My Project

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

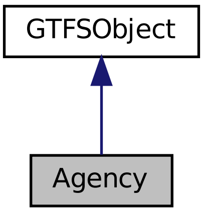
Class Documentation

4.1 Agency Class Reference

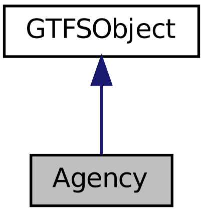
Represents a transit agency in the GTFS data.

```
#include <Agency.h>
```

Inheritance diagram for Agency:



Collaboration diagram for Agency:



Additional Inherited Members

4.1.1 Detailed Description

Represents a transit agency in the GTFS data.

This class inherits from GTFSObject and encapsulates the details of a transit agency.

Note

- This class currently acts as a placeholder and can be extended with specific attributes and methods relevant to transit agencies.

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

- src/NetworkObjects/GTFSObjects/Agency.h

4.2 Application Class Reference

The main application class for managing the RAPTOR transit algorithm.

```
#include <Application.h>
```

Public Member Functions

- Application (std::vector< std::string > inputDirectories)

  Constructs an Application instance with the given input directories.
- void run ()

  Starts the application, providing a command-line interface for users.

Private Member Functions

- void initializeRaptor ()

  Initializes the RAPTOR data structures by parsing input files.
- void handleQuery ()

  Handles a user query by executing the RAPTOR algorithm and displaying results.
- Query getQuery ()

  Retrieves a query from the user, including source, target, date, and time.
- std::string getSource ()

  Prompts the user to enter the source stop ID.
- std::string getTarget ()

  Prompts the user to enter the target stop ID.

Static Private Member Functions

- static void showCommands ()

  Displays the list of available commands to the user.
- static Date getDate ()

  Prompts the user to enter the journey date.
- static int getYear ()

  Prompts the user to enter the year.
- static int getMonth ()

  Prompts the user to enter the month.
- static int getDay (int year, int month)

  Prompts the user to enter the day.
- static Time getDepartureTime ()

  Prompts the user to enter the departure time.
- static int getHours ()

  Prompts the user to enter the hour component of the departure time.
- static int getMinutes ()

  Prompts the user to enter the minutes component of the departure time.

Private Attributes

- std::vector< std::string > inputDirectories

  Directories containing transit data files.
- std::optional< Raptor > raptor_

  Optional instance of the RAPTOR algorithm.

4.2.1 Detailed Description

The main application class for managing the RAPTOR transit algorithm.

This class provides methods to initialize data structures, handle user input, and execute the RAPTOR algorithm for transit planning.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Application()

```
Application::Application (
            std::vector< std::string > inputDirectories )  [explicit]
```

Constructs an Application instance with the given input directories.

Parameters

inputDirectories	A vector of directories containing transit data files.

4.2.3 Member Function Documentation

4.2.3.1 getDate()

```
Date Application::getDate ( )  [static], [private]
```

Prompts the user to enter the journey date.

Returns

A Date object representing the entered date.

4.2.3.2 getDay()

```
int Application::getDay (
             int year,
             int month )  [static], [private]
```

Prompts the user to enter the day.

Parameters

year	The year of the journey, used for validation.
month	The month of the journey, used for validation.

Returns

The entered day as an integer.

4.2.3.3 getDepartureTime()

```
Time Application::getDepartureTime ( )  [static], [private]
```

Prompts the user to enter the departure time.

Returns

A Time object representing the departure time.

4.2.3.4 getHours()

```
int Application::getHours ( )  [static], [private]
```

Prompts the user to enter the hour component of the departure time.

Returns

The entered hour as an integer.

4.2.3.5 getMinutes()

```
int Application::getMinutes ( )  [static], [private]
```

Prompts the user to enter the minutes component of the departure time.

Returns

The entered minutes as an integer.

4.2.3.6 getMonth()

```
int Application::getMonth ( )  [static], [private]
```

Prompts the user to enter the month.

Returns

The entered month as an integer.

4.2.3.7 getQuery()

```
Query Application::getQuery ( )  [private]
```

Retrieves a query from the user, including source, target, date, and time.

Returns

A Query object representing the user's transit request.

4.2.3.8 getSource()

```
std::string Application::getSource ( )  [private]
```

Prompts the user to enter the source stop ID.

Returns

A valid source stop ID.

4.2.3.9 getTarget()

```
std::string Application::getTarget ( )  [private]
```

Prompts the user to enter the target stop ID.

Returns

A valid target stop ID.

4.2.3.10 getYear()

```
int Application::getYear ( )  [static], [private]
```

Prompts the user to enter the year.

Returns

The entered year as an integer.

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

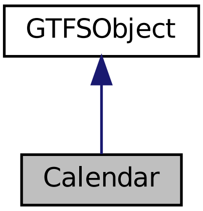
- src/Application.h
- src/Application.cpp

4.3 Calendar Class Reference

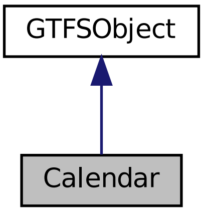
Represents active weekdays for calendar in the GTFS data.

```
#include <Calendar.h>
```

Inheritance diagram for Calendar:



Collaboration diagram for Calendar:



Additional Inherited Members

4.3.1 Detailed Description

Represents active weekdays for calendar in the GTFS data.

This class inherits from GTFSObject and encapsulates the details of active weekdays for calendar.

Note

- This class currently acts as a placeholder and can be extended with specific attributes and methods relevant to active weekdays for calendar.

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

- src/NetworkObjects/GTFSObjects/Calendar.h

4.4 Date Struct Reference

Represents a specific date in the Gregorian calendar.

```
#include <DateTime.h>
```

Public Attributes

- int year

  Year of the date (e.g., 2024).
- int month

  Month of the date (1 = January, ..., 12 = December).
- int day

  Day of the month (1-31).
- int weekday

  Day of the week (0 = Sunday, 1 = Monday, ..., 6 = Saturday).

4.4.1 Detailed Description

Represents a specific date in the Gregorian calendar.

Includes fields for the year, month, day, and the day of the week.

