faketicket

Generated by Doxygen 1.9.3

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

ticket	**9**
ticket::command **Classes and parsers for commands**	**13**
ticket::file **File utilities**	**17**
ticket::rollback	**18**
ticket::Station	**19**

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 rollback
- namespace Station

Classes

- 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
- class OutOfBounds
- class Overflow
- class Pair

  A pair of objects.
- class ParseException
- struct PendingOrder
- class Result

  Result<Res, Err> = Res | Err.
- struct Ride
- struct RideSeats
- struct Train
- class Underflow
- struct Unit

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

  A tagged union, aka sum type.
- class Vector

  A data container like std::vector.

Typedefs

- template< typename Lt = internal::LessOp>
  using Less = Cmp< Lt >
- template< typename Lt = internal::LessOp>
  using Greater = Cmp< internal::GreaterOp< Lt > >

Functions

- 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.

Variables

- file::File orders {"orders"}
- file::Index< User::Id, Order, decltype(orders)> ixOrdersUserId {&Order::user, "orders.user.ix", orders}
- file::File pendingOrders {"pending-orders"}
- file::Index< Ride, PendingOrder, decltype(pendingOrders)> ixPendingOrdersRide
- file::File logEntries {"rollback-log"}
- file::File trains {"trains"}
- file::Index< Train::Id, Train, decltype(trains)> ixTrainsId {&Train::trainId, "trains.train-id.ix", trains}
- file::BpTree< size_t, int > ixTrainsStop {"trains.stop.ix"}
- file::File rideSeats {"ride-seats"}
- file::Index< Ride, RideSeats, decltype(rideSeats)> ixRideSeatsRide
- file::File users {"users"}
- file::Index< User::Id, User, decltype(users)> ixUsersUsername {&User::username, "users.username.ix", users}
- 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 Greater `template<typename Lt = internal::LessOp>`

```
using ticket::Greater = typedef Cmp<internal::GreaterOp<Lt> >
```

5.1.2.2 Less `template<typename Lt = internal::LessOp>`

```
using ticket::Less = typedef Cmp<Lt>
```

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 move() `template<typename T >`

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

forcefully make an rvalue.

5.1.3.4 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 ixOrdersUserId `file::Index< User::Id, Order, decltype(orders)> ticket::ixOrdersUserId {&Order::user, "orders.user.ix", orders}`

5.1.4.2 ixPendingOrdersRide `file::Index< Ride, PendingOrder, decltype(pendingOrders)> ticket↩ ::ixPendingOrdersRide`

Initial value:

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

5.1.4.3 ixRideSeatsRide `file::Index< Ride, RideSeats, decltype(rideSeats)> ticket::ixRide↩ SeatsRide`

Initial value:

```
{
    &RideSeats::ride,
    "ride-seats.ride.ix",
    rideSeats
  }
```

5.1.4.4 ixTrainsId `file::Index< Train::Id, Train, decltype(trains)> ticket::ixTrainsId {&Train::trainId, "trains.train-id.ix", trains}`

5.1.4.5 ixTrainsStop `file::BpTree< size_t, int > ticket::ixTrainsStop {"trains.stop.ix"}`

5.1.4.6 ixUsersUsername `file::Index< User::Id, User, decltype(users)> ticket::ixUsersUsername {&User::username, "users.username.ix", users}`

5.1.4.7 logEntries `file::File ticket::logEntries {"rollback-log"}`

5.1.4.8 orders `file::File ticket::orders {"orders"}`

5.1.4.9 pendingOrders `file::File ticket::pendingOrders {"pending-orders"}`

5.1.4.10 rideSeats `file::File ticket::rideSeats {"ride-seats"}`

5.1.4.11 trains `file::File ticket::trains {"trains"}`

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

5.1.4.13 users `file::File ticket::users {"users"}`

5.1.4.14 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 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, 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 dispatch (const AddUser &cmd) -> void

  Visitor for the commands.
- auto dispatch (const Login &cmd) -> void
- auto dispatch (const Logout &cmd) -> void
- auto dispatch (const QueryProfile &cmd) -> void
- auto dispatch (const ModifyProfile &cmd) -> void
- auto dispatch (const AddTrain &cmd) -> void
- auto dispatch (const ReleaseTrain &cmd) -> void
- auto dispatch (const QueryTrain &cmd) -> void
- auto dispatch (const QueryTicket &cmd) -> void
- auto dispatch (const QueryTransfer &cmd) -> void
- auto dispatch (const BuyTicket &cmd) -> void
- auto dispatch (const QueryOrder &cmd) -> void
- auto dispatch (const RefundTicket &cmd) -> void
- auto dispatch (const Rollback &cmd) -> void
- auto dispatch (const Clean &cmd) -> void
- auto dispatch (const Exit &cmd) -> void

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, 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() [1/16]

```
auto ticket::command::dispatch (
            const AddTrain & cmd ) -> void
```

5.2.4.2 dispatch() [2/16]

```
auto ticket::command::dispatch (
            const AddUser & cmd ) -> void
```

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() [3/16]

```
auto ticket::command::dispatch (
            const BuyTicket & cmd ) -> void
```

5.2.4.4 dispatch() [4/16]

```
auto ticket::command::dispatch (
            const Clean & cmd ) -> void
```

5.2.4.5 dispatch() [5/16]

```
auto ticket::command::dispatch (
            const Exit & cmd ) -> void
```

5.2.4.6 dispatch() [6/16]

```
auto ticket::command::dispatch (
            const Login & cmd ) -> void
```

5.2.4.7 dispatch() [7/16]

```
auto ticket::command::dispatch (
            const Logout & cmd ) -> void
```

5.2.4.8 dispatch() [8/16] auto ticket::command::dispatch (const ModifyProfile & *cmd*) -> void

5.2.4.9 dispatch() [9/16] auto ticket::command::dispatch (const QueryOrder & *cmd*) -> void

5.2.4.10 dispatch() [10/16] auto ticket::command::dispatch (const QueryProfile & *cmd*) -> void

5.2.4.11 dispatch() [11/16] auto ticket::command::dispatch (const QueryTicket & *cmd*) -> void

5.2.4.12 dispatch() [12/16] auto ticket::command::dispatch (const QueryTrain & *cmd*) -> void

5.2.4.13 dispatch() [13/16] auto ticket::command::dispatch (const QueryTransfer & *cmd*) -> void

5.2.4.14 dispatch() [14/16] auto ticket::command::dispatch (const RefundTicket & *cmd*) -> void

5.2.4.15 dispatch() [15/16] auto ticket::command::dispatch (const ReleaseTrain & *cmd*) -> void

5.2.4.16 dispatch() [16/16] auto ticket::command::dispatch (const Rollback & *cmd*) -> void

5.2.4.17 parse() `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, DataFile >

  Specialization of Index on Varchar.
- class ManagedObject

  an opinionated utility base class 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::rollback Namespace Reference

Classes

- struct AddTrain
- struct AddUser
- struct BuyTicket
- struct LogEntry
- struct ModifyProfile
- struct RefundTicket
- struct ReleaseTrain

Functions

- auto dispatch (const AddUser &log) -> void

    Visitor for the log entries.
- auto dispatch (const ModifyProfile &log) -> void
- auto dispatch (const AddTrain &log) -> void
- auto dispatch (const ReleaseTrain &log) -> void
- auto dispatch (const BuyTicket &log) -> void
- auto dispatch (const RefundTicket &log) -> void

Variables

- file::File logEntries

5.4.1 Function Documentation

5.4.1.1 dispatch() `[1/6]`

```
auto ticket::rollback::dispatch (
        const AddTrain & log ) -> void
```

5.4.1.2 dispatch() `[2/6]`

```
auto ticket::rollback::dispatch (
        const AddUser & log ) -> void
```

Visitor for the log entries.

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

5.4.1.3 dispatch() `[3/6]`

```
auto ticket::rollback::dispatch (
        const BuyTicket & log ) -> void
```

5.4.1.4 dispatch() [4/6]

```
auto ticket::rollback::dispatch (
        const ModifyProfile & log ) -> void
```

5.4.1.5 dispatch() [5/6]

```
auto ticket::rollback::dispatch (
        const RefundTicket & log ) -> void
```

5.4.1.6 dispatch() [6/6]

```
auto ticket::rollback::dispatch (
        const ReleaseTrain & log ) -> void
```

5.4.2 Variable Documentation

5.4.2.1 logEntries

```
file::File ticket::rollback::logEntries  [extern]
```

5.5 ticket::Station Namespace Reference

Typedefs

- using Id = file::Varchar< 30 >

5.5.1 Typedef Documentation

5.5.1.1 Id

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

6 Class Documentation

6.1 ticket::command::AddTrain Struct Reference

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

Public Attributes

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

6.1.1 Member Data Documentation

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

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

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

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

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

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

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

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

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

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

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

- src/parser.h

6.2 ticket::rollback::AddTrain Struct Reference

```
#include <rollback.h>
```

Public Attributes

- int id

6.2.1 Member Data Documentation

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

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

- src/rollback.h

6.3 ticket::command::AddUser Struct Reference

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

Public Attributes

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

6.3.1 Member Data Documentation

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

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

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

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

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

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

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

- src/parser.h

6.4 ticket::rollback::AddUser Struct Reference

`#include <rollback.h>`

Public Attributes

- int id

6.4.1 Member Data Documentation

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

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

- src/rollback.h

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

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

```
#include <array.h>
```

Public Member Functions

- auto indexOf (const T &element) -> size_t

  finds the index of element in the array.
- auto includes (const T &element) -> bool

  checks if the elements is included in the array.
- auto insert (const T &element, size_t offset) -> void

  moves the elements after offset backwards, and inserts the element at the offset.
- auto remove (const T &element) -> void

  removes the element, and moves forward the elements after it.
- auto removeAt (size_t offset) -> void

  removes the element at offset, and moves forward the elements after it.
- auto clear () -> void

  clears the array.
- auto copyFrom (const Array &other, size_t ixFrom, size_t ixTo, size_t count) -> void

  copies a portion of another array to this.
- auto operator[ ] (size_t index) -> T &
- auto operator[ ] (size_t index) const -> const T &
- auto pop () -> T

  pops the last element.
- auto shift () -> T

  pops the first element.
- auto push (const T &object) -> void

  pushes after the last element.
- auto unshift (const T &object) -> void

  pushes before the first element.
- template<typename Functor >
  auto forEach (const Functor &callback) -> T

  calls the callback for each element in the array.

Public Attributes

- size_t length = 0
- T content [maxLength]

6.5.1 Detailed Description

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

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

The value type needs to be trivial.

6.5.2 Member Function Documentation

6.5.2.1 clear()

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

clears the array.

6.5.2.2 copyFrom()

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

copies a portion of another array to this.

6.5.2.3 forEach()

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

calls the callback for each element in the array.

6.5.2.4 includes()

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

checks if the elements is included in the array.

6.5.2.5 indexOf()

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

finds the index of element in the array.

6.5.2.6 insert()

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

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

6.5.2.7 operator[]() [1/2]

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

6.5.2.8 operator[]() [2/2]

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

6.5.2.9 pop()

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

pops the last element.

6.5.2.10 push()

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

pushes after the last element.

6.5.2.11 remove()

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

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

6.5.2.12 removeAt() `template<typename T , size_t maxLength, typename Cmp = Less<>>`

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

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

6.5.2.13 shift() `template<typename T , size_t maxLength, typename Cmp = Less<>>`

```
auto ticket::file::Array< T, maxLength, Cmp >::shift ( ) -> T   [inline]
```

pops the first element.

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

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

pushes before the first element.

6.5.3 Member Data Documentation

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

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

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

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

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

- lib/file/array.h

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

an implementation of the B+ tree.

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

Public Member Functions

- BpTree (const char ∗filename)

  constructs a B+ tree on the given file.
- auto insert (const KeyType &key, const ValueType &value) -> void

  inserts a key-value pair into the tree.
- auto remove (const KeyType &key, const ValueType &value) -> void

  removes a key-value pair from the tree.
- auto findOne (const KeyType &key) -> Optional< ValueType >

  finds the first entry with the given key.
- auto findMany (const KeyType &key) -> Vector< ValueType >

  finds all entries with the given key.
- auto findAll () -> Vector< ticket::Pair< KeyType, ValueType > >

  finds all entries.
- auto includes (const KeyType &key, const ValueType &value) -> bool

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

6.6.3.3 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.

6.6.3.4 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.

6.6.3.5 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.

6.6.3.6 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.

6.6.3.7 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.

6.6.3.8 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.

6.6.3.9 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::command::Clean Struct Reference

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

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

- src/parser.h

6.10 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

6.10.1 Detailed Description

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

Comparison utilities.

6.10.2 Member Function Documentation

6.10.2.1 equals() `template<typename Lt >`

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

6.10.2.2 geq() `template<typename Lt >`

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

6.10.2.3 gt() `template<typename Lt >`

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

6.10.2.4 leq() `template<typename Lt >`

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

6.10.2.5 lt() `template<typename Lt >`

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

6.10.2.6 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

6.11 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

6.11.1 Member Typedef Documentation

6.11.1.1 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
```

6.11.1.2 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
```

6.11.1.3 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 *
```

6.11.1.4 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 &
```

6.11.1.5 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
```

6.11.2 Constructor & Destructor Documentation

6.11.2.1 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]
```

6.11.2.2 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]
```

6.11.2.3 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]
```

6.11.3 Member Function Documentation

6.11.3.1 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]
```

6.11.3.2 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]
```

6.11.3.3 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]
```

6.11.3.4 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]
```

6.11.3.5 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]
```

6.11.3.6 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]
```

6.11.3.7 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]
```

6.11.3.8 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]
```

6.11.3.9 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]
```

6.11.3.10 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]
```

6.11.4 Friends And Related Function Documentation

6.11.4.1 HashMap

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

6.11.4.2 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

6.12 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

6.12.1 Member Typedef Documentation

6.12.1.1 difference_type `template<typename T >`

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

6.12.1.2 iterator_category `template<typename T >`

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

6.12.1.3 pointer `template<typename T >`

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

6.12.1.4 reference `template<typename T >`

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

6.12.1.5 value_type `template<typename T >`

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

6.12.2 Member Function Documentation

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

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

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

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

6.12.2.3 operator*() `template<typename T >`

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

6.12.2.4 operator+() `template<typename T >`

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

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

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

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

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

6.12.2.7 operator+=() `template<typename T >`

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

6.12.2.8 operator-() [1/2]

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

6.12.2.9 operator-() [2/2]

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

6.12.2.10 operator--() [1/2]

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

6.12.2.11 operator--() [2/2]

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

6.12.2.12 operator-=()

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

6.12.2.13 operator<() [1/2]

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

6.12.2.14 operator<() [2/2]

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

6.12.2.15 operator==() [1/2]

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

6.12.2.16 operator==() [2/2]

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

6.12.3 Friends And Related Function Documentation

6.12.3.1 iterator

```
template<typename T >
friend class iterator [friend]
```

6.12.3.2 Vector

```
template<typename T >
friend class Vector [friend]
```

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

- lib/vector.h

6.13 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).

6.13.1 Detailed Description

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

6.13.2 Constructor & Destructor Documentation

6.13.2.1 Date() [1/3] `ticket::Date::Date ( ) [default]`

6.13.2.2 Date() [2/3]

```
ticket::Date::Date (
        int month,
        int date )
```

6.13.2.3 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.

6.13.3 Member Function Documentation

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

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

6.13.3.2 inRange()

```
auto ticket::Date::inRange (
        Date begin,
        Date end ) const -> bool
```

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

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

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

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

gets a MM-DD representation of the Date.

6.13.3.5 operator+()

```
auto ticket::Date::operator+ (
        int dt ) const -> Date
```

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

6.13.3.6 operator-() [1/2]

```
auto ticket::Date::operator- (
        Date rhs ) const -> int
```

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

6.13.3.7 operator-() [2/2]

```
auto ticket::Date::operator- (
        int dt ) const -> Date
```

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

6.13.3.8 operator<()

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

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

- lib/datetime.h

6.14 ticket::Duration Class Reference

Class representing a length of timespan.

