src/misc.cpp File Reference

```
#include "parser.h"
```

Namespaces

- namespace ticket

7.37 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

7.37.1 Function Documentation

7.37.1.1 execute() template<typename T >

```
auto execute (
            Napi::Env env,
            const T & cmd ) -> Napi::Value
```

7.37.1.2 handler() auto handler (

```
            const Napi::CallbackInfo & info ) -> Napi::Value
```

7.37.1.3 init() auto init (

```
            Napi::Env env,
            Napi::Object exports ) -> Napi::Object
```

7.38 src/order.cpp File Reference

```
#include "order.h"
#include "parser.h"
#include "rollback.h"
```

Namespaces

- namespace ticket

7.39 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::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 >

7.40 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 OrderBase {
13   using Id = int;
14   enum Status { kSuccess, kPending, kRefunded };
15
16   User::Id user;
17   Ride ride;
18   int ixFrom, ixTo;
19   int seats;
20   Status status;
21
23   auto getTrain () -> Train;
25   auto satisfiable () -> bool;
26
27   static constexpr const char *filename = "orders";
28 };
```

```
29 struct Order : public file::Managed<OrderBase> {
30   Order () = default;
31   Order (const file::Managed<OrderBase> &order)
32     : file::Managed<OrderBase>(order) {}
33   static file::Index<User::Id, Order> ixUserId;
34   static file::Index<Ride, Order> pendingOrders;
35 };
36
43 struct BuyTicketEnqueued {};
50 struct BuyTicketSuccess {
51   int price;
52 };
53 using BuyTicketResponse = Variant<
54   BuyTicketSuccess,
55   BuyTicketEnqueued
56 >;
57
58 } // namespace ticket
59
60 #endif // TICKET_ORDER_H_
```

7.41 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 >

7.42 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, Parse↩ Exception >
- auto ticket::command::dispatch (const AddUser &cmd) -> Result< Response, Exception >

  Visitor for the commands.
- auto ticket::command::dispatch (const Login &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const Logout &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const QueryProfile &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const ModifyProfile &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const AddTrain &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const DeleteTrain &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const ReleaseTrain &cmd) -> Result< Response, Exception >

- auto ticket::command::dispatch (const QueryTrain &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const QueryTicket &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const QueryTransfer &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const BuyTicket &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const QueryOrder &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const RefundTicket &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const Rollback &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const Clean &cmd) -> Result< Response, Exception >
- auto ticket::command::dispatch (const Exit &cmd) -> Result< Response, Exception >

7.43 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 dispatch (const AddUser &cmd) -> Result<Response, Exception>;
164 auto dispatch (const Login &cmd) -> Result<Response, Exception>;
165 auto dispatch (const Logout &cmd) -> Result<Response, Exception>;
166 auto dispatch (const QueryProfile &cmd) -> Result<Response, Exception>;
167 auto dispatch (const ModifyProfile &cmd) -> Result<Response, Exception>;
```

```
168 auto dispatch (const AddTrain &cmd) -> Result<Response, Exception>;
169 auto dispatch (const DeleteTrain &cmd) -> Result<Response, Exception>;
170 auto dispatch (const ReleaseTrain &cmd) -> Result<Response, Exception>;
171 auto dispatch (const QueryTrain &cmd) -> Result<Response, Exception>;
172 auto dispatch (const QueryTicket &cmd) -> Result<Response, Exception>;
173 auto dispatch (const QueryTransfer &cmd) -> Result<Response, Exception>;
174 auto dispatch (const BuyTicket &cmd) -> Result<Response, Exception>;
175 auto dispatch (const QueryOrder &cmd) -> Result<Response, Exception>;
176 auto dispatch (const RefundTicket &cmd) -> Result<Response, Exception>;
177 auto dispatch (const Rollback &cmd) -> Result<Response, Exception>;
178 auto dispatch (const Clean &cmd) -> Result<Response, Exception>;
179 auto dispatch (const Exit &cmd) -> Result<Response, Exception>;
180
181 } // namespace ticket::command
182
183 #endif // TICKET_PARSER_H_
```

7.44 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 Order &order) -> void

7.45 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, 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 Order &order) -> void

7.46 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   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 Order &order) -> 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 Order &order) -> Napi::Object;
44
45 #endif // BUILD_NODEJS
46
47 } // namespace response
48
49 } // namespace ticket
50
51 #endif // TICKET_RESPONSE_H_
```

7.47 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.

7.48 src/rollback.h File Reference

```
#include "file/file.h"
#include "optional.h"
#include "order.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::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::dispatch (const AddUser &log) -> Result< Unit, Exception >