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

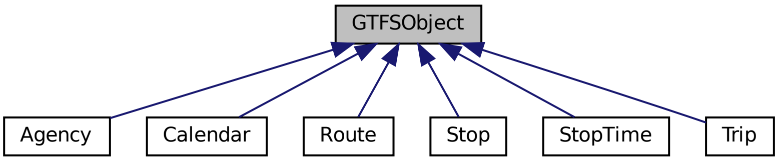
- src/DateTime.h

4.5 GTFSObject Class Reference

Represents a generic GTFS object.

```
#include <GTFSObject.h>
```

Inheritance diagram for GTFSObject:



Public Member Functions

- void setField (const std::string &field, const std::string &value)

  Sets the value of a field.
- std::string getField (const std::string &field) const

  Retrieves the value of a field.
- const std::unordered_map< std::string, std::string > & getFields () const

  Gets all fields as an unordered map.
- bool hasField (const std::string &field) const

  Checks if a field exists.

Protected Attributes

- std::unordered_map< std::string, std::string > fields

  Map of field names and values.

4.5.1 Detailed Description

Represents a generic GTFS object.

This class serves as a base class for all GTFS objects. It provides a generic interface for setting and getting field values.

4.5.2 Member Function Documentation

4.5.2.1 getField()

```
std::string GTFSObject::getField (
            const std::string & field ) const
```

Retrieves the value of a field.

Parameters

field	The name of the field to retrieve.

Returns

The value of the specified field.

Exceptions

std::runtime_error	If the field does not exist.

4.5.2.2 getFields()

```
const std::unordered_map< std::string, std::string > & GTFSObject::getFields ( ) const
```

Gets all fields as an unordered map.

Returns

A reference to the map of fields.

4.5.2.3 hasField()

```
bool GTFSObject::hasField (
             const std::string & field ) const
```

Checks if a field exists.

Parameters

field	The name of the field to check.

Returns

True if the field exists, false otherwise.

4.5.2.4 setField()

```
void GTFSObject::setField (
             const std::string & field,
             const std::string & value )
```

Sets the value of a field.

Parameters

field	The name of the field.
value	The value to assign to the field.

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

- src/NetworkObjects/GTFSObjects/GTFSObject.h
- src/NetworkObjects/GTFSObjects/GTFSObject.cpp

4.6 Journey Struct Reference

Represents an entire journey consisting of multiple steps.

```
#include <DataStructures.h>
```

Public Attributes

- std::vector< JourneyStep > steps

  Steps making up the journey.
- int departure_secs

  Overall departure time in seconds from midnight.
- Day departure_day

  Departure day of the journey.
- int arrival_secs

  Overall arrival time in seconds from midnight.
- Day arrival_day

  Arrival day of the journey.
- int duration

  Total duration of the journey in seconds.

4.6.1 Detailed Description

Represents an entire journey consisting of multiple steps.

The `Journey` structure contains details about all steps in the journey, as well as overall departure and arrival times and durations.

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

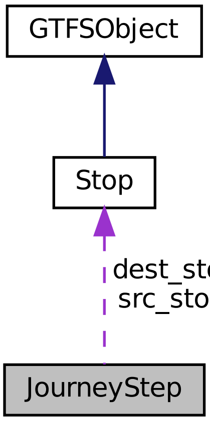
- src/NetworkObjects/DataStructures.h

4.7 JourneyStep Struct Reference

Represents a single step in a journey.

```
#include <DataStructures.h>
```

Collaboration diagram for JourneyStep:



Public Attributes

- std::optional< std::string > route_id

  ID of the route, or `std::nullopt` for footpaths.
- std::optional< std::string > trip_id

  ID of the trip, or `std::nullopt` for footpaths.
- std::optional< std::string > agency_name

  Name of the agency, or `std::nullopt` for footpaths.
- Stop ∗ src_stop {}

  Pointer to the source stop.
- Stop ∗ dest_stop {}

  Pointer to the destination stop.
- int departure_secs {}

  Departure time in seconds from midnight.
- Day day {}

  Day of the journey step.
- int duration {}

  Duration of the step in seconds.
- int arrival_secs {}

  Arrival time in seconds from midnight.

4.7.1 Detailed Description

Represents a single step in a journey.

A journey step can correspond to a trip or a footpath. It contains information about the source and destination stops, departure and arrival times, and duration.

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

- src/NetworkObjects/DataStructures.h

4.8 nested_pair_hash Struct Reference

Hash function for nested pairs of strings.

```
#include <DataStructures.h>
```

Public Member Functions

- std::size_t operator() (const std::pair< std::pair< std::string, std::string >, std::string > &nested_pair) const

  Computes the hash value for a nested pair of strings.

4.8.1 Detailed Description

Hash function for nested pairs of strings.

Provides a custom hash implementation for nested pairs of strings, used in unordered containers for hierarchical keys.

4.8.2 Member Function Documentation

4.8.2.1 operator()()

```
std::size_t nested_pair_hash::operator() (
            const std::pair< std::pair< std::string, std::string >, std::string > & nested↩
_pair ) const  [inline]
```

Computes the hash value for a nested pair of strings.

Parameters

nested_pair	The nested pair to hash.

Returns

The computed hash value.

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

- src/NetworkObjects/DataStructures.h

4.9 pair_hash Struct Reference

Hash function for a pair of strings.

```
#include <DataStructures.h>
```

Public Member Functions

- std::size_t operator() (const std::pair< std::string, std::string > &pair) const

  Computes the hash value for a pair of strings.

4.9.1 Detailed Description

Hash function for a pair of strings.

Provides a custom hash implementation for pairs of strings, used in unordered containers like `std::unordered_map` and `std::unordered_set`.

4.9.2 Member Function Documentation

4.9.2.1 operator()()

```
std::size_t pair_hash::operator() (
            const std::pair< std::string, std::string > & pair ) const  [inline]
```

Computes the hash value for a pair of strings.

Parameters

pair	The pair of strings to hash.

Returns

The computed hash value.

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

- src/NetworkObjects/DataStructures.h

4.10 Parser Class Reference

Class for parsing GTFS data files and organizing the information.