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

Public Member Functions

- Duration ()=default
- Duration (int hour, int minute)
- Duration (int minutes)
- Duration (const char ∗str)

  constructs a Duration from an HH:MM format string.
- auto hours () const -> int

  gets the hour part of the duration, may be negative.
- auto minutes () const -> int

  gets the minute part of the duration, may be negative.
- auto totalMinutes () 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

6.14.1 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.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Duration() [1/4]

```
ticket::Duration::Duration ( )  [default]
```

6.14.2.2 Duration() [2/4]

```
ticket::Duration::Duration (
            int hour,
            int minute )
```

6.14.2.3 Duration() [3/4]

```
ticket::Duration::Duration (
            int minutes )  [explicit]
```

6.14.2.4 Duration() `[4/4]` `ticket::Duration::Duration ( const char * str ) [explicit]`

constructs a Duration from an HH:MM format string.

6.14.3 Member Function Documentation

6.14.3.1 hours() `auto ticket::Duration::hours ( ) const -> int`

gets the hour part of the duration, may be negative.

6.14.3.2 minutes() `auto ticket::Duration::minutes ( ) const -> int`

gets the minute part of the duration, may be negative.

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

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

negates the Duration.

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

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

6.14.3.7 totalMinutes() `auto ticket::Duration::totalMinutes ( ) const -> int`

gets how many minutes are there in this Duration.

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

- lib/datetime.h

6.15 ticket::Train::Edge Struct Reference

```
#include <train.h>
```

Public Attributes

- int price
- Instant departure
- Instant arrival

6.15.1 Member Data Documentation

6.15.1.1 arrival `Instant ticket::Train::Edge::arrival`

6.15.1.2 departure `Instant ticket::Train::Edge::departure`

6.15.1.3 price `int ticket::Train::Edge::price`

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

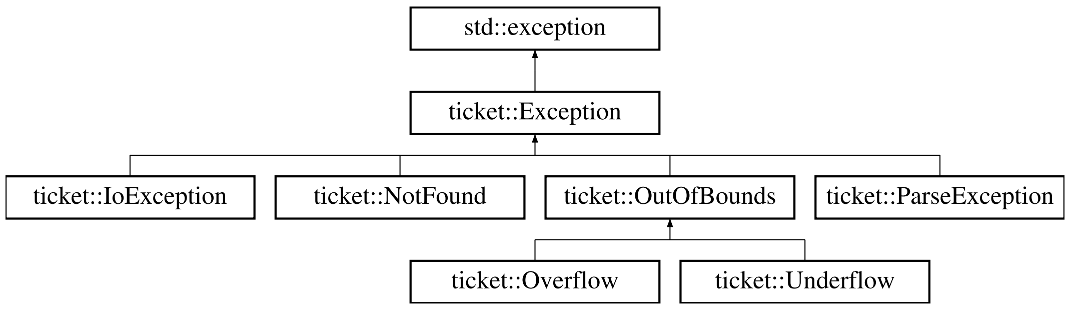
- src/train.h

6.16 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.

6.16.1 Detailed Description

The base exception class.

6.16.2 Constructor & Destructor Documentation

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

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

6.16.2.3 ∼Exception() `virtual ticket::Exception::~Exception ( ) [virtual], [default]`

6.16.3 Member Function Documentation

6.16.3.1 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

6.17 ticket::command::Exit Struct Reference

`#include <parser.h>`

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

- src/parser.h

6.18 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.

6.18.1 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.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 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.

6.18.2.2 File() [2/2]

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

6.18.2.3 ~File()

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

6.18.3 Member Function Documentation

6.18.3.1 clearCache()

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

clears the cache.

6.18.3.2 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.

6.18.3.3 getMeta()

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

gets user-provided metadata.

6.18.3.4 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

6.18.3.5 remove()

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

6.18.3.6 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.

6.18.3.7 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

6.19 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 find (const Key &key) -> iterator
- auto find (const Key &key) const -> const_iterator

6.19.1 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.

6.19.2 Member Typedef Documentation

6.19.2.1 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>
```

6.19.3 Constructor & Destructor Documentation

6.19.3.1 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]
```

6.19.3.2 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]
```

6.19.3.3 ∼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]
```

6.19.4 Member Function Documentation

6.19.4.1 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'

6.19.4.2 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]
```

6.19.4.3 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

6.19.4.4 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]
```

6.19.4.5 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]
```

6.19.4.6 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

6.19.4.7 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.

6.19.4.8 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

6.19.4.9 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

6.19.4.10 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)

6.19.4.11 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.

6.19.4.12 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]
```

6.19.4.13 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.

6.19.4.14 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]
```

6.19.4.15 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.

6.19.4.16 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.

6.19.4.17 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

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

Class representing an index file.

`#include <index.h>`

Public Member Functions

- Index (Key Model::*ptr, const char *filename, DataFile &datafile)

  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.

6.20.1 Detailed Description

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

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.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 Index() `template<typename Key , typename Model , typename DataFile >`

```
ticket::file::Index< Key, Model, DataFile >::Index (
            Key Model::* ptr,
            const char * filename,
            DataFile & datafile )   [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.

6.20.3 Member Function Documentation

6.20.3.1 findMany()

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

finds all Models of the given key in the index.

6.20.3.2 findManyId()

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

finds all IDs of the given keys in the index.

6.20.3.3 findOne()

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

finds one Model in the index.

6.20.3.4 findOneId()

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

finds one identifier in the index.

6.20.3.5 insert()

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

inserts an object into the index.

6.20.3.6 remove() `template<typename Key , typename Model , typename DataFile >`

```
auto ticket::file::Index< Key, Model, DataFile >::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

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

Specialization of Index on Varchar.

`#include <index.h>`

Public Member Functions

- Index (Key Model::∗ptr, const char ∗filename, DataFile &datafile)
  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.

6.21.1 Detailed Description

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

Specialization of Index on Varchar.

It makes use of hashes to speed up the process.

6.21.2 Constructor & Destructor Documentation

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

```
ticket::file::Index< Varchar< maxLength >, Model, DataFile >::Index (
             Key Model::* ptr,
             const char * filename,
             DataFile & datafile )   [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.

6.21.3 Member Function Documentation

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

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

finds all Models of the given key in the index.

6.21.3.2 findManyId() `template<size_t maxLength, typename Model , typename DataFile >`

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

finds all IDs of the given keys in the index.

6.21.3.3 findOne() `template<size_t maxLength, typename Model , typename DataFile >`

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

finds one Model in the index.

6.21.3.4 findOneId() `template<size_t maxLength, typename Model , typename DataFile >`

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

finds one identifier in the index.

6.21.3.5 insert() `template<size_t maxLength, typename Model , typename DataFile >`

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

inserts an object into the index.

6.21.3.6 remove() `template<size_t maxLength, typename Model , typename DataFile >`

```
auto ticket::file::Index< Varchar< maxLength >, Model, DataFile >::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

6.22 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

6.22.1 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.

6.22.2 Constructor & Destructor Documentation

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

6.22.2.2 Instant() [2/3]

```
ticket::Instant::Instant (
    int hour,
    int minute )
```

6.22.2.3 Instant() [3/3]

```
ticket::Instant::Instant (
    const char * str ) [explicit]
```

constructs an Instant from an HH:MM format string.

6.22.3 Member Function Documentation

6.22.3.1 daysOverflow()

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

6.22.3.2 hour()

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

6.22.3.3 minute()

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

6.22.3.4 operator std::string()

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

gets an HH:MM representation of the Instant.

6.22.3.5 operator+()

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

6.22.3.6 operator-() [1/2]

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

6.22.3.7 operator-() [2/2]

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

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

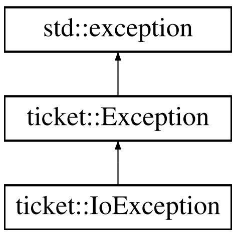
The documentation for this class was generated from the following file:

- lib/datetime.h

6.23 ticket::IoException Class Reference

```
#include <exception.h>
```

Inheritance diagram for ticket::IoException:



Public Member Functions

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

6.23.1 Constructor & Destructor Documentation

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

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

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

- lib/exception.h

6.24 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

6.24.1 Member Typedef Documentation

6.24.1.1 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
```

6.24.1.2 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
```

6.24.1.3 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 ∗
```

6.24.1.4 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 &
```

6.24.1.5 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
```

6.24.2 Constructor & Destructor Documentation

6.24.2.1 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]
```

6.24.2.2 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]
```

6.24.3 Member Function Documentation

6.24.3.1 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]
```

6.24.3.2 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]
```

6.24.3.3 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]
```

6.24.3.4 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]
```

6.24.3.5 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]
```

6.24.3.6 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]
```

6.24.3.7 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]
```

6.24.3.8 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]
```

6.24.3.9 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]
```

6.24.3.10 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]
```

6.24.4 Friends And Related Function Documentation

6.24.4.1 const_iterator `template<typename Key , typename Value , typename Hash = std::hash<↩ Key>, typename Equal = std::equal_to<Key>>`

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

6.24.4.2 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

6.25 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

6.25.1 Member Typedef Documentation

6.25.1.1 difference_type `template<typename T >`

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

6.25.1.2 iterator_category `template<typename T >`

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

6.25.1.3 pointer `template<typename T >`

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

6.25.1.4 reference `template<typename T >`

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

6.25.1.5 value_type `template<typename T >`

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

6.25.2 Member Function Documentation

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

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

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

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

some other operator for iterator.

6.25.2.3 operator∗() `template<typename T >`

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

6.25.2.4 operator+() `template<typename T >`

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

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

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

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

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

6.25.2.7 operator+=() `template<typename T >`

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

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

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

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

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

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

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

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

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

6.25.2.12 operator-=() `template<typename T >`

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

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

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

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

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

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

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

6.25.2.16 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).

6.25.3 Friends And Related Function Documentation

6.25.3.1 const_iterator `template<typename T >`

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

6.25.3.2 Vector `template<typename T >`

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

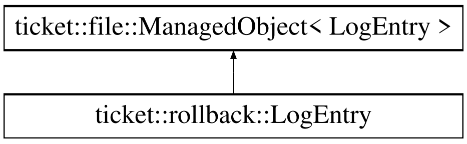
The documentation for this class was generated from the following file:

- lib/vector.h

6.26 ticket::rollback::LogEntry Struct Reference

```
#include <rollback.h>
```

Inheritance diagram for ticket::rollback::LogEntry:


ticket::file::ManagedObject< LogEntry >
ticket::rollback::LogEntry

Public Attributes

- int timestamp
- Variant< AddUser, ModifyProfile, AddTrain, ReleaseTrain, BuyTicket, RefundTicket > content

Additional Inherited Members

6.26.1 Member Data Documentation

6.26.1.1 content `Variant< AddUser, ModifyProfile, AddTrain, ReleaseTrain, BuyTicket, RefundTicket > ticket::rollback::LogEntry::content`

6.26.1.2 timestamp `int ticket::rollback::LogEntry::timestamp`

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

- src/rollback.h

6.27 ticket::command::Login Struct Reference

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

Public Attributes

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

6.27.1 Member Data Documentation

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

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

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

- src/parser.h

6.28 ticket::command::Logout Struct Reference

`#include <parser.h>`

Public Attributes

- std::string username

6.28.1 Member Data Documentation

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

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

- src/parser.h

6.29 ticket::file::ManagedObject< T, Meta, szChunk > Class Template Reference

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

`#include <file.h>`

Public Member Functions

- ManagedObject (File_ &file)
- virtual ~ManagedObject ()=default
- auto id () -> 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 (File_ &file, size_t id) -> T

  gets the object at id in file.

6.29.1 Detailed Description

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

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

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

6.29.2 Constructor & Destructor Documentation

6.29.2.1 ManagedObject()

```
template<typename T , typename Meta = Unit, size_t szChunk = k↩
DefaultSzChunk>
ticket::file::ManagedObject< T, Meta, szChunk >::ManagedObject (
            File_ & file )  [inline]
```

6.29.2.2 ~ManagedObject()

```
template<typename T , typename Meta = Unit, size_t szChunk = k↩
DefaultSzChunk>
virtual ticket::file::ManagedObject< T, Meta, szChunk >::~ManagedObject ( )  [virtual], [default]
```

6.29.3 Member Function Documentation

6.29.3.1 destroy()

```
template<typename T , typename Meta = Unit, size_t szChunk = kDefaultSz↩
Chunk>
auto ticket::file::ManagedObject< T, Meta, szChunk >::destroy ( ) -> void   [inline]
```

removes the object from the file.

6.29.3.2 get()

```
template<typename T , typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
static auto ticket::file::ManagedObject< T, Meta, szChunk >::get (
            File_ & file,
            size_t id ) -> T   [inline], [static]
```

gets the object at id in file.

6.29.3.3 id()

```
template<typename T , typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::ManagedObject< T, Meta, szChunk >::id ( ) -> size_t   [inline]
```

the unique immutable numeral identifier of the object.

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

6.29.3.4 save()

```
template<typename T , typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
auto ticket::file::ManagedObject< T, Meta, szChunk >::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().

6.29.3.5 update()

```
template<typename T , typename Meta = Unit, size_t szChunk = kDefaultSz↔
Chunk>
auto ticket::file::ManagedObject< T, Meta, szChunk >::update ( ) -> void   [inline]
```

updates a modified object.

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

- lib/file/file.h

6.30 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

6.30.1 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.

6.30.2 Member Typedef Documentation

6.30.2.1 const_iterator `template<typename KeyType , typename ValueType , typename Compare = internal::LessOp>`

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

6.30.2.2 iterator `template<typename KeyType , typename ValueType , typename Compare = internal↩ ::LessOp>`

```
using ticket::Map< KeyType, ValueType, Compare >::iterator = typename TreeType::iterator
```

6.30.2.3 value_type `template<typename KeyType , typename ValueType , typename Compare = internal↩ ::LessOp>`

```
using ticket::Map< KeyType, ValueType, Compare >::value_type = Pair<const KeyType, ValueType>
```

6.30.3 Constructor & Destructor Documentation

6.30.3.1 Map() `template<typename KeyType , typename ValueType , typename Compare = internal↩ ::LessOp>`

```
ticket::Map< KeyType, ValueType, Compare >::Map ( )  [default]
```

6.30.4 Member Function Documentation

6.30.4.1 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’

6.30.4.2 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]
```

6.30.4.3 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

6.30.4.4 cbegin()

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

6.30.4.5 cend()

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

6.30.4.6 clear()

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

clears the contents

6.30.4.7 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)

6.30.4.8 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.

6.30.4.9 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.

6.30.4.10 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)

6.30.4.11 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.