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

7.49 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 "train.h"
8 #include "user.h"
9 #include "variant.h"
10
11 namespace ticket {
13 auto setTimestamp (int timestamp) -> void;
14 } // namespace ticket
15
16
17 namespace ticket::rollback {
18
19 struct AddUser {
20   int id;
21 };
22
23 struct ModifyProfile {
24   int id;
25   Optional<User::Password> password;
26   Optional<User::Name> name;
27   Optional<User::Email> email;
28   Optional<User::Privilege> privilege;
29 };
30
31 struct AddTrain {
32   int id;
33 };
34
35 struct DeleteTrain {
36   int id;
37 };
38
39 struct ReleaseTrain {
40   int id;
41 };
42
43 struct BuyTicket {
44   int id;
45 };
46
47 struct RefundTicket {
48   int id;
49   Order::Status status;
50 };
51
52 struct LogEntryBase {
53   using Content = Variant<
54     AddUser,
55     ModifyProfile,
56     AddTrain,
57     DeleteTrain,
58     ReleaseTrain,
59     BuyTicket,
60     RefundTicket
61   >;
62
63   int timestamp;
64   Content content;
65
66   static constexpr const char *filename = "rollback-log";
67 };
68 using LogEntry = file::Managed<LogEntryBase>;
69
71 auto log (const LogEntry::Content &content) -> void;
72
79 auto dispatch (const AddUser &log) -> Result<Unit, Exception>;
80 auto dispatch (const ModifyProfile &log) -> Result<Unit, Exception>;
81 auto dispatch (const AddTrain &log) -> Result<Unit, Exception>;
82 auto dispatch (const DeleteTrain &log) -> Result<Unit, Exception>;
83 auto dispatch (const ReleaseTrain &log) -> Result<Unit, Exception>;
84 auto dispatch (const BuyTicket &log) -> Result<Unit, Exception>;
85 auto dispatch (const RefundTicket &log) -> Result<Unit, Exception>;
86
87 } // namespace ticket::rollback
88
89 #endif // TICKET_BACKLOG_H_
```

7.50 src/strings.h File Reference

Namespaces

- namespace ticket
- namespace ticket::strings

Variables

- constexpr const char * ticket::strings::kSuccess = "0\n"
- constexpr const char * ticket::strings::kFail = "-1\n"

7.51 strings.h

Go to the documentation of this file.

```
1 #ifndef TICKET_STRINGS_H_
2 #define TICKET_STRINGS_H_
3
4 namespace ticket::strings {
5
6 constexpr const char *kSuccess = "0\n";
7 constexpr const char *kFail = "-1\n";
8
9 } // namespace ticket::strings
10
11 #endif // TICKET_STRINGS_H_
```

7.52 src/train.cpp File Reference

```
#include "train.h"
#include "parser.h"
#include "rollback.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 "result.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 "result.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
43   auto indexOfStop (const std::string &name) -> Result<int, NotFound>;
45   auto totalPrice (int ixDeparture, int ixArrival) -> int;
46
52   auto getRide (Date date) -> RideSeats;
59   auto getRide (Date date, int ixDeparture) -> RideSeats;
60
66   auto runsOnDate (Date date) -> bool;
```

```
73   auto runsOnDate (Date date, int ixDeparture) -> bool;
74
75   static constexpr const char *filename = "trains";
76 };
77 struct Train : public file::Managed<TrainBase> {
78   Train () = default;
79   Train (const file::Managed<TrainBase> &train)
80     : file::Managed<TrainBase>(train) {}
81   static file::Index<Train::Id, Train> ixId;
82   static file::BpTree<size_t, int> ixStop;
83 };
84
85
86 struct Ride {
88   int train;
89   Date date;
90
91   auto operator< (const Ride &rhs) const -> bool;
92 };
93
94 struct RideSeatsBase {
95   Ride ride;
96   file::Array<int, 99> seatsRemaining;
97
103   auto ticketsAvailable (int ixFrom, int ixTo) -> int;
104
105   static constexpr const char *filename = "ride-seats";
106 };
107 struct RideSeats : public file::Managed<RideSeatsBase> {
108   RideSeats () = default;
109   RideSeats (const file::Managed<RideSeatsBase> &rideSeats)
110     : file::Managed<RideSeatsBase>(rideSeats) {}
111   static file::Index<Ride, RideSeats> ixRide;
112 };
113
114 } // namespace ticket
115
116 #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"
#include "strings.h"
```

Namespaces

- namespace ticket

Functions

- auto ticket::isValidUsername (const std::string &username) -> bool
- auto ticket::isValidPassword (const std::string &password) -> bool
- auto ticket::isValidName (const std::string &name) -> bool
- auto ticket::isValidEmail (const std::string &email) -> bool
- auto ticket::isValidPrivilege (int privilege) -> bool
- auto ticket::isValidAddUser (const command::AddUser &cmd) -> bool
- auto ticket::makeUser (const command::AddUser &cmd) -> User
- auto ticket::insufficientPrivileges (const User &op, const User &target) -> bool

Variables

- HashMap< std::string, Unit > ticket::usersLoggedIn

  a set of users that are logged in.

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;
25
26   static constexpr const char *filename = "users";
27 };
28 struct User : public file::Managed<UserBase> {
29   User () = default;
30   User (const file::Managed<UserBase> &user)
31     : file::Managed<UserBase>(user) {}
32   static file::Index<User::Id, User> ixUsername;
33 };
34
35 } // namespace ticket
36
37 #endif // TICKET_USER_H_
```

Index