`#include <Parser.h>`

Public Member Functions

- Parser (std::string directory)

  Constructor for the Parser class.
- std::unordered_map< std::string, Agency > getAgencies ()

  Gets the parsed agencies data.
- std::unordered_map< std::string, Calendar > getCalendars ()

  Gets the parsed calendars data.
- std::unordered_map< std::string, Stop > getStops ()

  Gets the parsed stops data.
- std::unordered_map< std::pair< std::string, std::string >, Route, pair_hash > getRoutes ()

  Gets the parsed routes data.
- std::unordered_map< std::string, Trip > getTrips ()

  Gets the parsed trips data.
- std::unordered_map< std::pair< std::string, std::string >, StopTime, pair_hash > getStopTimes ()

  Gets the parsed stop times data.

Private Member Functions

- void parseAgencies ()

  Parses the agencies file and stores the results in the agencies_ map.
- void parseCalendars ()

  Parses the calendars file and stores the results in the calendars_ map.
- void parseRoutes ()

  Parses the routes file and stores the results in the routes_ map.
- void parseStops ()

  Parses the stops file and stores the results in the stops_ map.
- void parseTrips ()

  Parses the trips file and stores the results in the trips_ map.
- void parseStopTimes ()

  Parses the stop times file and stores the results in the stop_times_ map.
- void associateData ()

  Associates data across various GTFS components (routes, trips, stops, etc.).

Private Attributes

- std::string inputDirectory
- std::unordered_map< std::string, Agency > agencies_

  A map from agency IDs to Agency objects.
- std::unordered_map< std::string, Calendar > calendars_

  A map from calendar IDs to Calendar objects.
- std::unordered_map< std::string, Stop > stops_

  A map from stop IDs to Stop objects.
- std::unordered_map< std::pair< std::string, std::string >, Route, pair_hash > routes_

  A map from (route_id, direction_id) to Route objects.
- std::unordered_map< std::string, Trip > trips_

  A map from trip IDs to Trip objects.
- std::unordered_map< std::pair< std::string, std::string >, StopTime, pair_hash > stop_times_

  A map from (trip_id, stop_id) to StopTime objects.

4.10.1 Detailed Description

Class for parsing GTFS data files and organizing the information.

This class is responsible for parsing various GTFS data files such as agencies, calendars, stops, routes, trips, and stop times. It stores the parsed data in appropriate data structures and allows access to the parsed information.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Parser()

```
Parser::Parser (
            std::string directory )  [explicit]
```

Constructor for the Parser class.

Initializes the parser with the specified directory containing the GTFS data files.

Parameters

in	*directory*	Path to the directory containing the GTFS files.

4.10.3 Member Function Documentation

4.10.3.1 associateData()

```
void Parser::associateData ( )  [private]
```

Associates data across various GTFS components (routes, trips, stops, etc.).

This method processes the data from different GTFS files and associates the relevant information such as matching trips with corresponding stops and stop times.

4.10.3.2 getAgencies()

```
std::unordered_map< std::string, Agency > Parser::getAgencies ( )
```

Gets the parsed agencies data.

Returns

A map of agency IDs to Agency objects.

4.10.3.3 getCalendars()

```
std::unordered_map< std::string, Calendar > Parser::getCalendars ( )
```

Gets the parsed calendars data.

Returns

A map of calendar IDs to Calendar objects.

4.10.3.4 getRoutes()

```
std::unordered_map< std::pair< std::string, std::string >, Route, pair_hash > Parser::get←
Routes ( )
```

Gets the parsed routes data.

Returns

A map of (route_id, direction_id) pairs to Route objects.

4.10.3.5 getStops()

```
std::unordered_map< std::string, Stop > Parser::getStops ( )
```

Gets the parsed stops data.

Returns

A map of stop IDs to Stop objects.

4.10.3.6 getStopTimes()

```
std::unordered_map< std::pair< std::string, std::string >, StopTime, pair_hash > Parser←
::getStopTimes ( )
```

Gets the parsed stop times data.

Returns

A map of (trip_id, stop_id) pairs to StopTime objects.

4.10.3.7 getTrips()

```
std::unordered_map< std::string, Trip > Parser::getTrips ( )
```

Gets the parsed trips data.

Returns

A map of trip IDs to Trip objects.

4.10.4 Member Data Documentation

4.10.4.1 agencies_

```
std::unordered_map<std::string, Agency> Parser::agencies_ [private]
```

A map from agency IDs to Agency objects.

Maps to store parsed data.

4.10.4.2 inputDirectory

```
std::string Parser::inputDirectory [private]
```

Directory where the input files are located.

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

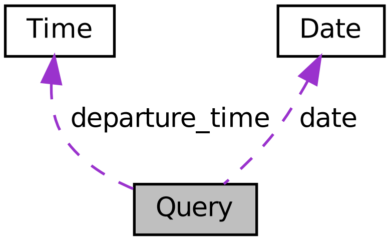
- src/Parser.h
- src/Parser.cpp

4.11 Query Struct Reference

Represents a transit query.

```
#include <DataStructures.h>
```

Collaboration diagram for Query:



Public Attributes

- std::string source_id

  ID of the source stop.
- std::string target_id

  ID of the target stop.
- Date date

  Date of the journey.
- Time departure_time

  Desired departure time for the journey.

4.11.1 Detailed Description

Represents a transit query.

This structure is used to define a user's query for transit planning, including source and target stops, the desired date, and departure time.

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

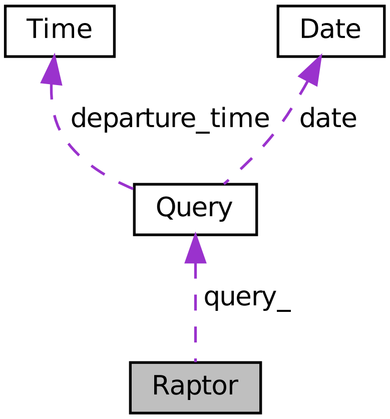
- src/NetworkObjects/DataStructures.h

4.12 Raptor Class Reference

Implements the RAPTOR algorithm for finding Pareto-optimal journeys.

```
#include <Raptor.h>
```

Collaboration diagram for Raptor:



Public Member Functions

- Raptor ()=default

  Default constructor for the Raptor class.
- Raptor (const std::unordered_map< std::string, Agency > &agencies_, const std::unordered_map< std::string, Calendar > &calendars_, const std::unordered_map< std::string, Stop > &stops, const std::unordered_map< std::pair< std::string, std::string >, Route, pair_hash > &routes, const std::unordered_map< std::string, Trip > &trips, const std::unordered_map< std::pair< std::string, std::string >, StopTime, pair_hash > &stop_times)

  Parameterized constructor for Raptor.
- void setQuery (const Query &query)

  Sets the query for the Raptor algorithm.
- std::vector< Journey > findJourneys ()

  Finds all Pareto-optimal journeys.
- const std::unordered_map< std::string, Stop > & getStops () const

  Gets the stops in the system.
- bool isValidJourney (Journey journey) const

  Validates if the given journey is valid.