6.30.4.12 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]
```

6.30.4.13 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.

6.30.4.14 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.

6.30.4.15 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.

6.30.4.16 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

6.31 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

6.31.1 Member Data Documentation

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

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

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

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

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

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

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

- src/parser.h

6.32 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

6.32.1 Member Data Documentation

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

6.32.1.2 id `int ticket::rollback::ModifyProfile::id`

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

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

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

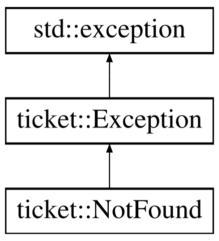
The documentation for this struct was generated from the following file:

- src/rollback.h

6.33 ticket::NotFound Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::NotFound:



Public Member Functions

- NotFound ()
- NotFound (const char ∗what)

6.33.1 Constructor & Destructor Documentation

6.33.1.1 NotFound() [1/2] `ticket::NotFound::NotFound ( ) [inline]`

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

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

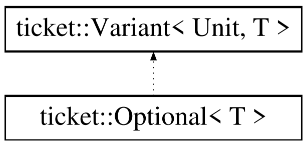
- lib/exception.h

6.34 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.
- Optional (const T &value)

  constructs a filled optional.
- auto operator= (const T &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 ∗

6.34.1 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.

6.34.2 Constructor & Destructor Documentation

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

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

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

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

constructs a empty optional.

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

```
ticket::Optional< T >::Optional (
             const T & value )  [inline]
```

constructs a filled optional.

6.34.3 Member Function Documentation

6.34.3.1 operator bool() `template<typename T >`

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

true if the optional has value.

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

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

provides access to the actual object.

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

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

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

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

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

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

6.34.3.6 operator=() `template<typename T >`

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

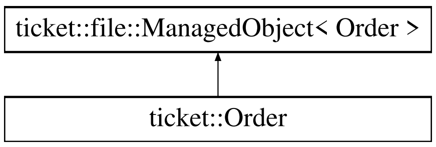
The documentation for this class was generated from the following file:

- lib/optional.h

6.35 ticket::Order Struct Reference

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

Inheritance diagram for ticket::Order:



Public Types

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

Public Member Functions

- auto getTrain () -> Train

  gets the corresponding train object.

Public Attributes

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

Additional Inherited Members

6.35.1 Member Typedef Documentation

6.35.1.1 Id `using ticket::Order::Id = int`

6.35.2 Member Enumeration Documentation

6.35.2.1 Status `enum ticket::Order::Status`

Enumerator

kSuccess	
kPending	
kRefunded	

6.35.3 Member Function Documentation

6.35.3.1 getTrain() `auto ticket::Order::getTrain ( ) -> Train`

gets the corresponding train object.

6.35.4 Member Data Documentation

6.35.4.1 ixFrom `int ticket::Order::ixFrom`

6.35.4.2 ixTo `int ticket::Order::ixTo`

6.35.4.3 ride `Ride ticket::Order::ride`

6.35.4.4 seats `int ticket::Order::seats`

6.35.4.5 status `Status ticket::Order::status`

6.35.4.6 user `User::Id ticket::Order::user`

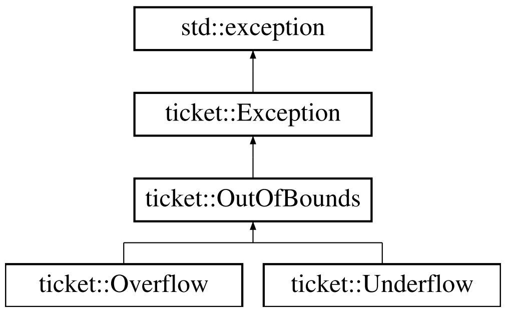
The documentation for this struct was generated from the following file:

- src/order.h

6.36 ticket::OutOfBounds Class Reference

```
#include <exception.h>
```

Inheritance diagram for ticket::OutOfBounds:



Public Member Functions

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

6.36.1 Constructor & Destructor Documentation

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

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

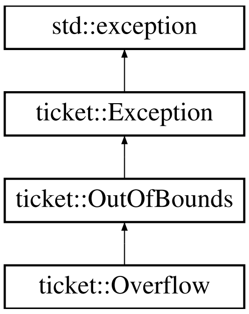
The documentation for this class was generated from the following file:

- lib/exception.h

6.37 ticket::Overflow Class Reference

```
#include <exception.h>
```

Inheritance diagram for ticket::Overflow:



Public Member Functions

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

6.37.1 Constructor & Destructor Documentation

6.37.1.1 Overflow() [1/2] `ticket::Overflow::Overflow ( ) [inline]`

6.37.1.2 Overflow() [2/2]

`ticket::Overflow::Overflow ( const char * what ) [inline]`

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

- lib/exception.h

6.38 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

6.38.1 Detailed Description

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

A pair of objects.

6.38.2 Constructor & Destructor Documentation

6.38.2.1 Pair() [1/7]

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

6.38.2.2 Pair() [2/7]

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

6.38.2.3 Pair() [3/7]

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

6.38.2.4 Pair() [4/7]

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

6.38.2.5 Pair() [5/7]

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

6.38.2.6 Pair() [6/7]

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

6.38.2.7 Pair() [7/7]

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

6.38.3 Member Data Documentation

6.38.3.1 first

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

6.38.3.2 second `template<typename T1 , typename T2 >`

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

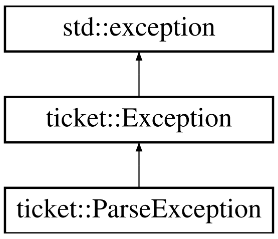
The documentation for this class was generated from the following file:

- lib/utility.h

6.39 ticket::ParseException Class Reference

`#include <exception.h>`

Inheritance diagram for ticket::ParseException:



Public Member Functions

- ParseException ()
- ParseException (const char ∗what)

6.39.1 Constructor & Destructor Documentation

6.39.1.1 ParseException() [1/2] `ticket::ParseException::ParseException ( ) [inline]`

6.39.1.2 ParseException() [2/2] `ticket::ParseException::ParseException (`

`const char * what ) [inline]`

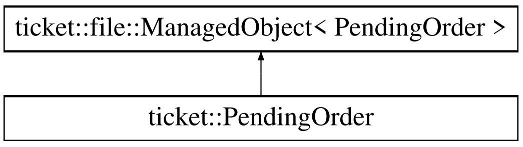
The documentation for this class was generated from the following file:

- lib/exception.h

6.40 ticket::PendingOrder Struct Reference

`#include <order.h>`

Inheritance diagram for ticket::PendingOrder:



Public Member Functions

- auto satisfiable () -> bool

  checks if the order is satisfiable.
- auto getOrder () -> Order

  gets the corresponding order object.

Public Attributes

- Ride ride
- int ixFrom
- int ixTo
- int seats
- Order::Id order

Additional Inherited Members

6.40.1 Member Function Documentation

6.40.1.1 getOrder() `auto ticket::PendingOrder::getOrder ( ) -> Order`

gets the corresponding order object.

6.40.1.2 satisfiable() `auto ticket::PendingOrder::satisfiable ( ) -> bool`

checks if the order is satisfiable.

6.40.2 Member Data Documentation

6.40.2.1 ixFrom `int ticket::PendingOrder::ixFrom`

6.40.2.2 ixTo `int ticket::PendingOrder::ixTo`

6.40.2.3 order `Order::Id ticket::PendingOrder::order`

6.40.2.4 ride `Ride ticket::PendingOrder::ride`

6.40.2.5 seats `int ticket::PendingOrder::seats`

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

- src/order.h

6.41 ticket::command::QueryOrder Struct Reference

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

Public Attributes

- std::string currentUser

6.41.1 Member Data Documentation

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

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

- src/parser.h

6.42 ticket::command::QueryProfile Struct Reference

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

Public Attributes

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

6.42.1 Member Data Documentation

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

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

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

- src/parser.h

6.43 ticket::command::QueryTicket Struct Reference

`#include <parser.h>`

Public Attributes

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

6.43.1 Member Data Documentation

6.43.1.1 date `Date ticket::command::QueryTicket::date`

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

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

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

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

- src/parser.h

6.44 ticket::command::QueryTrain Struct Reference

`#include <parser.h>`

Public Attributes

- std::string id
- Date date

6.44.1 Member Data Documentation

6.44.1.1 date `Date ticket::command::QueryTrain::date`

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

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

- src/parser.h

6.45 ticket::command::QueryTransfer Struct Reference

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

Public Attributes

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

6.45.1 Member Data Documentation

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

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

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

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

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

- src/parser.h

6.46 ticket::command::RefundTicket Struct Reference

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

Public Attributes

- std::string currentUser
- int index = 1

6.46.1 Member Data Documentation

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

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

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

- src/parser.h

6.47 ticket::rollback::RefundTicket Struct Reference

```
#include <rollback.h>
```

Public Attributes

- int id
- Order::Status status

6.47.1 Member Data Documentation

6.47.1.1 id `int ticket::rollback::RefundTicket::id`

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

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

- src/rollback.h

6.48 ticket::command::ReleaseTrain Struct Reference

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

Public Attributes

- std::string id

6.48.1 Member Data Documentation

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

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

- src/parser.h

6.49 ticket::rollback::ReleaseTrain Struct Reference

```
#include <rollback.h>
```

Public Attributes

- int id

6.49.1 Member Data Documentation

6.49.1.1 id `int ticket::rollback::ReleaseTrain::id`

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

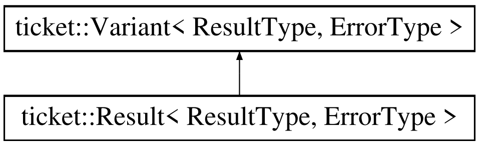
- src/rollback.h

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

Result<Res, Err> = Res | Err.

`#include <result.h>`

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


ticket::Variant< ResultType, ErrorType >
ticket::Result< ResultType, ErrorType >

Public Member Functions

- Result ()=delete
- template<typename 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.

6.50.1 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.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 Result() [1/2] template<typename ResultType , typename ErrorType >

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

6.50.2.2 Result() [2/2]

```
template<typename ResultType , typename ErrorType >
template<typename T >
ticket::Result< ResultType, ErrorType >::Result (
            const T & value )  [inline]
```

6.50.3 Member Function Documentation

6.50.3.1 error() [1/2]

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

6.50.3.2 error() [2/2]

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

6.50.3.3 result() [1/2]

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

6.50.3.4 result() [2/2]

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

6.50.3.5 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

6.51 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

6.51.1 Member Function Documentation

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

6.51.2 Member Data Documentation

6.51.2.1 date `Date ticket::Ride::date`

6.51.2.2 train `int ticket::Ride::train`

the numerical id of the train.

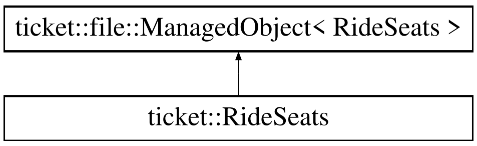
The documentation for this struct was generated from the following file:

- src/train.h

6.52 ticket::RideSeats Struct Reference

```
#include <train.h>
```

Inheritance diagram for ticket::RideSeats:



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

Additional Inherited Members

6.52.1 Member Function Documentation

6.52.1.1 ticketsAvailable()

```
auto ticket::RideSeats::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

6.52.2 Member Data Documentation

6.52.2.1 ride `Ride ticket::RideSeats::ride`

6.52.2.2 seatsRemaining `file::Array<int, 99> ticket::RideSeats::seatsRemaining`

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

- src/train.h

6.53 ticket::command::Rollback Struct Reference

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

Public Attributes

- int timestamp

6.53.1 Member Data Documentation

6.53.1.1 timestamp `int ticket::command::Rollback::timestamp`

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

- src/parser.h

6.54 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]

6.54.1 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.

6.54.2 Constructor & Destructor Documentation

6.54.2.1 Set()

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

6.54.3 Member Function Documentation

6.54.3.1 clear()

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

clears the set.

6.54.3.2 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.

6.54.3.3 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.

6.54.3.4 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.

6.54.3.5 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.

6.54.3.6 indexOfInsert() `template<typename T , size_t maxLength, typename Cmp = Less<>>`

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

6.54.3.7 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.

6.54.3.8 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]
```

6.54.3.9 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]
```

6.54.3.10 pop() `template<typename T , size_t maxLength, typename Cmp = Less<>>`

```
auto ticket::file::Set< T, maxLength, Cmp >::pop ( ) -> T   [inline]
```

pops the greatest element.

6.54.3.11 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.

6.54.3.12 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.

6.54.3.13 shift()

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

pops the least element.

6.54.4 Member Data Documentation

6.54.4.1 content

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

6.54.4.2 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

6.55 ticket::Train::Stop Struct Reference

```
#include <train.h>
```

Public Attributes

- Station::Id name

6.55.1 Member Data Documentation

6.55.1.1 name `Station::Id ticket::Train::Stop::name`

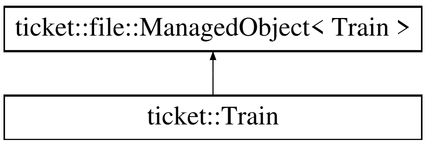
The documentation for this struct was generated from the following file:

- src/train.h

6.56 ticket::Train Struct Reference

`#include <train.h>`

Inheritance diagram for ticket::Train:



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

Additional Inherited Members

6.56.1 Member Typedef Documentation

6.56.1.1 Id `using ticket::Train::Id = file::Varchar<20>`

6.56.1.2 Type `using ticket::Train::Type = char`

6.56.2 Member Function Documentation

6.56.2.1 getRide() [1/2]

```
auto ticket::Train::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).

6.56.2.2 getRide() [2/2]

```
auto ticket::Train::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.

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

finds the index of the station of the given name.

6.56.2.4 runsOnDate() [1/2] `auto ticket::Train::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.

6.56.2.5 runsOnDate() [2/2] `auto ticket::Train::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.

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

calculates the total price of a trip.

6.56.3 Member Data Documentation

6.56.3.1 begin `Date ticket::Train::begin`

6.56.3.2 edges `file::Array<Edge, 99> ticket::Train::edges`

6.56.3.3 end `Date ticket::Train::end`

6.56.3.4 released `bool ticket::Train::released`

6.56.3.5 seats `int ticket::Train::seats`

6.56.3.6 stops `file::Array<Stop, 100> ticket::Train::stops`

6.56.3.7 trainId `Id ticket::Train::trainId`

6.56.3.8 type `Type ticket::Train::type`

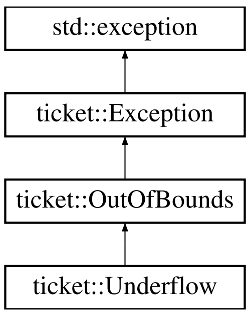
The documentation for this struct was generated from the following file:

- src/train.h

6.57 ticket::Underflow Class Reference

```
#include <exception.h>
```

Inheritance diagram for ticket::Underflow:



Public Member Functions

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

6.57.1 Constructor & Destructor Documentation

6.57.1.1 Underflow() [1/2] `ticket::Underflow::Underflow ( ) [inline]`

6.57.1.2 Underflow() [2/2] `ticket::Underflow::Underflow ( const char ∗ what ) [inline]`

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

- lib/exception.h

6.58 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

6.58.1 Detailed Description

An empty class, used at various places.

6.58.2 Constructor & Destructor Documentation

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

6.58.2.2 Unit() [2/2]

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

6.58.3 Member Function Documentation

6.58.3.1 operator<()

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

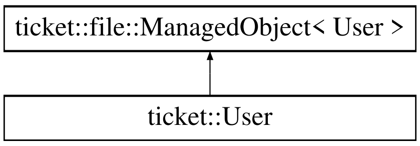
The documentation for this struct was generated from the following file:

- lib/utility.h

6.59 ticket::User Struct Reference

```
#include <user.h>
```

Inheritance diagram for ticket::User:



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 hasUser (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

Additional Inherited Members

6.59.1 Member Typedef Documentation

6.59.1.1 Email `using ticket::User::Email = file::Varchar<30>`

6.59.1.2 Id `using ticket::User::Id = file::Varchar<20>`

6.59.1.3 Name `using ticket::User::Name = file::Varchar<15>`

6.59.1.4 Password `using ticket::User::Password = file::Varchar<30>`

6.59.1.5 Privilege `using ticket::User::Privilege = int`

6.59.2 Member Function Documentation

6.59.2.1 hasUser()

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

checks if there is a user with the given username.

6.59.3 Member Data Documentation

6.59.3.1 email `Email ticket::User::email`

6.59.3.2 name `Name ticket::User::name`

6.59.3.3 password `Password ticket::User::password`

6.59.3.4 privilege `Privilege ticket::User::privilege`

6.59.3.5 username `Id ticket::User::username`

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

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

6.60 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 () -> 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

Friends

- template<int A>
  class Varchar

6.60.1 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.

6.60.2 Constructor & Destructor Documentation

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

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

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

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

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

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

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

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

6.60.3 Member Function Documentation

6.60.3.1 hash() template<int maxLength>

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

6.60.3.2 length() `template<int maxLength>`

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

6.60.3.3 operator std::string() `template<int maxLength>`

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

6.60.3.4 operator"!=() `template<int maxLength>`

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

6.60.3.5 operator<() `template<int maxLength>`

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

6.60.3.6 operator=() `template<int maxLength>`

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

6.60.3.7 operator==() `template<int maxLength>`

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

6.60.3.8 str() `template<int maxLength>`

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

6.60.4 Friends And Related Function Documentation

6.60.4.1 Varchar `template<int maxLength>`

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

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

- lib/file/varchar.h

6.61 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.

6.61.1 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.

6.61.2 Constructor & Destructor Documentation

6.61.2.1 Variant() [1/4]

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

6.61.2.2 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.

6.61.2.3 Variant() [3/4]

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

6.61.2.4 Variant() [4/4]

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

6.61.2.5 ∼Variant()

```
template<typename ... Ts>
virtual ticket::Variant< Ts >::∼Variant ( ) [inline], [virtual]
```

6.61.3 Member Function Documentation

6.61.3.1 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.

6.61.3.2 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.

6.61.3.3 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.