Static Public Member Functions

- static void showJourney (const Journey &journey)

  Displays the steps of a journey.

Private Member Functions

- void initializeFootpaths ()

  Initializes the footpaths between stops.
- void initializeAlgorithm ()

  Initializes the algorithm by setting required parameters.
- void setMinArrivalTime (const std::string &stop_id, StopInfo stop_info)

  Sets the minimum arrival time for a given stop.
- void fillActiveTrips (Day day)

  Fills the active trips for a given day.
- void setUpperBound ()

  Sets the upper bound for the search, based on previous round.
- void resetMarkedStops ()

  Cleans marked stops and updates previous marked stops.
- std::unordered_set< std::pair< std::pair< std::string, std::string >, std::string >, nested_pair_hash > accumulateRoutesServingStops ()

  Accumulates routes serving each stop.
- void traverseRoutes (std::unordered_set< std::pair< std::pair< std::string, std::string >, std::string >, nested_pair_hash > routes_stops_set)

  Traverses the routes serving each stop.
- std::optional< std::pair< std::string, Day > > findEarliestTrip (const std::string &pi_stop_id, const std::pair< std::string, std::string > &route_key)

  Finds the earliest trip for a given stop and route.
- bool isValidTrip (const std::pair< std::string, std::string > &route_key, const StopTime &stop_time, const Day &day)

  Checks if a trip is valid based on the route and stop time.
- void traverseTrip (std::string &et_id, Day &et_day, std::string &pi_stop_id)

Traverses a specific trip.

- bool improvesArrivalTime (int arrival, const std::string &dest_id)

  Checks if a step improves the arrival time for a destination.
- void markStop (const std::string &stop_id, int arrival, const std::optional< std::string > &parent_trip_id, const std::optional< std::string > &parent_stop_id)

  Marks a stop with the arrival time, parent trip, and parent stop.
- void handleFootpaths ()

  Handles footpath logic during traversal.
- Journey reconstructJourney ()

  Reconstructs the journey based on the current round.

Static Private Member Functions

- static bool isServiceActive (const Calendar &calendar, const Date &date)

  Checks if the service is active based on the calendar and date.
- static bool earlier (int secondsA, std::optional< int > secondsB)

  Compares two arrival times to determine which is earlier.
- static bool isFootpath (const StopInfo &stop_info)

  Checks if the given stop info represents a footpath.
- static bool isDominatedByAny (const std::vector< Journey > &journeys, const Journey &journey)
- static void keepParetoOptimal (std::vector< Journey > &journeys)

  Keeps the Pareto-optimal journeys from a list of journeys.
- static bool dominates (const Journey &journey1, const Journey &journey2)

  Compares two journeys to check if one dominates the other.

Private Attributes

- std::unordered_map< std::string, Agency > agencies_

  Map of agency IDs to Agency objects.
- std::unordered_map< std::string, Calendar > calendars_

  Map of service IDs to Calendar objects.
- std::unordered_map< std::string, Stop > stops_

  Map of stop IDs to Stop objects.
- std::unordered_map< std::pair< std::string, std::string >, Route, pair_hash > routes_

  Map of route keys to Route objects.
- std::unordered_map< std::string, Trip > trips_

  Map of trip IDs to Trip objects.
- std::unordered_map< std::pair< std::string, std::string >, StopTime, pair_hash > stop_times_

  Map of stop time keys to StopTime objects.
- Query query_

  The current query for the RAPTOR algorithm.
- std::unordered_map< std::string, std::vector< StopInfo > > arrivals_

  Map of stop IDs to vectors of StopInfo for each k.
- std::unordered_set< std::string > prev_marked_stops

  Set of previously marked stops.
- std::unordered_set< std::string > marked_stops

  Set of currently marked stops.
- int k {}

  The current round of the algorithm.

4.12.1 Detailed Description

Implements the RAPTOR algorithm for finding Pareto-optimal journeys.

The Raptor class provides methods to set a query, find Pareto-optimal journeys, and print journey steps. It uses various data structures to store information about agencies, calendars, stops, routes, trips, and stop times.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Raptor() [1/2]

```
Raptor::Raptor ( )  [default]
```

Default constructor for the Raptor class.

Initializes the Raptor object with empty data structures.

4.12.2.2 Raptor() [2/2]

```
Raptor::Raptor (
             const std::unordered_map< std::string, Agency > & agencies_,
             const std::unordered_map< std::string, Calendar > & calendars_,
             const std::unordered_map< std::string, Stop > & stops,
             const std::unordered_map< std::pair< std::string, std::string >, Route, pair_hash
> & routes,
             const std::unordered_map< std::string, Trip > & trips,
             const std::unordered_map< std::pair< std::string, std::string >, StopTime, pair_hash
> & stop_times )
```

Parameterized constructor for Raptor.

Initializes the Raptor object with provided agency, calendar, stop, route, trip, and stop time data.

Parameters

in	*agencies_*	A map of agency IDs to Agency objects.
in	*calendars_*	A map of calendar IDs to Calendar objects.
in	*stops*	A map of stop IDs to Stop objects.
in	*routes*	A map of pairs of route IDs and direction IDs to Route objects.
in	*trips*	A map of trip IDs to Trip objects.
in	*stop_times*	A map of pairs of trip IDs and stop IDs to StopTime objects.

4.12.3 Member Function Documentation

4.12.3.1 accumulateRoutesServingStops()

```
std::unordered_set< std::pair< std::pair< std::string, std::string >, std::string >, nested_pair_hash
> Raptor::accumulateRoutesServingStops ( )  [private]
```

Accumulates routes serving each stop.

Returns

A set of routes that serve stops.