6.61.3.4 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.

6.61.3.5 index()

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

returns the current index of the current state.

6.61.3.6 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.

6.61.3.7 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.

6.61.3.8 operator=() [2/3]

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

6.61.3.9 operator=() [3/3]

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

6.61.3.10 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

6.62 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

6.62.1 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.

6.62.2 Constructor & Destructor Documentation

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

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

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

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

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

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

6.62.2.4 ∼Vector() template<typename T >

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

6.62.3 Member Function Documentation

6.62.3.1 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)

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

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

6.62.3.3 back() template<typename T >

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

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

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

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

returns an iterator to the beginning.

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

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

6.62.3.6 cbegin() `template<typename T >`

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

6.62.3.7 cend() `template<typename T >`

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

6.62.3.8 clear() `template<typename T >`

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

clears the contents

6.62.3.9 empty() `template<typename T >`

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

checks whether the container is empty

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

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

returns an iterator to the end.

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

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

6.62.3.12 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

6.62.3.13 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.

6.62.3.14 front() `template<typename T >`

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

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

6.62.3.15 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.)

6.62.3.16 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.

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

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

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

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

6.62.3.19 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.

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

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

6.62.3.21 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

6.62.3.22 push_back() template<typename T >

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

adds an element to the end.

6.62.3.23 reserve() `template<typename T >`

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

6.62.3.24 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.

```
// This file includes some common algorithms.
#ifndef TICKET_LIB_ALGORITHM_H_
#define TICKET_LIB_ALGORITHM_H_

#include <iostream>

#include "utility.h"

namespace ticket {

using std::distance, std::advance;

#define TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(name, cf) \
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; \
}
TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(upperBound, geq)
TICKET_ALGORIGHM_DEFINE_BOUND_FUNC(lowerBound, gt)
#undef TICKET_ALGORIGHM_DEFINE_BOUND_FUNC

} // namespace ticket

#endif // TICKET_LIB_ALGORITHM_H_
```

7.3 lib/datetime.cpp File Reference

```
#include "datetime.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.

```
// This file includes date and time utilities.
#ifndef TICKET_LIB_DATETIME_H_
#define TICKET_LIB_DATETIME_H_

#include <iostream>

namespace ticket {

class Date {
 public:
  Date () = default;
  Date (int month, int date);
  explicit Date (const char *str);
  auto month () const -> int;
  auto date () const -> int;
  operator std::string () const;
  auto operator+ (int dt) const -> Date;
  auto operator- (int dt) const -> Date;
  auto operator- (Date rhs) const -> int;
  auto operator< (const Date &rhs) const -> bool;
  auto inRange (Date begin, Date end) const -> bool;
 private:
  int days_ = 0;
};

class Duration {
 public:
  Duration () = default;
  Duration (int hour, int minute);
  explicit Duration (int minutes);
  explicit Duration (const char *str);
  auto hours () const -> int;
  auto minutes () const -> int;
  auto totalMinutes () const -> int;
  auto operator+ (Duration dt) const -> Duration;
  auto operator- (Duration dt) const -> Duration;
  auto operator- () const -> Duration;
  auto operator< (const Duration &rhs) const -> bool;
 private:
  int minutes_ = 0;
};

class Instant {
 public:
  Instant () = default;
  Instant (int hour, int minute);
  explicit Instant (const char *str);
  auto daysOverflow () const -> int;
  auto hour () const -> int;
  auto minute () const -> int;
  operator std::string () const;
  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;
 private:
  int minutes_ = 0;
};

} // namespace ticket

#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.

```

#ifndef TICKET_LIB_EXCEPTION_H_
#define TICKET_LIB_EXCEPTION_H_

#include <iostream>

namespace ticket {

class Exception : public std::exception {
 public:
  Exception () = default;
  Exception (const char *what) : what_(what) {}
  virtual ~Exception () = default;
  virtual auto what () const noexcept -> const char * {
    return what_;
  }
 private:
  const char * const what_ = "unknown exception";
};

class IoException : public Exception {
 public:
  IoException () : Exception("IO exception") {}
  IoException (const char *what) : Exception(what) {}
};

class OutOfBounds : public Exception {
 public:
  OutOfBounds () : Exception("out of bounds") {}
  OutOfBounds (const char *what) : Exception(what) {}
};

class Overflow : public OutOfBounds {
 public:
  Overflow () : OutOfBounds("overflow") {}
  Overflow (const char *what) : OutOfBounds(what) {}
};

class Underflow : public OutOfBounds {
 public:
  Underflow () : OutOfBounds("underflow") {}
  Underflow (const char *what) : OutOfBounds(what) {}
};

class NotFound : public Exception {
 public:
  NotFound () : Exception("underflow") {}
  NotFound (const char *what) : Exception(what) {}
};

class ParseException : public Exception {
 public:
  ParseException () : Exception("parse exception") {}
  ParseException (const char *what) : Exception(what) {}
};

} // namespace ticket

#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.

```
#ifndef TICKET_LIB_FILE_ARRAY_H_
#define TICKET_LIB_FILE_ARRAY_H_

#include <cstring>

#include "exception.h"
#include "utility.h"

namespace ticket::file {

template <typename T, size_t maxLength, typename Cmp = Less<»
struct Array {
 private:
  auto boundsCheck_ (size_t index) -> void {
    if (index >= length) throw OutOfBounds("Array: overflow or underflow");
  }
  Cmp cmp_;
 public:
  size_t length = 0;
  T content[maxLength];
  auto indexOf (const T &element) -> size_t {
    for (size_t i = 0; i < length; ++i) {
      if (cmp_.equals(element, content[i])) return i;
    }
    throw NotFound("Array::indexOf: element not found");
  }
  auto includes (const T &element) -> bool {
    for (size_t i = 0; i < length; ++i) {
      if (cmp_.equals(element, content[i])) return true;
    }
    return false;
  }
  auto insert (const T &element, size_t offset) -> void {
    if (offset != length) boundsCheck_(offset);
    if (length == maxLength) {
      throw Overflow("Array::insert: overflow");
    }
    if (offset != length) {
      memmove(
        &content[offset + 1],
        &content[offset],
        (length - offset) * sizeof(content[0])
      );
    }
    content[offset] = element;
    ++length;
  }

  auto remove (const T &element) -> void {
    removeAt(indexOf(element));
  }
  auto removeAt (size_t offset) -> void {
    boundsCheck_(offset);
    if (offset != length - 1) {
      memmove(
        &content[offset],
        &content[offset + 1],
        (length - offset - 1) * sizeof(content[0])
      );
    }
    --length;
  }
  auto clear () -> void { length = 0; }

  auto copyFrom (
    const Array &other,
    size_t ixFrom,
    size_t ixTo,
    size_t count
  ) -> void {
    if (this == &other) {
      memmove(
        &content[ixTo],
        &content[ixFrom],
        count * sizeof(content[0])
      );
    } else {
      memcpy(
        &content[ixTo],
        &other.content[ixFrom],
        count * sizeof(content[0])
      );
    }
  }

  auto operator[] (size_t index) -> T & {
    boundsCheck_(index);
    return content[index];
  }
  auto operator[] (size_t index) const -> const T & {
    boundsCheck_(index);
    return content[index];
  }

  auto pop () -> T {
    if (length == 0) throw Underflow("Array::pop: underflow");
    return content[--length];
  }
  auto shift () -> T {
    if (length == 0) throw Underflow("Array::pop: underflow");
    T result = content[0];
    removeAt(0);
    return result;
  }
  auto push (const T &object) -> void { insert(object, length); }
  auto unshift (const T &object) -> void { insert(object, 0); }

  template <typename Functor>
  auto forEach (const Functor &callback) -> T {
    for (size_t i = 0; i < length; ++i) callback(content[i]);
  }
};

} // namespace ticket::file

#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/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.

```
#ifndef TICKET_LIB_FILE_BPTREE_H_
#define TICKET_LIB_FILE_BPTREE_H_

#include <cstring>

#include "algorithm.h"
#include "file/array.h"
#include "file/file.h"
#include "file/set.h"
#include "optional.h"
#include "utility.h"
#include "vector.h"

#ifdef TICKET_DEBUG
#include <iostream>
#endif

namespace ticket::file {

template <
  typename KeyType,
  typename ValueType,
  typename CmpKey = Less<>,
  typename CmpValue = Less<>,
  typename Meta = Unit,
  size_t szChunk = kDefaultSzChunk
>
class BpTree {
 private:
  struct Node;
 public:
  BpTree (const char *filename) : file_(filename, [this] () { this->init_(); }) {}
  auto insert (const KeyType &key, const ValueType &value) -> void {
    Node root = Node::root(*this);
    insert_({ .key = key, .value = value }, root);
    if (root.shouldSplit()) split_(root, root, 0);
    root.update();
  }
  auto remove (const KeyType &key, const ValueType &value) -> void {
    Node root = Node::root(*this);
    remove_({ .key = key, .value = value }, root);
    if (root.shouldMerge()) merge_(root, root, 0);
    root.update();
  }
  auto findOne (const KeyType &key) -> Optional<ValueType> {
    return findOne_(key, Node::root(*this));
  }
  auto findMany (const KeyType &key) -> Vector<ValueType> {
    return findMany_(key, Node::root(*this));
  }
  auto findAll () -> Vector<ticket::Pair<KeyType, ValueType>> {
    return findAll_(Node::root(*this));
  }
  auto includes (const KeyType &key, const ValueType &value) -> bool {
    return includes_({ .key = key, .value = value }, Node::root(*this));
  }

  auto getMeta () -> Meta {
    return file_.getMeta();
  }
  auto setMeta (const Meta &meta) -> void {
    return file_.setMeta(meta);
  }

  auto clearCache () -> void { file_.clearCache(); }

#ifdef TICKET_DEBUG
  auto print () -> void { print_(Node::root(*this)); }
#endif

 private:
  File<Meta, szChunk> file_;
  CmpKey cmpKey_;
  CmpValue cmpValue_;

  // data structures
  struct Pair {
    KeyType key;
    ValueType value;
    auto operator< (const Pair &that) const -> bool {
      CmpKey cmpKey_;
      CmpValue cmpValue_;
      if (!cmpKey_.equals(key, that.key)) return cmpKey_.lt(key, that.key);
      return cmpValue_.lt(value, that.value);
    }
  };
  class KeyComparatorLess_ {
   public:
    auto operator() (const Pair &lhs, const KeyType &rhs) -> bool {
      return cmpKey_.lt(lhs.key, rhs);
    }
    auto operator() (const KeyType &lhs, const Pair &rhs) -> bool {
      return cmpKey_.geq(rhs.key, lhs);
    }
   private:
    CmpKey cmpKey_;
    CmpValue cmpValue_;
  };

  using NodeId = unsigned int;
  // ROOT and INTERMEDIATE nodes are index nodes
  enum NodeType { kRoot, kIntermediate, kRecord };
  // if k > kLengthMax, there must be an overflow.
  static constexpr size_t kLengthMax = 18446744073709000000ULL;
  struct IndexPayload {
    static constexpr size_t k = (szChunk - 2 * sizeof(NodeId)) / (sizeof(NodeId) + sizeof(Pair)) / 2 -
   1;
    static_assert(k >= 2 && k < kLengthMax);
    bool leaf = false;
    Array<NodeId, 2 * k> children;
    Set<Pair, 2 * k> splits;
  };
  struct RecordPayload {
    static constexpr size_t l = (szChunk - 3 * sizeof(NodeId)) / sizeof(Pair) / 2 - 1;
    static_assert(l >= 2 && l < kLengthMax);
    NodeId prev = 0;
    NodeId next = 0;
    Set<Pair, 2 * l> entries;
  };
  union NodePayload {
    IndexPayload index;
    RecordPayload record;
    NodePayload () {} // NOLINT
  };
  struct Node : public ManagedObject<Node, Meta, szChunk> {
    NodeType type;
    NodePayload payload;
    static_assert(sizeof(NodeType) + sizeof(NodePayload) <= szChunk);

    // dynamically type-safe accessors
    auto leaf () -> bool & { TICKET_ASSERT(type != kRecord); return payload.index.leaf; }
    auto children () -> Array<NodeId, 2 * IndexPayload::k> & { TICKET_ASSERT(type != kRecord); return
   payload.index.children; }
    auto splits () -> Set<Pair, 2 * IndexPayload::k> & { TICKET_ASSERT(type != kRecord); return
   payload.index.splits; }
    auto prev () -> NodeId & { TICKET_ASSERT(type == kRecord); return payload.record.prev; }
    auto next () -> NodeId & { TICKET_ASSERT(type == kRecord); return payload.record.next; }
    auto entries () -> Set<Pair, 2 * RecordPayload::l> & { TICKET_ASSERT(type == kRecord); return
   payload.record.entries; }

    Node (BpTree &tree, NodeType type) : ManagedObject<Node, Meta, szChunk>(tree.file_), type(type) {
      if (type == kRecord) {
        new(&payload.record) RecordPayload;
      } else {
        new(&payload.index) IndexPayload;
      }
    }
    ~Node () {
      if (type == kRecord) {
        payload.record.~RecordPayload();
      } else {
        payload.index.~IndexPayload();
      }
    }

    static auto root (BpTree &tree) -> Node { return Node::get(tree.file_, 0); }

    auto halfLimit () -> size_t {
      return type == kRecord ? RecordPayload::l : IndexPayload::k;
    }
    auto length () -> size_t {
      return type == kRecord ? payload.record.entries.length : payload.index.children.length;
    }
    auto shouldSplit () -> bool { return length() == 2 * halfLimit(); }
    auto shouldMerge () -> bool { return length() < halfLimit(); }
    auto lowerBound () -> Pair {
      return type == kRecord ? payload.record.entries[0] : payload.index.splits[0];
    }
  };

  // helper functions
  auto ixInsert_ (const Pair &entry, Node &node) -> size_t {
    TICKET_ASSERT(node.type != kRecord);
    auto &splits = node.splits();
    size_t ix = upperBound(splits.content, splits.content + splits.length, entry) - splits.content;
    return ix == 0 ? ix : ix - 1;
  }
  auto splitRoot_ (Node &node) -> void {
    Node left(*this, kIntermediate), right(*this, kIntermediate);

    // copy children and splits
    left.children().copyFrom(node.children(), 0, 0, IndexPayload::k);
    left.splits().copyFrom(node.splits(), 0, 0, IndexPayload::k);
    right.children().copyFrom(node.children(), IndexPayload::k, 0, IndexPayload::k);
    right.splits().copyFrom(node.splits(), IndexPayload::k, 0, IndexPayload::k);
    left.children().length = left.splits().length = right.children().length = right.splits().length =
   IndexPayload::k;

    // set misc properties and save
    left.leaf() = right.leaf() = node.leaf();
    node.leaf() = false;
    left.save();
    right.save();

    // initiate the new root node
    node.children().clear();
    node.children().insert(left.id(), 0);
    node.children().insert(right.id(), 1);
    node.splits().clear();
    node.splits().insert(left.lowerBound());
    node.splits().insert(right.lowerBound());
  }
  auto split_ (Node &node, Node &parent, size_t ixChild) -> void {
    TICKET_ASSERT(node.shouldSplit());
#ifdef TICKET_DEBUG_BPTREE
    std::cerr « "[Split] " « node.id() « " (parent " « parent.id() « ")" « std::endl;
#endif
    if (node.type == kRoot) {
      // the split of the root node is a bit different from other nodes. it produces two extra subnodes.
      splitRoot_(node);
      return;
    }
    TICKET_ASSERT(node.type != kRoot);

    // create a new next node
    Node next(*this, node.type);
    if (node.type == kIntermediate) {
      next.children().copyFrom(node.children(), IndexPayload::k, 0, IndexPayload::k);
      next.splits().copyFrom(node.splits(), IndexPayload::k, 0, IndexPayload::k);
      node.children().length = node.splits().length = next.children().length = next.splits().length =
   IndexPayload::k;
      next.leaf() = node.leaf();
      next.save();
    } else {
      TICKET_ASSERT(node.type == kRecord);
      next.next() = node.next();
      next.prev() = node.id();
      memmove(
        next.entries().content,
        &node.entries().content[RecordPayload::l],
        RecordPayload::l * sizeof(node.entries()[0])
      );
      next.entries().length = node.entries().length = RecordPayload::l;
      next.save();
      if (next.next() != 0) {
        Node nextnext = Node::get(file_, next.next());
        nextnext.prev() = next.id();
        nextnext.update();
      }
      node.next() = next.id();
    }

    // update the parent node
    parent.children().insert(next.id(), ixChild + 1);
    parent.splits().insert(next.lowerBound());
  }

  template <typename A, typename B>
  static auto unshift_ (A &to, B &from, size_t k) -> void {
    // 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]]
    to.copyFrom(to, 0, k, k - 1);
    to.copyFrom(from, 0, 0, k);
    to.length += from.length;
    from.length = 0;
  }
  template <typename A, typename B>
  static auto push_ (A &to, B &from, size_t k) -> void {
    to.copyFrom(from, 0, k - 1, k);
    to.length += from.length;
    from.length = 0;
  }
  auto merge_ (Node &node, Node &parent, size_t ixChild) -> void {
    TICKET_ASSERT(node.shouldMerge());
#ifdef TICKET_DEBUG_BPTREE
    std::cerr « "[Merge] " « node.id() « " (parent " « parent.id() « ")" « std::endl;
#endif
    if (node.type == kRoot) {
      if (node.length() > 1 || node.leaf()) return;
      Node onlyChild = Node::get(file_, node.children()[0]);
      memcpy(&node, &onlyChild, sizeof(node));
      node.type = kRoot;
      return;
    }
    const bool hasPrev = ixChild != 0;
    const bool hasNext = ixChild != parent.children().length - 1;
    if (!hasNext) {
      // don't do anything to the only data node.
      if (!hasPrev && node.type == kRecord) return;
      // all index nodes has at least 2 child nodes, except for the root node.
      TICKET_ASSERT(hasPrev);
      Node prev = Node::get(file_, parent.children()[ixChild - 1]);
      if (prev.length() > prev.halfLimit()) {
        if (node.type == kRecord) {
          node.entries().insert(prev.entries().pop());
        } else {
          node.children().unshift(prev.children().pop());
          node.splits().insert(prev.splits().pop());
        }
        prev.update();
        parent.splits()[ixChild] = node.lowerBound();
        return;
      }
      TICKET_ASSERT(prev.length() == prev.halfLimit());

      if (node.type == kRecord) {
        unshift_(node.entries(), prev.entries(), RecordPayload::l);
        if (prev.prev() != 0) {
          Node prevprev = Node::get(file_, prev.prev());
          prevprev.next() = node.id();
          prevprev.update();
        }
        node.prev() = prev.prev();
      } else {
        TICKET_ASSERT(node.type == kIntermediate);
        unshift_(node.children(), prev.children(), IndexPayload::k);
        unshift_(node.splits(), prev.splits(), IndexPayload::k);
      }
      parent.splits()[ixChild] = node.lowerBound();
      parent.children().removeAt(ixChild - 1);
      parent.splits().removeAt(ixChild - 1);
      prev.destroy();
      return;
    }
    TICKET_ASSERT(hasNext);

    // FIXME: remove dupe code here
    Node next = Node::get(file_, parent.children()[ixChild + 1]);
    if (next.length() > next.halfLimit()) {
      if (node.type == kRecord) {
        node.entries().insert(next.entries().shift());
      } else {
        node.children().push(next.children().shift());
        node.splits().insert(next.splits().shift());
      }
      next.update();
      parent.splits()[ixChild + 1] = next.lowerBound();
      return;
    }
    TICKET_ASSERT(next.length() == next.halfLimit());

    if (node.type == kRecord) {
      push_(node.entries(), next.entries(), RecordPayload::l);
      if (next.next() != 0) {
        Node nextnext = Node::get(file_, next.next());
        nextnext.prev() = node.id();
        nextnext.update();
      }
      node.next() = next.next();
    } else {
      TICKET_ASSERT(node.type == kIntermediate);
      push_(node.children(), next.children(), IndexPayload::k);
      push_(node.splits(), next.splits(), IndexPayload::k);
    }

    parent.children().removeAt(ixChild + 1);
    parent.splits().removeAt(ixChild + 1);
    next.destroy();
  }

  // FIXME: lengthy function name
  auto addValuesToVectorForAllKeyFrom_ (Vector<ValueType> &vec, const KeyType &key, Node node, int first) -> void {
    // we need to declare i outside to see if we have advanced to the last element
    int i = first;
    for (; i < node.length() && cmpKey_.equals(node.entries()[i].key, key); ++i) vec.push_back(node.entries()[i].value);
    if (i == node.length() && node.next() != 0) addValuesToVectorForAllKeyFrom_(vec, key, Node::get(file_, node.next()), 0);
  }
  auto addEntriesToVector_ (Vector<ticket::Pair<KeyType, ValueType>> &vec, Node node) -> void {
    for (int i = 0; i < node.length(); ++i) vec.emplace_back(node.entries()[i].key, node.entries()[i].value);
    if (node.next() != 0) addEntriesToVector_(vec, Node::get(file_, node.next()));
  }
  auto findFirstChildWithKey_ (const KeyType &key, Node &node) -> ticket::Pair<Node, Optional<Node>> {
    TICKET_ASSERT(node.type != kRecord);
    size_t ixGreater = upperBound(
      node.splits().content,
      node.splits().content + node.length(),
      key,
      Less<KeyComparatorLess_>()
    ) - node.splits().content;
    bool hasCdr = ixGreater < node.length() && cmpKey_.equals(node.splits()[ixGreater].key, key);
    auto cdr = hasCdr ? Optional<Node>(Node::get(file_, node.children()[ixGreater])) : Optional<Node>(unit);
    size_t ix = ixGreater == 0 ? ixGreater : ixGreater - 1;
    return { Node::get(file_, node.children()[ix]), cdr };
  }

  // operation functions
  auto insert_ (const Pair &entry, Node &node) -> void {
    if (node.type == kRecord) {
      node.entries().insert(entry);
      TICKET_ASSERT(node.entries().length <= 2 * RecordPayload::l);
      return;
    }
    // if this is the first entry of the root, go create a record node.
    if (node.children().length == 0) {
      TICKET_ASSERT(node.type == kRoot);
      TICKET_ASSERT(node.leaf());
      Node child(*this, kRecord);
      child.entries().insert(entry);
      child.save();
      node.children().push(child.id());
      node.splits().insert(entry);
      return;
    }
    size_t ix = ixInsert_(entry, node);
    if (entry < node.splits()[ix]) node.splits()[ix] = entry;
    Node nodeToInsert = Node::get(file_, node.children()[ix]);
    insert_(entry, nodeToInsert);
    node.splits()[ix] = nodeToInsert.lowerBound();
    if (nodeToInsert.shouldSplit()) split_(nodeToInsert, node, ix);
    nodeToInsert.update();
  }
  auto remove_ (const Pair &entry, Node &node) -> void {
    if (node.type == kRecord) {
      node.entries().remove(entry);
      return;
    }
    size_t ix = ixInsert_(entry, node);
    Node child = Node::get(file_, node.children()[ix]);
    remove_(entry, child);
    if (child.length() == 0) {
      TICKET_ASSERT(node.type == kRoot);
      TICKET_ASSERT(child.type == kRecord);
      child.destroy();
      node.children().clear();
      node.splits().clear();
      return;
    }
    node.splits()[ix] = child.lowerBound();
    if (child.shouldMerge()) merge_(child, node, ix);
    child.update();
  }
  auto findOne_ (const KeyType &key, Node node) -> Optional<ValueType> {
    if (node.type != kRecord) {
      if (node.length() == 0) return unit;
      auto [ car, cdr ] = findFirstChildWithKey_(key, node);
      if (!cdr) return findOne_(key, car);
      auto res = findOne_(key, car);
      if (res) return res;
      return findOne_(key, *cdr);
    }
    size_t ix = upperBound(
      node.entries().content,
      node.entries().content + node.length(),
      key,
      Less<KeyComparatorLess_>()
    ) - node.entries().content;
    if (ix >= node.length()) return unit;
    Pair entry = node.entries()[ix];
    if (!cmpKey_.equals(entry.key, key)) return unit;
    return entry.value;
  }
  auto includes_ (const Pair &entry, Node node) -> bool {
    if (node.type == kRecord) return node.entries().includes(entry);
    if (node.length() == 0) return false;
    return includes_(entry, Node::get(file_, node.children()[ixInsert_(entry, node)]));
  }
  auto findMany_ (const KeyType &key, Node node) -> Vector<ValueType> {
    if (node.type != kRecord) {
      if (node.length() == 0) return {};
      auto [ car, cdr ] = findFirstChildWithKey_(key, node);
      if (!cdr) return findMany_(key, car);
      Vector<ValueType> res = findMany_(key, car);
      if (!res.empty()) return res;
      return findMany_(key, *cdr);
    }
    size_t ix = upperBound(
      node.entries().content,
      node.entries().content + node.length(),
      key,
      Less<KeyComparatorLess_>()
    ) - node.entries().content;
    if (ix >= node.length()) return {};
    Vector<ValueType> res;
    addValuesToVectorForAllKeyFrom_(res, key, node, ix);
    return res;
  }
  auto findAll_ (Node node) -> Vector<ticket::Pair<KeyType, ValueType» {
    if (node.type != kRecord) {
      if (node.length() == 0) return {};
      return findAll_(Node::get(file_, node.children()[0]));
    }
    Vector<ticket::Pair<KeyType, ValueType» res;
    addEntriesToVector_(res, node);
    return res;
  }
  auto init_ () -> void {
    Node root(*this, kRoot);
    root.leaf() = true;
    root.save();
    TICKET_ASSERT(root.id() == 0);
  }
#ifdef TICKET_DEBUG
  auto print_ (Node node) -> void {
    if (node.type == RECORD) {
      std::cerr « "[Record " « node.id() « " (" « node.length() « "/" « 2 * RecordPayload::l - 1 « ")]";
      for (int i = 0; i < node.length(); ++i) std::cerr « " (" « std::string(node.entries()[i].key) « ",
    " « node.entries()[i].value « ")";
      std::cerr « std::endl;
      return;
    }
    std::cerr « "[Node " « node.id() « " (" « node.length() « "/" « 2 * IndexPayload::k - 1 « ")" «
   (node.leaf() ? " leaf" : "") « "]";
```

```
    for (int i = 0; i < node.length(); ++i) std::cerr « " (" « std::string(node.splits()[i].key) « ", "
   « node.splits()[i].value « ") " « node.children()[i];
    std::cerr « std::endl;
    for (int i = 0; i < node.length(); ++i) print_(Node::get(file_, node.children()[i]));
  }
#endif
};

} // namespace ticket::file

#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::ManagedObject< T, Meta, szChunk >

  an opinionated utility base class 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.

```
// This file defines several basic file-based utilities.
#ifndef TICKET_LIB_FILE_FILE_H_
#define TICKET_LIB_FILE_FILE_H_

#include <cstring>
#include <fstream>

#include "hashmap.h"
#include "utility.h"
#include "exception.h"

namespace ticket::file {

constexpr size_t kDefaultSzChunk = 4096;

template <typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
class File {
 private:
  class Metadata;
```

```
public:
 template <typename Functor>
 File (const char *filename, const Functor &initializer) {
   init_(filename, initializer);
 }
 File (const char *filename) {
   init_(filename, [] {});
 }
 ~File () { clearCache(); }

 auto get (void *buf, size_t index, size_t n) -> void {
   if (index != -1 && cache_.count(index) > 0) {
     memcpy(buf, cache_[index], n);
     return;
   }
   file_.seekg(offset_(index));
   file_.read((char *) buf, n);
   TICKET_ASSERT(file_.good());
   if (index != -1) putCache_(buf, index, n);
 }
 auto set (const void *buf, size_t index, size_t n) -> void {
   if (index != -1) {
     // dirty check
     if (cache_.count(index) > 0 && memcmp(buf, cache_[index], n) == 0) return;
     putCache_(buf, index, n);
   }
   file_.seekp(offset_(index));
   file_.write((const char *) buf, n);
   TICKET_ASSERT(file_.good());
 }
 auto push (const void *buf, size_t n) -> size_t {
   Metadata meta = meta_();
   size_t id = meta.next;
   if (meta.hasNext) {
     Metadata nextMeta;
     get(&nextMeta, meta.next, sizeof(nextMeta));
     set(&nextMeta, -1, sizeof(nextMeta));
   } else {
     ++meta.next;
     set(&meta, -1, sizeof(meta));
   }
   set(buf, id, n);
   return id;
 }
 auto remove (size_t index) -> void {
   Metadata meta = meta_();
   set(&meta, index, sizeof(meta));
   Metadata newMeta(index, true);
   set(&newMeta, -1, sizeof(newMeta));
   if (cache_.count(index) > 0) delete[] cache_[index];
   cache_.erase(cache_.find(index));
 }

 auto getMeta () -> Meta {
   return meta_().user;
 }
 auto setMeta (const Meta &user) -> void {
   Metadata meta = meta_();
   meta.user = user;
   set(&meta, -1, sizeof(meta));
 }

 auto clearCache () -> void {
   for (const auto &[ _, ptr ] : cache_) delete[] ptr;
   cache_.clear();
 }

private:
 struct Metadata {
   size_t next;
   bool hasNext;
   Meta user;
   Metadata () = default;
   Metadata (size_t next, bool hasNext) : next(next), hasNext(hasNext) {}
 };
 static_assert(szChunk > sizeof(Metadata));

 template <typename Functor>
 auto init_ (const char *filename, const Functor &initializer) -> void {
   bool shouldCreate = false;
   auto testFile = fopen(filename, "r");
   if (testFile == nullptr) {
     if (errno != ENOENT) {
       throw IoException("Unable to open file");
     }
     shouldCreate = true;
   } else if (fclose(testFile)) {
      throw IoException("Unable to close file");
    }
    if (shouldCreate) {
      auto file = fopen(filename, "w+");
      if (file == nullptr) {
        throw IoException("Unable to create file");
      }
      if (fclose(file)) {
        throw IoException("Unable to close file when creating file");
      }
    }
    file_.open(filename);
    if (!file_.is_open() || !file_.good()) {
      throw IoException("Unable to open file");
    }
    if (shouldCreate) {
      Metadata meta(0, false);
      set(&meta, -1, sizeof(meta));
      initializer();
    }
  }

  auto meta_ () -> Metadata {
    Metadata retval;
    get(&retval, -1, sizeof(retval));
    return retval;
  }
  auto offset_ (size_t index) -> size_t {
    return (index + 1) * szChunk;
  }
  std::fstream file_;
  HashMap<size_t, char *> cache_;
  auto putCache_ (const void *buf, size_t index, size_t n) -> void {
    char *cache = new char[n];
    memcpy(cache, buf, n);
    if (cache_.count(index) > 0) delete[] cache_[index];
    cache_[index] = cache;
  }
};

template <typename T, typename Meta = Unit, size_t szChunk = kDefaultSzChunk>
class ManagedObject {
 private:
  using File_ = File<Meta, szChunk>;
 public:
  ManagedObject (File_ &file) : file_(&file) {}
  virtual ~ManagedObject () = default;

  auto id () -> size_t { return id_; }

  static auto get (File_ &file, size_t id) -> T {
    char buf[sizeof(T)];
    file.get(buf, id, sizeof(T));
    ManagedObject &result = *reinterpret_cast<ManagedObject *>(buf);
    result.file_ = &file;
    result.id_ = id;
    return *reinterpret_cast<T *>(buf);
  }

  auto save () -> void {
    if (id_ != -1) throw Exception("Already saved");
    id_ = file_->push(reinterpret_cast<char *>(this), sizeof(T));
  }
  auto update () -> void {
    if (id_ == -1) throw Exception("Not saved");
    file_->set(reinterpret_cast<char *>(this), id_, sizeof(T));
  }
  auto destroy () -> void {
    if (id_ == -1) throw Exception("Not saved");
    file_->remove(id_);
    id_ = -1;
  }
 private:
  File_ *file_;
  size_t id_ = -1;
  ManagedObject (File_ &file, size_t id) : file_(&file), id_(id) {}
};

} // namespace ticket::file

#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, DataFile >

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

  Specialization of Index on Varchar.

Namespaces

- namespace ticket
- namespace ticket::file

  File utilities.