4.12.3.2 dominates()

```
bool Raptor::dominates (
            const Journey & journey1,
            const Journey & journey2 )  [static], [private]
```

Compares two journeys to check if one dominates the other.

Parameters

in	*journey1*	The first journey to be compared.
in	*journey2*	The second journey to be compared.

Returns

True if the first journey dominates the second, false otherwise.

4.12.3.3 earlier()

```
bool Raptor::earlier (
            int secondsA,
            std::optional< int > secondsB )  [static], [private]
```

Compares two arrival times to determine which is earlier.

Parameters

in	*secondsA*	The first arrival time in seconds.
in	*secondsB*	The second arrival time in seconds.

Returns

True if the first arrival time is earlier, false otherwise.

4.12.3.4 fillActiveTrips()

```
void Raptor::fillActiveTrips (
            Day day )  [private]
```

Fills the active trips for a given day.

Parameters

in	*day*	The day for which trips are being filled.

4.12.3.5 findEarliestTrip()

```
std::optional< std::pair< std::string, Day > > Raptor::findEarliestTrip (
            const std::string & pi_stop_id,
            const std::pair< std::string, std::string > & route_key )  [private]
```

Finds the earliest trip for a given stop and route.

Parameters

in	*pi_stop↵ _id*	The ID of the stop.
in	*route_key*	The key consisting of route and direction.

Returns

An optional pair of trip ID and day if found.

4.12.3.6 findJourneys()

```
std::vector< Journey > Raptor::findJourneys ( )
```

Finds all Pareto-optimal journeys.

This function uses the RAPTOR algorithm to compute all optimal journeys based on the provided query.

Returns

A vector of Journey objects representing the Pareto-optimal journeys.

4.12.3.7 getStops()

```
const std::unordered_map< std::string, Stop > & Raptor::getStops ( ) const
```

Gets the stops in the system.

Returns

A reference to the map of stop IDs to Stop objects.

4.12.3.8 improvesArrivalTime()

```
bool Raptor::improvesArrivalTime (
            int arrival,
            const std::string & dest_id )  [private]
```

Checks if a step improves the arrival time for a destination.

Parameters

in	*arrival*	The arrival time.
in	*dest← _id*	The destination stop ID.

Returns

True if the arrival time improves, false otherwise.

4.12.3.9 isDominatedByAny()

```
bool Raptor::isDominatedByAny (
            const std::vector< Journey > & journeys,
            const Journey & journey )  [static], [private]
```

Checks if a given journey is dominated by any other journey in the list.

Parameters

journeys	A list of all journeys to compare against.
journey	The journey to check.

Returns

True if the journey is dominated, otherwise false.

4.12.3.10 isFootpath()

```
bool Raptor::isFootpath (
            const StopInfo & stop_info ) [static], [private]
```

Checks if the given stop info represents a footpath.

Parameters

in	*stop_info*	The stop info to be checked.

Returns

True if the stop is a footpath, false otherwise.

4.12.3.11 isServiceActive()

```
bool Raptor::isServiceActive (
            const Calendar & calendar,
            const Date & date ) [static], [private]
```

Checks if the service is active based on the calendar and date.

Parameters

in	*calendar*	The calendar object containing service dates.
in	*date*	The date to check.

Returns

True if the service is active on the given date, false otherwise.

4.12.3.12 isValidJourney()

```
bool Raptor::isValidJourney (
            Journey journey ) const
```

Validates if the given journey is valid.

Checks whether the given journey meets the required criteria.

Parameters

in	*journey*	The Journey object to be validated.

Returns

True if the journey is valid, false otherwise.

4.12.3.13 isValidTrip()

```
bool Raptor::isValidTrip (
            const std::pair< std::string, std::string > & route_key,
            const StopTime & stop_time,
            const Day & day ) [private]
```

Checks if a trip is valid based on the route and stop time.

Parameters

in	*route_key*	The key consisting of route and direction.
in	*stop_time*	The stop time for the trip.
in	*day*	The day to check the trip against.

Returns

True if the trip is valid, false otherwise.

4.12.3.14 keepParetoOptimal()

```
void Raptor::keepParetoOptimal (
            std::vector< Journey > & journeys ) [static], [private]
```

Keeps the Pareto-optimal journeys from a list of journeys.

Parameters

in	*journeys*	The list of journeys to be filtered.

Returns

A list of Pareto-optimal journeys.

4.12.3.15 markStop()

```
void Raptor::markStop (
            const std::string & stop_id,
            int arrival,
            const std::optional< std::string > & parent_trip_id,
            const std::optional< std::string > & parent_stop_id )  [private]
```

Marks a stop with the arrival time, parent trip, and parent stop.

Parameters

in	*stop_id*	The ID of the stop.
in	*arrival*	The arrival time at the stop.
in	*parent_trip_id*	The ID of the parent trip.
in	*parent_stop_id*	The ID of the parent stop.

4.12.3.16 reconstructJourney()

```
Journey Raptor::reconstructJourney ( )  [private]
```

Reconstructs the journey based on the current round.

Returns

A Journey object representing the reconstructed journey.

4.12.3.17 setMinArrivalTime()

```
void Raptor::setMinArrivalTime (
            const std::string & stop_id,
            StopInfo stop_info )  [private]
```

Sets the minimum arrival time for a given stop.

Parameters

in	*stop_id*	The ID of the stop.
in	*stop_info*	The stop info containing the arrival time, parent trip ID, and parent stop ID.

4.12.3.18 setQuery()

```
void Raptor::setQuery (
            const Query & query )
```

Sets the query for the Raptor algorithm.

Parameters

in	*query*	The query containing the parameters for journey search.

4.12.3.19 showJourney()

```
void Raptor::showJourney (
             const Journey & journey )  [static]
```

Displays the steps of a journey.

Prints each step of the given journey to the console.

Parameters

in	*journey*	The Journey object to be displayed.

4.12.3.20 traverseRoutes()

```
void Raptor::traverseRoutes (
             std::unordered_set< std::pair< std::pair< std::string, std::string >, std←
::string >, nested_pair_hash > routes_stops_set )  [private]
```

Traverses the routes serving each stop.

Parameters

in	*routes_stops_set*	The set of routes and stops to be traversed.