7.15 index.h

Go to the documentation of this file.

```
#ifndef TICKET_LIB_FILE_INDEX_H_
#define TICKET_LIB_FILE_INDEX_H_

#include "file/bptree.h"
#include "file/varchar.h"
#include "optional.h"
#include "vector.h"

namespace ticket::file {

template <typename Key, typename Model, typename DataFile>
class Index {
 public:
  Index (Key Model::*ptr, const char *filename, DataFile &datafile)
    : ptr_(ptr), tree_(filename), file_(datafile) {}
  auto insert (const Model &model) -> void {
    tree_.insert(model.*ptr_, model.id());
  }
  auto remove (const Model &model) -> void {
    tree_.remove(model.*ptr_, model.id());
  }
  auto findOne (const Key &key) -> Optional<Model> {
    auto id = tree_.findOne(key);
    if (!id) return unit;
    return Model::get(file_, *id);
  }
  auto findOneId (const Key &key) -> Optional<int> {
    return tree_.findOne(key);
  }
  auto findMany (const Key &key) -> Vector<Model> {
    Vector<Model> res;
    auto ids = tree_.findMany(key);
    if (ids.size() > 0) res.reserve(ids.size());
    for (auto id : ids) {
      res.push_back(Model::get(file_, id));
    }
    return res;
  }
  auto findManyId (const Key &key) -> Vector<int> {
    return tree_.findMany(key);
  }
 private:
  Key Model::*ptr_;
  BpTree<Key, int> tree_;
  DataFile &file_;
};

template <size_t maxLength, typename Model, typename DataFile>
class Index<Varchar<maxLength>, Model, DataFile> {
 private:
  using Key = Varchar<maxLength>;
 public:
  Index (Key Model::*ptr, const char *filename, DataFile &datafile)
    : ptr_(ptr), tree_(filename), file_(datafile) {}
  auto insert (const Model &model) -> void {
    tree_.insert(model.*ptr_.hash(), model.id());
  }
  auto remove (const Model &model) -> void {
    tree_.remove(model.*ptr_.hash(), model.id());
  }
  auto findOne (const Key &key) -> Optional<Model> {
    auto id = tree_.findOne(key.hash());
    if (!id) return unit;
    return Model::get(file_, *id);
  }
  auto findOneId (const Key &key) -> Optional<int> {
    return tree_.findOne(key.hash());
  }
  auto findMany (const Key &key) -> Vector<Model> {
    Vector<Model> res;
    auto ids = tree_.findMany(key.hash());
    if (ids.size() > 0) res.reserve(ids.size());
    for (auto id : ids) {
      res.push_back(Model::get(file_, id));
    }
    return res;
  }
  auto findManyId (const Key &key) -> Vector<int> {
    return tree_.findMany(key.hash());
  }
 private:
  Key Model::*ptr_;
  BpTree<size_t, int> tree_;
  DataFile &file_;
};

} // namespace ticket::file

#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.

```
#ifndef TICKET_LIB_FILE_SET_H_
#define TICKET_LIB_FILE_SET_H_

#include <cstring>

#include "algorithm.h"
#include "exception.h"
#include "utility.h"

// FIXME: remove dupe code of Set and Array. does C++ support mixins?
namespace ticket::file {

template <typename T, size_t maxLength, typename Cmp = Less<»
struct Set {
 private:
  auto boundsCheck_ (size_t index) -> void {
    if (index >= length) {
      throw OutOfBounds("Set: overflow or underflow");
    }
  }
  Cmp cmp_;
 public:
  Set () = default;
  size_t length = 0;
  T content[maxLength];
  auto indexOfInsert (const T &element) -> size_t {
    return lowerBound(content, content + length, element) - content;
  }
  auto indexOf (const T &element) -> size_t {
    size_t index = indexOfInsert(element);
    if (index >= length || !cmp_.equals(content[index], element)) {
      throw NotFound("Set::indexOf: element not found");
    }
    return index;
  }
  auto includes (const T &element) -> bool {
    size_t ix = indexOfInsert(element);
    return ix < length && cmp_.equals(content[ix], element);
  }
  auto insert (const T &element) -> void {
    if (length == maxLength) {
      throw Overflow("Set::insert: overflow");
    }
    size_t offset = indexOfInsert(element);
    if (offset != length) {
      memmove(
        &content[offset + 1],
        &content[offset],
        (length - offset) * sizeof(content[0])
      );
    }
    content[offset] = element;
    ++length;
  }

  auto remove (const T &element) -> void {
    removeAt(indexOf(element));
  }
  auto removeAt (size_t offset) -> void {
    boundsCheck_(offset);
    if (offset != length - 1) {
      memmove(
        &content[offset],
        &content[offset + 1],
        (length - offset - 1) * sizeof(content[0])
      );
    }
    --length;
  }
  auto clear () -> void { length = 0; }

  void copyFrom (const Set &other, size_t ixFrom, size_t ixTo, size_t count) {
    if (this == &other) {
      memmove(
        &content[ixTo],
        &content[ixFrom],
        count * sizeof(content[0])
      );
    } else {
      memcpy(
        &content[ixTo],
        &other.content[ixFrom],
        count * sizeof(content[0])
      );
    }
  }

  auto operator[] (size_t index) -> T & {
    boundsCheck_(index);
    return content[index];
  }
  auto operator[] (size_t index) const -> const T & {
    boundsCheck_(index);
    return content[index];
  }

  auto pop () -> T {
    if (length == 0) throw Underflow("Set::pop: underflow");
    return content[--length];
  }
  auto shift () -> T {
    if (length == 0) throw Underflow("Set::pop: underflow");
    T result = content[0];
    removeAt(0);
    return result;
  }

  template <typename Functor>
  auto forEach (const Functor &callback) -> void {
    for (int i = 0; i < length; ++i) callback(content[i]);
  }
};

} // namespace ticket::file

#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.

```
#ifndef TICKET_LIB_FILE_VARCHAR_H_
#define TICKET_LIB_FILE_VARCHAR_H_

#include <cstring>
#include <iostream>

#include "exception.h"

namespace ticket::file {

template <int maxLength>
struct Varchar {
 public:
  Varchar () { content[0] = '\0'; }
  Varchar (const std::string &s) {
    if (s.length() > maxLength) {
      throw Overflow("Varchar length overflow");
    }
    strncpy(content, s.c_str(), maxLength + 1);
  }
  Varchar (const char *cstr) : Varchar(std::string(cstr)) {
    if (strlen(cstr) > maxLength) {
      throw Overflow("Varchar length overflow");
    }
    strncpy(content, cstr, maxLength + 1);
  }

  template<int A>
  Varchar (const Varchar<A> &that) { *this = that; }
  operator std::string () const {
    return std::string(content);
  }
  [[nodiscard]] auto str () const -> std::string {
    return std::string(*this);
  }

  auto length () -> int {
    return strlen(content);
  }

  template <int A>
  auto operator= (const Varchar<A> &that) -> Varchar & {
    if (that.length() > maxLength) {
      throw Overflow("Varchar length overflow");
    }
    strcpy(content, that.content);
    hash_ = that.hash_;
    return *this;
  }

  template <int A>
  auto operator< (const Varchar<A> &that) const -> bool {
    return hash() < that.hash();
  }
  template <int A>
  auto operator== (const Varchar<A> &that) const -> bool {
    return hash() == that.hash();
  }
  template <int A>
  auto operator!= (const Varchar<A> &that) const -> bool {
    return hash() != that.hash();
  }

  auto hash () const -> size_t {
    if (hash_ != 0) return hash_;
    return hash_ = std::hash<std::string_view>()(content);
  }

 private:
  template <int A>
  friend class Varchar;
  char content[maxLength + 1];
  mutable size_t hash_ = 0;
};

} // namespace ticket::file

#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.

```
#ifndef TICKET_LIB_HASHMAP_H_
#define TICKET_LIB_HASHMAP_H_

// only for std::equal_to<T> and std::hash<T>
#include <functional>
#include <cstddef>

#include "exception.h"
#include "utility.h"

#ifdef DEBUG
#include <iostream>
#endif

namespace ticket {

#include "internal/rehash.inc"

template <
  typename Key,
  typename Value,
  typename Hash = std::hash<Key>,
  typename Equal = std::equal_to<Key>
> class HashMap {
 private:
  struct ListNode;
  struct Node;
 public:
  using value_type = Pair<const Key, Value>;

  class const_iterator;
  class iterator {
   public:
    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;

    iterator () = default;
    iterator (ListNode *node, HashMap *home) : node_(node), home_(home) {}
    auto operator++ (int) -> iterator {
      if (node_ == &home_->pivot_) throw Exception("invalid state");
      auto node = node_;
      node_ = node_->next_;
      return { node, home_ };
    }
    auto operator++ () -> iterator & {
      if (node_ == &home_->pivot_) throw Exception("invalid state");
      node_ = node_->next_;
      return *this;
    }
    auto operator-- (int) -> iterator {
      if (node_ == home_->pivot_.next_) throw Exception("invalid state");
      auto node = node_;
      node_ = node_->prev_;
      return { node, home_ };
    }
    auto operator-- () -> iterator & {
      if (node_ == home_->pivot_.next_) throw Exception("invalid state");
      node_ = node_->prev_;
      return *this;
    }
    auto operator* () const -> reference {
      return node_->self->value;
    }
    auto operator== (const iterator &rhs) const -> bool {
      return node_ == rhs.node_;
    }
    auto operator== (const const_iterator &rhs) const -> bool {
      return node_ == rhs.node_;
    }
    auto operator!= (const iterator &rhs) const -> bool {
      return !(*this == rhs);
    }
    auto operator!= (const const_iterator &rhs) const -> bool {
      return !(*this == rhs);
    }
    auto operator-> () const noexcept -> pointer {
      return &**this;
    }
   private:
    ListNode *node_;
    HashMap *home_;
    friend class const_iterator;
    friend class HashMap;
  };

   class const_iterator {
    public:
     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;

     const_iterator () = default;
     const_iterator (const ListNode *node, const HashMap *home) : node_(node), home_(home) {}
     const_iterator (const iterator &other) : node_(other.node_), home_(other.home_) {}
     auto operator++ (int) -> const_iterator {
       if (node_ == &home_->pivot_) throw Exception("invalid state");
       auto node = node_;
       node_ = node_->next_;
       return { node, home_ };
     }
     auto operator++ () -> const_iterator & {
       if (node_ == &home_->pivot_) throw Exception("invalid state");
       node_ = node_->next_;
       return *this;
     }
     auto operator-- (int) -> const_iterator {
       if (node_ == home_->pivot_.next_) throw Exception("invalid state");
       auto node = node_;
       node_ = node_->prev_;
       return { node, home_ };
     }
     auto operator-- () -> const_iterator & {
       if (node_ == home_->pivot_.next_) throw Exception("invalid state");
       node_ = node_->prev_;
       return *this;
     }
     auto operator* () const -> reference {
       return node_->self->value;
     }
     auto operator== (const iterator &rhs) const -> bool {
       return node_ == rhs.node_;
     }
     auto operator== (const const_iterator &rhs) const -> bool {
       return node_ == rhs.node_;
     }
     auto operator!= (const iterator &rhs) const -> bool {
       return !(*this == rhs);
     }
     auto operator!= (const const_iterator &rhs) const -> bool {
       return !(*this == rhs);
     }
     auto operator-> () const noexcept -> pointer {
       return &**this;
     }
    private:
     const ListNode *node_;
     const HashMap *home_;
     friend class iterator;
     friend class HashMap;
   };

   HashMap () = default;
   HashMap (const HashMap &other) { *this = other; }
```

```
auto operator= (const HashMap &other) -> HashMap & {
  if (this == &other) return *this;
  clear();
  capacity_ = other.capacity_;
  size_ = other.size_;
  store_ = new ListNode[internal::pow2[capacity_]];
  const ListNode *node = &other.pivot_;
  for (int i = 0; i < size_; ++i) {
    node = node->next_;
    Node *newNode = new Node(*(node->self));
    int ix = newNode->hash & internal::mask[capacity_];
    newNode->hashList.insertBefore(&store_[ix]);
    newNode->iteratorList.insertBefore(&pivot_);
  }
  return *this;
}
~HashMap () {
  destroy_();
}

auto at (const Key &key) -> Value & {
  auto it = find(key);
  if (it == end()) throw OutOfBounds();
  return it->second;
}
auto at (const Key &key) const -> const Value & {
  return const_cast<HashMap *>(this)->at(key);
}

auto operator[] (const Key &key) -> Value & {
  return insert({ key, Value() }).first->second;
}

auto operator[] (const Key &key) const -> const Value & { return at(key); }

auto begin () -> iterator { return { pivot_.next_, this }; }
auto cbegin () const -> const_iterator { return { pivot_.next_, this }; }

auto end () -> iterator { return { &pivot_, this }; }
auto cend () const -> const_iterator { return { &pivot_, this }; }

auto empty () const -> bool {
  return size_ == 0;
}
auto size () const -> size_t {
  return size_;
}

auto clear () -> void {
  destroy_();
}

auto insert (const value_type &value) -> Pair<iterator, bool> {
  auto &[ k, _ ] = value;
  auto hash = hash_(k);
  if (capacity_ > 0) {
    int ix = hash & internal::mask[capacity_];
    if (store_[ix].next() != nullptr) {
      Node *node = store_[ix].next()->find(k);
      if (node != nullptr) return { { &node->iteratorList, this }, false };
    }
  }
  growIfNeeded_();
  int ix = hash & internal::mask[capacity_];
  Node *node = new Node(value, hash);
  node->hashList.insertBefore(&store_[ix]);
  node->iteratorList.insertBefore(&pivot_);
  ++size_;
  return { { &node->iteratorList, this }, true };
}

auto erase (iterator pos) -> void {
  if (pos == end() || pos.home_ != this) throw Exception("invalid state");
  pos.node_->self->hashList.remove();
  pos.node_->self->iteratorList.remove();
  delete pos.node_->self;
  pos.node_ = &pivot_;
  --size_;
}

auto count (const Key &key) const -> size_t {
  return find(key) == cend() ? 0 : 1;
}

auto find (const Key &key) -> iterator {
  if (empty()) return end();
  auto ix = hash_(key) & internal::mask[capacity_];
```

```
    if (store_[ix].next() == nullptr) return end();
    Node *node = store_[ix].next()->find(key);
    if (node == nullptr) return end();
    return { &node->iteratorList, this };
  }
  auto find (const Key &key) const -> const_iterator {
    return const_cast<HashMap *>(this)->find(key);
  }

 private:
  struct ListNode {
    ListNode *prev_ = this;
    ListNode *next_ = this;
    auto next () -> Node * { return next_->self; }
    auto prev () -> Node * { return prev_->self; }
    Node *self = nullptr;
    ListNode () = default;
    ListNode (Node *node) : self(node) {}

    auto insertBefore (ListNode *pivot) -> void {
      prev_ = pivot->prev_;
      next_ = pivot;
      pivot->prev_ = prev_->next_ = this;
    }
    auto remove () -> void {
      prev_->next_ = next_;
      next_->prev_ = prev_;
    }
    auto init () -> void {
      prev_ = next_ = this;
    }
  };

  struct Node {
    value_type value;
    unsigned hash;
    ListNode iteratorList = this, hashList = this;
    Node () = default;
    Node (const Node &node) : value(node.value), hash(node.hash) {}
    Node (const value_type &value, unsigned hash) : value(value), hash(hash) {}
    auto find (const Key &key) -> Node * {
      if (Equal()(key, value.first)) return this;
      if (hashList.next() == nullptr) return nullptr;
      return hashList.next()->find(key);
    }
  };
  ListNode pivot_;
  ListNode *store_ = nullptr;
  int size_ = 0;
  int capacity_ = 0;
  constexpr static int kThreshold_ = 2;
  Hash hash0_;
  auto hash_ (const Key &key) const -> unsigned {
    return internal::rehash(hash0_(key));
  }
  auto growIfNeeded_ () -> void {
    auto capacityNeeded = static_cast<unsigned long long>((size_ + 1) * kThreshold_);
    if (capacityNeeded > internal::pow2[capacity_]) grow_();
  }
  auto grow_ () -> void {
    if (capacity_ == 0) {
      capacity_ = 2;
      store_ = new ListNode[4];
      return;
    }
    int newCapacity = capacity_ + 1;
    auto prospective = new ListNode[internal::pow2[newCapacity]];
    auto node = &pivot_;
    for (int i = 0; i < size_; ++i) {
      node = node->next_;
      int ix = node->self->hash & internal::mask[newCapacity];
      node->self->hashList.insertBefore(&prospective[ix]);
    }
    capacity_ = newCapacity;
    delete[] store_;
    store_ = prospective;
  }

  auto destroy_ () -> void {
    ListNode *node = pivot_.next_;
    for (int i = 0; i < size_; ++i) {
      ListNode *next = node->next_;
      delete node->self;
      node = next;
    }
    capacity_ = 0;
    size_ = 0;
    delete[] store_;
    store_ = nullptr;
    pivot_.init();
  }
};

} // namespace ticket

#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.

```
#ifndef TICKET_LIB_MAP_H_
#define TICKET_LIB_MAP_H_

#include <cstddef>

#include "internal/tree.h"
#include "utility.h"
#include "exception.h"

#ifdef DEBUG
#include <iostream>
#endif

namespace ticket {

#include "internal/map-value-compare.inc"

template <typename KeyType, typename ValueType, typename Compare = internal::LessOp>
class Map {
 public:
  using value_type = Pair<const KeyType, ValueType>;
 private:
  using TreeType = typename internal::RbTree<value_type, internal::MapValueCompare<KeyType, ValueType,
    Compare»;
 public:
  using iterator = typename TreeType::iterator;
  using const_iterator = typename TreeType::const_iterator;

  Map () = default;
  auto at (const KeyType &key) -> ValueType & {
    auto it = tree_.find(key);
    if (it == tree_.end()) throw OutOfBounds();
    return it->second;
  }
  auto at (const KeyType &key) const -> const ValueType & {
    auto it = tree_.find(key);
    if (it == tree_.cend()) throw OutOfBounds();
    return it->second;
  }
  auto operator[] (const KeyType &key) -> ValueType & {
    // we need to use the default constructor here. Too bad we have no choice.
    auto p = tree_.insert({ key, ValueType() });
    return p.first->second;
  }
  auto operator[] (const KeyType &key) const -> const ValueType & {
    return at(key);
  }
  auto begin () -> iterator {
    return tree_.begin();
  }
  auto cbegin () const -> const_iterator {
    return tree_.cbegin();
  }
  auto end () -> iterator {
    return tree_.end();
  }
  auto cend () const -> const_iterator {
    return tree_.cend();
  }
  auto empty () const -> bool {
    return tree_.empty();
  }
  auto size () const -> size_t {
    return tree_.size();
  }
  auto clear () -> void {
    tree_.clear();
  }
  auto insert (const value_type &value) -> Pair<iterator, bool> {
    return tree_.insert(value);
  }
  auto erase (iterator pos) -> void {
    return tree_.erase(pos);
  }
  auto count (const KeyType &key) const -> size_t {
    auto it = tree_.find(key);
    return it == tree_.cend() ? 0 : 1;
  }
  auto find (const KeyType &key) -> iterator {
    return tree_.find(key);
  }
  auto find (const KeyType &key) const -> const_iterator {
    return tree_.find(key);
  }

#ifdef DEBUG
  auto print () -> void {
    std::cout « "s=" « size() « " ";
    for (const auto &p : *this) {
      std::cout « "(" « p.first.print() « ", " « p.second.print() « ") ";
    }
    std::cout « std::endl;
  }
#endif

 private:
  TreeType tree_;
};

} // namespace ticket

#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.

```

#ifndef TICKET_LIB_OPTIONAL_H_
#define TICKET_LIB_OPTIONAL_H_

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

namespace ticket {

template <typename T>
class Optional : Variant<Unit, T> {
 private:
  using VarT = Variant<Unit, T>;
 public:
  Optional () = default;
  Optional (Unit /* unused */) : VarT(unit) {}
  Optional (const T &value) : VarT(value) {}
  auto operator= (const T &value) -> Optional & {
    VarT::operator=(value);
    return *this;
  }
  operator bool () const {
    return this->template is<T>();
  }
  auto operator* () -> T & {
    return *this->template get<T>();
  }
  auto operator* () const -> const T & {
    return *this->template get<T>();
  }
  auto operator-> () -> T * {
    return this->template get<T>();
  }
  auto operator-> () const -> const T * {
    return this->template get<T>();
  }
};

} // namespace ticket

#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.

```
#ifndef TICKET_LIB_RESULT_H_
#define TICKET_LIB_RESULT_H_

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

namespace ticket {

template <typename ResultType, typename ErrorType>
class Result : public Variant<ResultType, ErrorType> {
 public:
  Result () = delete;
  template <typename T>
  Result (const T &value) : Variant<ResultType, ErrorType>(value) {}
  auto result () -> ResultType & {
    return *this->template get<ResultType>();
  }
  auto result () const -> const ResultType & {
    return *this->template get<ResultType>();
  }
  auto error () -> ErrorType * {
    return this->template get<ErrorType>();
  }
  auto error () const -> const ErrorType * {
    return this->template get<ErrorType>();
  }

  auto success () const -> bool {
    return this->index() == 0;
  }
};

} // namespace ticket

#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::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.

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.

```
// This file defines several common utilities.
#ifndef TICKET_LIB_UTILITY_H_
#define TICKET_LIB_UTILITY_H_

#include <iostream>

#include "vector.h"

#ifdef TICKET_DEBUG
#include <cassert>
#define TICKET_ASSERT(x) assert(x)
#else
#define TICKET_ASSERT(x)
#endif // TICKET_DEBUG

namespace ticket {

auto split (std::string &str, char sep)
  -> Vector<std::string_view>;

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

struct Unit {
  constexpr Unit () = default;
  template <typename T>
  constexpr Unit (const T & /* unused */) {}
  auto operator< (const Unit & /* unused */) -> bool {
    return false;
  }
};
inline constexpr Unit unit;

template <typename T>
auto declval () -> T;

template <typename T>
auto move (T &val) -> T && {
  return reinterpret_cast<T &&>(val);
}

template <typename T1, typename T2>
class Pair {
 public:
  T1 first;
  T2 second;
  constexpr Pair () : first(), second() {}
  Pair (const Pair &other) = default;
  Pair (Pair &&other) noexcept = default;
  Pair (const T1 &x, const T2 &y) : first(x), second(y) {}
  template <class U1, class U2>
  Pair (U1 &&x, U2 &&y) : first(x), second(y) {}
  template <class U1, class U2>
  Pair (const Pair<U1, U2> &other) : first(other.first), second(other.second) {}
  template <class U1, class U2>
  Pair (Pair<U1, U2> &&other) : first(other.first), second(other.second) {}
};

template <typename Lt>
class Cmp {
 public:
  template <typename T, typename U>
  auto equals (const T &lhs, const U &rhs) -> bool {
    return !lt_(lhs, rhs) && !lt_(rhs, lhs);
  }
  template <typename T, typename U>
  auto ne (const T &lhs, const U &rhs) -> bool {
    return !equals(lhs, rhs);
  }
  template <typename T, typename U>
  auto lt (const T &lhs, const U &rhs) -> bool {
    return lt_(lhs, rhs);
  }
  template <typename T, typename U>
  auto gt (const T &lhs, const U &rhs) -> bool {
    return lt_(rhs, lhs);
  }
  template <typename T, typename U>
  auto leq (const T &lhs, const U &rhs) -> bool {
    return !gt(lhs, rhs);
  }
  template <typename T, typename U>
  auto geq (const T &lhs, const U &rhs) -> bool {
    return !lt(lhs, rhs);
  }
 private:
  Lt lt_;
};

#include "internal/cmp.inc"

template <typename Lt = internal::LessOp>
using Less = Cmp<Lt>;
template <typename Lt = internal::LessOp>
using Greater = Cmp<internal::GreaterOp<Lt>>;

} // namespace ticket

#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.

```
#ifndef TICKET_LIB_VARIANT_H_
#define TICKET_LIB_VARIANT_H_

#include "internal/variant-impl.h"
#include "utility.h"

namespace ticket {

template <typename ...Ts>
class Variant {
 private:
  using Traits = internal::VariantTraits<Ts...>;
  using First = typename Traits::template NthType<0>;
  using Second = typename Traits::template NthType<1>;
  static constexpr size_t length = sizeof...(Ts);
  static_assert(length >= 2);
  static_assert(!Traits::hasDuplicates());
 public:
  Variant () : ix_(0), store_(internal::ctorIndex<0>) {}
  template <typename T, int ix = Traits::template indexOf<T>()>
  Variant (const T &value) :
    ix_(ix),
    store_(internal::ctorIndex<ix>, value) {
    static_assert(Traits::template includes<T>());
  }
  Variant (const Variant &other) {
    *this = other;
  }
  Variant (Variant &&other) noexcept { *this = move(other); }
  // this class may be extended, so let it be virtual.
  virtual ~Variant () {
    destroy_();
  }
  auto operator= (const Variant &other) -> Variant & {
    if (this == &other) return *this;
    destroy_();
    ix_ = other.ix_;
    if constexpr (length == 2) {
      if (ix_ == 0) new(&get_<First>()) First(other.get_<First>());
      else new(&get_<Second>()) Second(other.get_<Second>());
    } else {
      other.visit([this] (auto &value) {
        using T = std::remove_cvref_t<decltype(value)>;
        new(&get_<T>()) T(value);
      });
    }
    return *this;
  }
  auto operator= (Variant &&other) noexcept -> Variant & {
    if (this == &other) return *this;
    destroy_();
    ix_ = other.ix_;
    if constexpr (length == 2) {
      if (ix_ == 0) new(&get_<First>()) First(move(other.get_<First>()));
      else new(&get_<Second>()) Second(move(other.get_<Second>()));
    } else {
      other.visit([this] (auto &value) {
        using T = decltype(value);
        new(&get_<T>()) T(move(value));
      });
    }
    return *this;
  }

  template <typename T, int ix = Traits::template indexOf<T>()>
  auto operator= (const T &value) -> Variant & {
    static_assert(Traits::template includes<T>());
    destroy_();
    ix_ = ix;
    new(&get_<T>()) T(value);
    return *this;
  }

  template <typename T>
  auto is () const -> bool {
    static_assert(Traits::template includes<T>());
    return ix_ == Traits::template indexOf<T>();
  }
  auto index () const -> int {
    return ix_;
  }

  template <typename T>
  auto get () -> T * {
    if (is<T>()) return &get_<T>();
    return nullptr;
  }
  template <typename T>
  auto get () const -> const T * {
    if (is<T>()) return &get_<T>();
    return nullptr;
  }
  template <int ix>
  auto get () -> typename Traits::template NthType<ix> * {
    if (ix_ != ix) return nullptr;
    return &get_<typename Traits::template NthType<ix»();
  }
  template <int ix>
  auto get () const -> const typename Traits::template NthType<ix> * {
    if (ix_ != ix) return nullptr;
    return &get_<typename Traits::template NthType<ix»();
  }

  template <typename Visitor>
  auto visit (const Visitor &f) const -> void {
    using Vt = typename Traits::template Vtable<Visitor>;
    // sorry about the C-style cast here... it casts away const.
    Vt::visit(ix_, f, (void *) &store_);
  }

private:
  int ix_ = -1;
  typename Traits::Impl store_{internal::ctorValueless};

  template <typename T = void>
  auto get_ () -> T & {
    return *reinterpret_cast<T *>(&store_);
  }
  template <typename T = void>
  auto get_ () const -> const T & {
    return *reinterpret_cast<const T *>(&store_);
  }

  auto destroy_ () -> void {
    if (ix_ == -1) return;
    if constexpr (length == 2) {
      if (ix_ == 0) get_<First>().~First();
      else get_<Second>().~Second();
    } else {
      visit([] (auto &value) {
        using T = std::remove_reference_t<decltype(value)>;
        value.~T();
      });
    }
    ix_ = -1;
  }
};

} // namespace ticket

#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.

```
#ifndef TICKET_LIB_VECTOR_H_
#define TICKET_LIB_VECTOR_H_

#include <climits>
#include <cstddef>
#include <iterator>

#include "exception.h"

namespace ticket {

template<typename T>
class Vector {
 public:
  class const_iterator;
  class iterator {
   public:
    using difference_type = std::ptrdiff_t;
    using value_type = T;
    using pointer = T *;
    using reference = T &;
    using iterator_category = std::output_iterator_tag;

   private:
    Vector *home_;
    pointer ptr_;
    iterator (Vector *home, pointer ptr) : home_(home), ptr_(ptr) {}
   public:
    auto operator+ (const int &n) const -> iterator {
      return iterator(home_, ptr_ + n);
    }
    auto operator- (const int &n) const -> iterator {
      return iterator(home_, ptr_ - n);
    }
    // return the distance between two iterators,
    // if these two iterators point to different vectors, throw invaild_iterator.
    auto operator- (const iterator &rhs) const -> int {
      if (home_ != rhs.home_) throw Exception("invalid operation");
      return ptr_ - rhs.ptr_;
    }
    auto operator+= (const int &n) -> iterator & {
      ptr_ += n;
      return *this;
    }
    auto operator-= (const int &n) -> iterator & { return (*this += -n); }
    auto operator++ (int) const -> iterator { return operator+(1); }
    auto operator++ () -> iterator & { return (*this += 1); }
    auto operator-- (int) const -> iterator { return operator+(-1); }
    auto operator-- () -> iterator & { return (*this -= 1); }
    auto operator* () const -> T & { return *ptr_; }
    auto operator== (const iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
    auto operator== (const const_iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
    auto operator!= (const iterator &rhs) const -> bool { return !(*this == rhs); }
    auto operator!= (const const_iterator &rhs) const -> bool { return !(*this == rhs); }
    auto operator< (const iterator &rhs) const -> bool {
      return **this < *rhs;
    }
    auto operator< (const const_iterator &rhs) const -> bool {
      return **this < *rhs;
    }
    friend class const_iterator;
    friend class Vector;
  };
  class const_iterator {
   public:
    using difference_type = std::ptrdiff_t;
    using value_type = T;
    using pointer = T *;
    using reference = T &;
    using iterator_category = std::output_iterator_tag;

   private:
    const Vector *home_;
    const T *ptr_;
    const_iterator (const Vector *home, pointer ptr) : home_(home), ptr_(ptr) {}
   public:
    auto operator+ (const int &n) const -> const_iterator {
      return const_iterator(home_, ptr_ + n);
    }
    auto operator- (const int &n) const -> const_iterator {
      return const_iterator(home_, ptr_ - n);
    }
    auto operator- (const const_iterator &rhs) const -> int {
      if (home_ != rhs.home_) throw Exception("invalid operation");
      return ptr_ - rhs.ptr_;
    }
    auto operator+= (const int &n) -> const_iterator & {
      ptr_ += n;
       return *this;
     }
     auto operator-= (const int &n) -> const_iterator & { return (*this += -n); }
     auto operator++ (int) const -> const_iterator { return operator+(1); }
     auto operator++ () -> const_iterator & { return (*this += 1); }
     auto operator-- (int) const -> const_iterator { return operator+(-1); }
     auto operator-- () -> const_iterator & { return (*this -= 1); }
     auto operator* () const -> const T & { return *ptr_; }
     auto operator== (const iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
     auto operator== (const const_iterator &rhs) const -> bool { return ptr_ == rhs.ptr_; }
     auto operator!= (const iterator &rhs) const -> bool { return !(*this == rhs); }
     auto operator!= (const const_iterator &rhs) const -> bool { return !(*this == rhs); }
     auto operator< (const iterator &rhs) const -> bool {
       return **this < *rhs;
     }
     auto operator< (const const_iterator &rhs) const -> bool {
       return **this < *rhs;
     }
     friend class iterator;
     friend class Vector;
   };
```