4.12.3.21 traverseTrip()

```
void Raptor::traverseTrip (
             std::string & et_id,
             Day & et_day,
             std::string & pi_stop_id )  [private]
```

Traverses a specific trip.

Parameters

in,out	*et_id*	The trip ID.
in,out	*et_day*	The day of travel.
in,out	*pi_stop←id*	The stop ID for the trip.

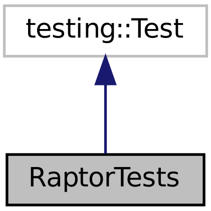
The documentation for this class was generated from the following files:

- src/Raptor.h
- src/Raptor.cpp

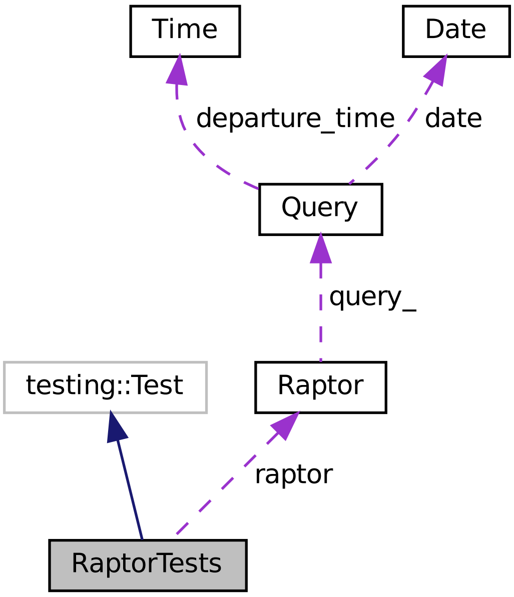
4.13 RaptorTests Class Reference

A test suite for validating the RAPTOR algorithm.

Inheritance diagram for RaptorTests:



Collaboration diagram for RaptorTests:



Static Protected Member Functions

- static void SetUpTestSuite ()

  Sets up data for all tests in the suite.

Static Protected Attributes

- static Raptor raptor

  Shared instance of the RAPTOR algorithm.

4.13.1 Detailed Description

A test suite for validating the RAPTOR algorithm.

This test suite uses Google Test to verify the correctness of the RAPTOR algorithm against various scenarios and edge cases.

4.13.2 Member Function Documentation

4.13.2.1 SetUpTestSuite()

```
static void RaptorTests::SetUpTestSuite ( )  [inline], [static], [protected]
```

Sets up data for all tests in the suite.

Loads GTFS data from specified directories and initializes the RAPTOR algorithm.

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

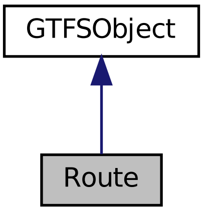
- tests/findJourneys.cpp

4.14 Route Class Reference

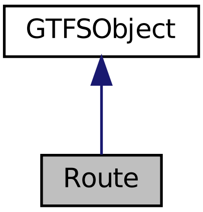
Represents a route in the GTFS data.

```
#include <Route.h>
```

Inheritance diagram for Route:



Collaboration diagram for Route:



Public Member Functions

- void addTripId (const std::string &trip_id)

  Adds a trip ID to the route.
- void addStopId (const std::string &stop_id)

  Adds a stop ID to the route.
- void sortTrips (const std::function< bool(const std::string &, const std::string &)> &comparator)

  Sorts the trips using a custom comparator.
- const std::vector< std::string > & getTripsIds () const

  Retrieves the list of trip IDs.
- const std::vector< std::string > & getStopsIds () const

  Retrieves the list of stop IDs.

Private Attributes

- std::vector< std::string > trips_ids

  Vector of trip IDs, sorted by earliest arrival time.
- std::vector< std::string > stops_ids

  Vector of stop IDs, sorted by stop sequence.

Additional Inherited Members

4.14.1 Detailed Description

Represents a route in the GTFS data.

This class inherits from GTFSObject and manages trip and stop information for a specific route. It provides methods for adding trip and stop IDs, retrieving sorted data, and defining custom sorting mechanisms.

4.14.2 Member Function Documentation

4.14.2.1 addStopId()

```
void Route::addStopId (
             const std::string & stop_id )
```

Adds a stop ID to the route.

Parameters

stop↩ _id	The ID of the stop to add.

4.14.2.2 addTripId()

```
void Route::addTripId (
             const std::string & trip_id )
```

Adds a trip ID to the route.

Parameters

trip↩ _id	The ID of the trip to add.

4.14.2.3 getStopsIds()

```
const std::vector< std::string > & Route::getStopsIds ( ) const
```

Retrieves the list of stop IDs.

Returns

A constant reference to the vector of stop IDs.

4.14.2.4 getTripsIds()

```
const std::vector< std::string > & Route::getTripsIds ( ) const
```

Retrieves the list of trip IDs.

Returns

A constant reference to the vector of trip IDs.

4.14.2.5 sortTrips()

```
void Route::sortTrips (
            const std::function< bool(const std::string &, const std::string &)> & comparator
)
```

Sorts the trips using a custom comparator.

Parameters

comparator	A function defining the sorting criteria.

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

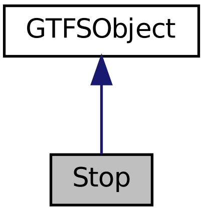
- src/NetworkObjects/GTFSObjects/Route.h
- src/NetworkObjects/GTFSObjects/Route.cpp

4.15 Stop Class Reference

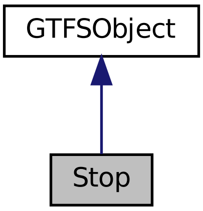
Represents a stop in the GTFS data.

```
#include <Stop.h>
```

Inheritance diagram for Stop:



Collaboration diagram for Stop:



Public Member Functions

- void addStopTimeKey (const std::pair< std::string, std::string > &stop_time_key)

  Adds a stop-time key (trip_id, stop_id) to the stop.
- void addRouteKey (const std::pair< std::string, std::string > &route_key)

  Adds a route key (route_id, direction_id) to the stop.
- void addFootpath (const std::string &other_id, int &duration)

  Adds a footpath to another stop.
- const std::vector< std::pair< std::string, std::string > > & getStopTimesKeys () const

  Retrieves the list of stop-time keys.
- const std::unordered_set< std::pair< std::string, std::string >, pair_hash > & getRouteKeys () const

  Retrieves the set of route keys.
- const std::unordered_map< std::string, int > & getFootpaths () const

  Retrieves the map of footpaths.
- void sortStopTimes (const std::function< bool(const std::pair< std::string, std::string > &, const std::pair< std::string, std::string > &)> &comparator)

  Sorts the stop times using a custom comparator.

Private Attributes

- std::vector< std::pair< std::string, std::string > > stop_times_keys

  Vector of stop-time keys, sorted by earliest departure time.
- std::unordered_set< std::pair< std::string, std::string >, pair_hash > routes_keys

  Set of route keys.
- std::unordered_map< std::string, int > footpaths

  Map of footpaths to other stops.

Additional Inherited Members

4.15.1 Detailed Description

Represents a stop in the GTFS data.

This class inherits from GTFSObject and manages stop time and route information for a specific stop. It provides methods for adding stop time and route IDs, retrieving sorted data, and defining custom sorting mechanisms.

4.15.2 Member Function Documentation

4.15.2.1 addFootpath()

```
void Stop::addFootpath (
             const std::string & other_id,
             int & duration )
```

Adds a footpath to another stop.

Parameters

other← _id	The ID of the other stop.
duration	The duration of the footpath in seconds.

4.15.2.2 addRouteKey()

```
void Stop::addRouteKey (
             const std::pair< std::string, std::string > & route_key )
```

Adds a route key (route_id, direction_id) to the stop.

Parameters

route_key	A pair representing the route key.

4.15.2.3 addStopTimeKey()

```
void Stop::addStopTimeKey (
             const std::pair< std::string, std::string > & stop_time_key )
```

Adds a stop-time key (trip_id, stop_id) to the stop.

Parameters

stop_time_key	A pair representing the stop-time key.

4.15.2.4 getFootpaths()

```
const std::unordered_map< std::string, int > & Stop::getFootpaths ( ) const
```

Retrieves the map of footpaths.

Returns

A constant reference to the map of footpaths.

4.15.2.5 getRouteKeys()

```
const std::unordered_set< std::pair< std::string, std::string >, pair_hash > & Stop::get↩
RouteKeys ( ) const
```

Retrieves the set of route keys.

Returns

A constant reference to the unordered set of route keys.

4.15.2.6 getStopTimesKeys()

```
const std::vector< std::pair< std::string, std::string > > & Stop::getStopTimesKeys ( ) const
```

Retrieves the list of stop-time keys.

Returns

A constant reference to the vector of stop-time keys.

4.15.2.7 sortStopTimes()

```
void Stop::sortStopTimes (
             const std::function< bool(const std::pair< std::string, std::string > &, const
std::pair< std::string, std::string > &)> & comparator )
```

Sorts the stop times using a custom comparator.

Parameters

comparator	A function defining the sorting criteria.

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

- src/NetworkObjects/GTFSObjects/Stop.h
- src/NetworkObjects/GTFSObjects/Stop.cpp

4.16 StopInfo Struct Reference

Represents information about a transit stop during a journey.

`#include <DataStructures.h>`

Public Attributes

- std::optional< int > arrival_seconds

    Arrival time in seconds, or `std::nullopt` *if unreachable.*
- std::optional< std::string > parent_trip_id

    ID of the parent trip, or `std::nullopt` *for footpaths.*
- std::optional< std::string > parent_stop_id

    ID of the parent stop, or `std::nullopt` *for first stops.*
- std::optional< Day > day

    Day of arrival, or `std::nullopt` *if unreachable.*

4.16.1 Detailed Description

Represents information about a transit stop during a journey.

This structure holds details about a stop's arrival time, the trip and stop it depends on, and the day of operation. Values are optional to handle cases where a stop is unreachable or is a starting point.

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

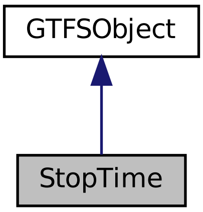
- src/NetworkObjects/DataStructures.h

4.17 StopTime Class Reference

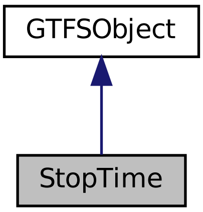
Represents a stop time in the GTFS data.

```
#include <StopTime.h>
```

Inheritance diagram for StopTime:



Collaboration diagram for StopTime:



Public Member Functions

- void setArrivalSeconds (int seconds)

  Sets the arrival time in seconds.
- void setDepartureSeconds (int seconds)

  Sets the departure time in seconds.
- int getArrivalSeconds () const

  Retrieves the arrival time in seconds.
- int getDepartureSeconds () const

  Retrieves the departure time in seconds.

Private Attributes

- int arrival_seconds {}

  Arrival time in seconds from midnight.
- int departure_seconds {}

  Departure time in seconds from midnight.

Additional Inherited Members

4.17.1 Detailed Description

Represents a stop time in the GTFS data.

This class inherits from GTFSObject and manages arrival and departure times for a specific stop. It provides methods for setting and getting arrival and departure times.

4.17.2 Member Function Documentation

4.17.2.1 getArrivalSeconds()

```
int StopTime::getArrivalSeconds ( ) const
```

Retrieves the arrival time in seconds.

Returns

The arrival time in seconds.

4.17.2.2 getDepartureSeconds()

```
int StopTime::getDepartureSeconds ( ) const
```

Retrieves the departure time in seconds.

Returns

The departure time in seconds.

4.17.2.3 setArrivalSeconds()

```
void StopTime::setArrivalSeconds (
             int seconds )
```

Sets the arrival time in seconds.

Parameters

seconds	The arrival time in seconds.

4.17.2.4 setDepartureSeconds()

```
void StopTime::setDepartureSeconds (
             int seconds )
```

Sets the departure time in seconds.

Parameters

seconds	The departure time in seconds.

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

- src/NetworkObjects/GTFSObjects/StopTime.h
- src/NetworkObjects/GTFSObjects/StopTime.cpp

4.18 Time Struct Reference

Represents a specific time of day in hours, minutes, and seconds.

```
#include <DateTime.h>
```

Public Attributes

- int hours

    Hours component of the time (0-23).
- int minutes

    Minutes component of the time (0-59).
- int seconds

    Seconds component of the time (0-59).