```
Vector () = default;
Vector (const Vector &other) { *this = other; }
Vector (Vector &&other) noexcept { *this = move(other); }
~Vector () {
  destroyContents_();
  delete[] reinterpret_cast<char *>(storage_);
}
auto operator= (const Vector &other) -> Vector & {
  if (this == &other) return *this;
  clear();
  grow_(other.capacity_);
  size_ = other.size_;
  copyContents_(storage_, other.storage_, size_);
  return *this;
}
auto operator= (Vector &&other) noexcept -> Vector & {
  if (this == &other) return *this;
  clear();
  storage_ = other.storage_;
  size_ = other.size_;
  capacity_ = other.capacity_;
  other.size_ = other.capacity_ = 0;
  other.storage_ = nullptr;
  return *this;
}

auto at (const size_t &pos) -> T & {
  checkPosition_(pos);
  return storage_[pos];
}
auto at (const size_t &pos) const -> const T & {
  return const_cast<Vector *>(this)->at(pos);
}
auto operator[] (const size_t &pos) -> T & { return at(pos); }
auto operator[] (const size_t &pos) const -> const T & { return at(pos); }
auto front () const -> const T & {
  checkNonEmpty_();
  return at(0);
}
auto back () const -> const T & {
  checkNonEmpty_();
  return at(size_ - 1);
}
auto begin () -> iterator {
  return iterator(this, storage_);
}
auto begin () const -> const_iterator { return cbegin(); }
auto cbegin () const -> const_iterator {
  return const_iterator(this, storage_);
}
auto end () -> iterator {
  return iterator(this, storage_ + size_);
}
auto end () const -> const_iterator { return cend(); }
auto cend () const -> const_iterator {
  return const_iterator(this, storage_ + size_);
}
auto empty () const -> bool {
  return size_ == 0;
}
auto size () const -> size_t {
  return size_;
}
auto clear () -> void {
  destroyContents_();
  delete[] reinterpret_cast<char *>(storage_);
  storage_ = nullptr;
  capacity_ = 0;
  size_ = 0;
}
auto insert (iterator pos, const T &value) -> iterator { return insert(pos.ptr_ - storage_, value); }
auto insert (const size_t &ix, const T &value) -> iterator {
  if (ix > size_) throw OutOfBounds();
  if (size_ == capacity_) grow_();
  for (size_t i = size_; i > ix; --i) {
    storage_[i] = move_(storage_[i - 1]);
  }
  storage_[ix] = value;
  ++size_;
  return iterator(this, storage_ + ix);
}
auto erase (iterator pos) -> iterator { return erase(pos.ptr_ - storage_); }
auto erase (const size_t &ix) -> iterator {
  checkPosition_(ix);
  for (size_t i = ix; i + 1 < size_; ++i) {
    storage_[i] = move_(storage_[i + 1]);
  }
```

```
    (storage_ + size_ - 1)->~T();
    --size_;
    return iterator(this, storage_ + ix);
  }
  auto push_back (const T &value) -> void {
    if (size_ == capacity_) grow_();
    new(storage_ + size_) T(value);
    ++size_;
  }
  auto pop_back () -> void {
    checkNonEmpty_();
    (storage_ + size_ - 1)->~T();
    --size_;
  }

  auto reserve (size_t capacity) -> void {
    if (capacity_ < capacity) grow_(capacity);
  }

 private:
  static constexpr size_t kSzDefault_ = 4;
  static constexpr size_t kSzT_ = sizeof(T);
  T *storage_ = nullptr;
  size_t capacity_ = 0;
  size_t size_ = 0;

  static auto move_ (T &el) -> T && { return reinterpret_cast<T &&>(el); }
  static auto copyContents_ (T *to, T *from, size_t n) -> void {
    for (size_t i = 0; i < n; ++i) {
      to[i] = from[i];
    }
  }
  static auto moveContents_ (T *to, T *from, size_t n) -> void {
    for (size_t i = 0; i < n; ++i) {
      new(to + i) T(move_(from[i]));
      from[i].~T();
    }
  }
  static auto destroyContents_ (T *array, size_t n) -> void {
    for (size_t i = 0; i < n; ++i) {
      (array + i)->~T();
    }
  }
  auto destroyContents_ () -> void { destroyContents_(storage_, size_); }
  auto grow_ (size_t capNew) -> void {
    T *storeNew = reinterpret_cast<T *>(new char[capNew * kSzT_]);
    if (storage_ != nullptr) {
      moveContents_(storeNew, storage_, size_);
      delete[] reinterpret_cast<char *>(storage_);
    }
    storage_ = storeNew;
    capacity_ = capNew;
  }
  auto grow_ () -> void {
    grow_(storage_ == nullptr ? kSzDefault_ : 2 * capacity_);
  }
  auto checkPosition_ (size_t pos) const -> void {
    // since this is size_t which is unsigned, we could not have pos < 0.
    if (pos >= size_) throw OutOfBounds();
  }
  auto checkNonEmpty_ () const -> void {
    if (size_ == 0) throw OutOfBounds();
  }
};

} // namespace ticket

#endif // TICKET_LIB_VECTOR_H_
```

7.35 src/main.cpp File Reference

```
#include <iostream>
#include "parser.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/order.cpp File Reference

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

Namespaces

- namespace ticket

Variables

- file::File ticket::orders {"orders"}
- file::Index< User::Id, Order, decltype(orders)> ticket::ixOrdersUserId {&Order::user, "orders.user.ix", orders}
- file::File ticket::pendingOrders {"pending-orders"}
- file::Index< Ride, PendingOrder, decltype(pendingOrders)> ticket::ixPendingOrdersRide

7.38 src/order.h File Reference

```
#include "file/file.h"
#include "file/index.h"
#include "train.h"
#include "user.h"
```

Classes

- struct ticket::Order
- struct ticket::PendingOrder

Namespaces

- namespace ticket

7.39 order.h

Go to the documentation of this file.

```
#ifndef TICKET_ORDER_H_
#define TICKET_ORDER_H_

#include "file/file.h"
#include "file/index.h"
#include "train.h"
#include "user.h"

namespace ticket {

struct Order : public file::ManagedObject<Order> {
  using Id = int;
  enum Status { kSuccess, kPending, kRefunded };

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

  auto getTrain () -> Train;
};

extern file::File<> orders;
extern file::Index<User::Id, Order, decltype(orders)>
  ixOrdersUserId;

struct PendingOrder : public file::ManagedObject<PendingOrder> {
  Ride ride;
  int ixFrom, ixTo;
  int seats;
  Order::Id order;

  auto satisfiable () -> bool;
  auto getOrder () -> Order;
};

extern file::File<> pendingOrders;
extern file::Index<Ride, PendingOrder, decltype(pendingOrders)>
  ixPendingOrdersRide;

} // namespace ticket

#endif // TICKET_ORDER_H_
```

7.40 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.

7.41 src/parser.h File Reference

```
#include <iostream>
#include "datetime.h"
#include "exception.h"
#include "optional.h"
#include "variant.h"
#include "result.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::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, Add↩ Train, ReleaseTrain, QueryTrain, QueryTicket, QueryTransfer, BuyTicket, QueryOrder, RefundTicket, Roll-back, 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::dispatch (const AddUser &cmd) -> void

  Visitor for the commands.
- auto ticket::command::dispatch (const Login &cmd) -> void
- auto ticket::command::dispatch (const Logout &cmd) -> void
- auto ticket::command::dispatch (const QueryProfile &cmd) -> void
- auto ticket::command::dispatch (const ModifyProfile &cmd) -> void
- auto ticket::command::dispatch (const AddTrain &cmd) -> void
- auto ticket::command::dispatch (const ReleaseTrain &cmd) -> void
- auto ticket::command::dispatch (const QueryTrain &cmd) -> void
- auto ticket::command::dispatch (const QueryTicket &cmd) -> void
- auto ticket::command::dispatch (const QueryTransfer &cmd) -> void
- auto ticket::command::dispatch (const BuyTicket &cmd) -> void
- auto ticket::command::dispatch (const QueryOrder &cmd) -> void
- auto ticket::command::dispatch (const RefundTicket &cmd) -> void
- auto ticket::command::dispatch (const Rollback &cmd) -> void
- auto ticket::command::dispatch (const Clean &cmd) -> void
- auto ticket::command::dispatch (const Exit &cmd) -> void

7.42 parser.h

Go to the documentation of this file.

```
// This file is autogenerated. Do not modify.
#ifndef TICKET_PARSER_H_
#define TICKET_PARSER_H_

#include <iostream>

#include "datetime.h"
#include "exception.h"
#include "optional.h"
#include "variant.h"
#include "result.h"

namespace ticket::command {

enum SortType { kTime, kCost };

struct AddUser {
  Optional<std::string> currentUser;
  std::string username;
  std::string password;
  std::string name;
  std::string email;
  Optional<int> privilege;
};

struct Login {
  std::string username;
  std::string password;
};

struct Logout {
  std::string username;
};

struct QueryProfile {
  std::string currentUser;
  std::string username;
};

struct ModifyProfile {
  std::string currentUser;
  std::string username;
  Optional<std::string> password;
  Optional<std::string> name;
  Optional<std::string> email;
  Optional<int> privilege;
};

struct AddTrain {
  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;
};

struct ReleaseTrain {
  std::string id;
};

struct QueryTrain {
  std::string id;
  Date date;
};

struct QueryTicket {
  std::string from;
  std::string to;
  Date date;
  SortType sort = kTime;
};

struct QueryTransfer {
  std::string from;
  std::string to;
  Date date;
  SortType sort = kTime;
};

struct BuyTicket {
  std::string currentUser;
  std::string train;
  Date date;
  int seats;
  std::string from;
  std::string to;
  bool queue = false;
};

struct QueryOrder {
  std::string currentUser;
};

struct RefundTicket {
  std::string currentUser;
  int index = 1;
};

struct Rollback {
  int timestamp;
};

struct Clean {

};

struct Exit {

};


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

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

auto dispatch (const AddUser &cmd) -> void;
auto dispatch (const Login &cmd) -> void;
auto dispatch (const Logout &cmd) -> void;
auto dispatch (const QueryProfile &cmd) -> void;
auto dispatch (const ModifyProfile &cmd) -> void;
auto dispatch (const AddTrain &cmd) -> void;
auto dispatch (const ReleaseTrain &cmd) -> void;
auto dispatch (const QueryTrain &cmd) -> void;
auto dispatch (const QueryTicket &cmd) -> void;
auto dispatch (const QueryTransfer &cmd) -> void;
auto dispatch (const BuyTicket &cmd) -> void;
auto dispatch (const QueryOrder &cmd) -> void;
auto dispatch (const RefundTicket &cmd) -> void;
auto dispatch (const Rollback &cmd) -> void;
auto dispatch (const Clean &cmd) -> void;
auto dispatch (const Exit &cmd) -> void;

} // namespace ticket::command

#endif // TICKET_PARSER_H_
```

7.43 src/rollback.cpp File Reference

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

Namespaces

- namespace ticket

Variables

- file::File ticket::logEntries {"rollback-log"}

7.44 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::ReleaseTrain
- struct ticket::rollback::BuyTicket
- struct ticket::rollback::RefundTicket
- struct ticket::rollback::LogEntry

Namespaces

- namespace ticket
- namespace ticket::rollback

Functions

- auto ticket::rollback::dispatch (const AddUser &log) -> void

  Visitor for the log entries.
- auto ticket::rollback::dispatch (const ModifyProfile &log) -> void
- auto ticket::rollback::dispatch (const AddTrain &log) -> void
- auto ticket::rollback::dispatch (const ReleaseTrain &log) -> void
- auto ticket::rollback::dispatch (const BuyTicket &log) -> void
- auto ticket::rollback::dispatch (const RefundTicket &log) -> void

Variables

- file::File ticket::rollback::logEntries

7.45 rollback.h

Go to the documentation of this file.

```
#ifndef TICKET_BACKLOG_H_
#define TICKET_BACKLOG_H_

#include "file/file.h"
#include "optional.h"
#include "order.h"
#include "train.h"
#include "user.h"
#include "variant.h"

namespace ticket::rollback {

struct AddUser {
  int id;
};

struct ModifyProfile {
  int id;
  Optional<User::Password> password;
  Optional<User::Name> name;
  Optional<User::Email> email;
  Optional<User::Privilege> privilege;
};

struct AddTrain {
  int id;
};

struct ReleaseTrain {
  int id;
};

struct BuyTicket {
  int id;
};

struct RefundTicket {
  int id;
  Order::Status status;
};

struct LogEntry : public file::ManagedObject<LogEntry> {
  int timestamp;
  Variant<
    AddUser,
    ModifyProfile,
    AddTrain,
    ReleaseTrain,
    BuyTicket,
    RefundTicket
  > content;
};

extern file::File<> logEntries;

auto dispatch (const AddUser &log) -> void;
auto dispatch (const ModifyProfile &log) -> void;
auto dispatch (const AddTrain &log) -> void;
auto dispatch (const ReleaseTrain &log) -> void;
auto dispatch (const BuyTicket &log) -> void;
auto dispatch (const RefundTicket &log) -> void;

} // namespace ticket::rollback

#endif // TICKET_BACKLOG_H_
```

7.46 src/train.cpp File Reference

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

Namespaces

- namespace ticket

Variables

- file::File ticket::trains {"trains"}
- file::Index< Train::Id, Train, decltype(trains)> ticket::ixTrainsId {&Train::trainId, "trains.train-id.ix", trains}
- file::BpTree< size_t, int > ticket::ixTrainsStop {"trains.stop.ix"}
- file::File ticket::rideSeats {"ride-seats"}
- file::Index< Ride, RideSeats, decltype(rideSeats)> ticket::ixRideSeatsRide

7.47 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::Train
- struct ticket::Train::Stop
- struct ticket::Train::Edge
- struct ticket::Ride
- struct ticket::RideSeats

Namespaces

- namespace ticket
- namespace ticket::Station

Typedefs

- using ticket::Station::Id = file::Varchar< 30 >

7.48 train.h

Go to the documentation of this file.

```
#ifndef TICKET_TRAIN_H_
#define TICKET_TRAIN_H_

#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"

namespace ticket {

namespace Station {
using Id = file::Varchar<30>;
} // namespace Station

struct RideSeats;

struct Train : public file::ManagedObject<Train> {
  using Id = file::Varchar<20>;
  using Type = char;
  struct Stop {
    Station::Id name;
  };
  struct Edge {
    int price;
    Instant departure;
    Instant arrival;
  };

  Id trainId;
  file::Array<Stop, 100> stops;
  file::Array<Edge, 99> edges;
  int seats;
  Date begin, end;
  Type type;
  bool released;

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

  auto getRide (Date date) -> RideSeats;
  auto getRide (Date date, int ixDeparture) -> RideSeats;

  auto runsOnDate (Date date) -> bool;
  auto runsOnDate (Date date, int ixDeparture) -> bool;
};

extern file::File<> trains;
extern file::Index<Train::Id, Train, decltype(trains)>
  ixTrainsId;
extern file::BpTree<size_t, int> ixTrainsStop;

struct Ride {
  int train;
  Date date;

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

struct RideSeats : public file::ManagedObject<RideSeats> {
  Ride ride;
  file::Array<int, 99> seatsRemaining;
  auto ticketsAvailable (int ixFrom, int ixTo) -> int;
};

extern file::File<> rideSeats;
extern file::Index<Ride, RideSeats, decltype(rideSeats)>
  ixRideSeatsRide;

} // namespace ticket

#endif // TICKET_TRAIN_H_
```

7.49 src/user.cpp File Reference

```
#include "user.h"
#include <iostream>
#include "hashmap.h"
#include "parser.h"
#include "rollback.h"
```

Namespaces

- namespace ticket

Variables

- file::File ticket::users {"users"}
- file::Index< User::Id, User, decltype(users)> ticket::ixUsersUsername {&User::username, "users.↩ username.ix", users}
- HashMap< std::string, Unit > ticket::usersLoggedIn

  a set of users that are logged in.

7.50 src/user.h File Reference

```
#include "file/file.h"
#include "file/index.h"
#include "file/varchar.h"
```

Classes

- struct ticket::User

Namespaces

- namespace ticket

7.51 user.h

Go to the documentation of this file.

```
#ifndef TICKET_USER_H_
#define TICKET_USER_H_

#include "file/file.h"
#include "file/index.h"
#include "file/varchar.h"

namespace ticket {

struct User : public file::ManagedObject<User> {
  using Id = file::Varchar<20>;
  using Password = file::Varchar<30>;
  using Name = file::Varchar<15>;
  using Email = file::Varchar<30>;
  using Privilege = int;

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

  static auto hasUser (const char *username) -> bool;
};

extern file::File<> users;
extern file::Index<User::Id, User, decltype(users)>
  ixUsersUsername;

} // namespace ticket

#endif // TICKET_USER_H_
```

Index