4.18.1 Detailed Description

Represents a specific time of day in hours, minutes, and seconds.

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

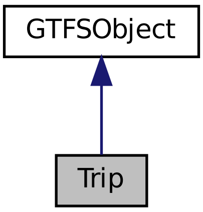
- src/DateTime.h

4.19 Trip Class Reference

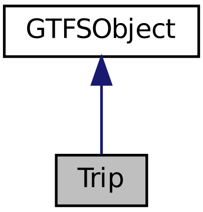
Represents a trip in the GTFS data.

```
#include <Trip.h>
```

Inheritance diagram for Trip:



Collaboration diagram for Trip:



Public Member Functions

- void addStopTimeKey (const std::pair< std::string, std::string > &stop_time_key)

  Adds a stop-time key (trip_id, stop_id) to the trip.
- const std::vector< std::pair< std::string, std::string > > & getStopTimesKeys () const

  Retrieves the list of stop-time keys.
- void sortStopTimes (const std::function< bool(const std::pair< std::string, std::string > &, const std::pair< std::string, std::string > &)> &comparator)

  Sorts the stop times using a custom comparator.
- void setActive (Day day, bool is_active)

  Sets the active status for a specific day.
- bool isActive (Day day) const

  Checks if a specific day is active.

Private Attributes

- std::vector< std::pair< std::string, std::string > > stop_times_keys

  Vector of stop-time keys, sorted by stopTime's sequence.
- std::unordered_map< Day, bool > active_days_

  Map of active days for the trip.

Additional Inherited Members

4.19.1 Detailed Description

Represents a trip in the GTFS data.

This class inherits from GTFSObject and manages stop time information for a specific trip. It provides methods for adding stop time keys, retrieving sorted data, and defining custom sorting mechanisms.

4.19.2 Member Function Documentation

4.19.2.1 addStopTimeKey()

```
void Trip::addStopTimeKey (
            const std::pair< std::string, std::string > & stop_time_key )
```

Adds a stop-time key (trip_id, stop_id) to the trip.

Parameters

stop_time_key	A pair representing the stop-time key.

4.19.2.2 getStopTimesKeys()

```
const std::vector< std::pair< std::string, std::string > > & Trip::getStopTimesKeys ( ) const
```

Retrieves the list of stop-time keys.

Returns

A constant reference to the vector of stop-time keys.

4.19.2.3 isActive()

```
bool Trip::isActive (
             Day day ) const
```

Checks if a specific day is active.

Parameters

day	

Returns

True if the day is active, false otherwise.

4.19.2.4 setActive()

```
void Trip::setActive (
             Day day,
             bool is_active )
```

Sets the active status for a specific day.

Parameters

day	
is_active	

4.19.2.5 sortStopTimes()

```
void Trip::sortStopTimes (
             const std::function< bool(const std::pair< std::string, std::string > &, const
std::pair< std::string, std::string > &)> & comparator )
```

Sorts the stop times using a custom comparator.

Parameters

comparator	A function defining the sorting criteria.

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

- src/NetworkObjects/GTFSObjects/Trip.h
- src/NetworkObjects/GTFSObjects/Trip.cpp

4.20 Utils Class Reference

A utility class providing various helper functions.

```
#include <Utils.h>
```

Static Public Member Functions

- static double manhattan (const double &lat1, const double &lon1, const double &lat2, const double &lon2)

  Computes the Manhattan distance between two geographical points.
- static int getDuration (const std::string &string_lat1, const std::string &string_lon1, const std::string &string↩ _lat2, const std::string &string_lon2)

  Calculates the duration between two geographical points in seconds.
- static std::string secondsToTime (std::optional< int > seconds)

  Converts a time in seconds to a string format (HH:MM:SS).
- static int timeToSeconds (const std::string &timeStr)

  Converts a time string to the equivalent number of seconds.
- static int timeToSeconds (const Time &time)

  Converts a Time object to the equivalent number of seconds.
- static std::vector< std::string > split (const std::string &str, char delimiter)

  Splits a string into a vector of substrings based on a delimiter.
- static std::string getFirstWord (const std::string &str)

  Retrieves the first word in a string.
- static void clean (std::string &input)

  Trims leading and trailing whitespace from a string.
- static bool isNumber (const std::string &str)

  Checks if a string represents a valid number.
- static int daysInMonth (int year, int month)

  Retrieves the number of days in a specific month of a specific year.
- static bool dateWithinRange (const Date &date, const std::string &start_date, const std::string &end_date)

  Checks if a date is within a specified date range.
- static Date addOneDay (Date date)

  Adds one day to a given date.
- static std::string dayToString (Day day)

  Converts a Day enum to a string representation.

4.20.1 Detailed Description

A utility class providing various helper functions.

This class contains static utility methods to handle mathematical calculations, time conversions, string manipulations, and date operations. These methods are used throughout the RAPTOR project to simplify code and provide common functionality.

4.20.2 Member Function Documentation

4.20.2.1 addOneDay()

```
Date Utils::addOneDay (
             Date date )  [static]
```

Adds one day to a given date.

This method increments the given date by one day.

Parameters

in	*date*	The date to which one day should be added.

Returns

The resulting date after adding one day.

4.20.2.2 clean()

```
void Utils::clean (
             std::string & input )  [static]
```

Trims leading and trailing whitespace from a string.

This method removes any leading or trailing whitespace from the given string.

Parameters

in,out	*line*	The line to be cleaned.

4.20.2.3 dateWithinRange()

```
bool Utils::dateWithinRange (
             const Date & date,
             const std::string & start_date,
             const std::string & end_date )  [static]
```

Checks if a date is within a specified date range.

This method checks whether a given date falls within the specified range of start and end dates.

Parameters

in	*date*	The date to be checked.
in	*start_date*	The start of the date range (in string format).
in	*end_date*	The end of the date range (in string format).

Returns

True if the date is within the range, false otherwise.

4.20.2.4 daysInMonth()

```
int Utils::daysInMonth (
             int year,
             int month )  [static]
```

Retrieves the number of days in a specific month of a specific year.

This method returns the number of days in a given month, accounting for leap years if applicable.

Parameters

in	*year*	The year of interest.
in	*month*	The month of interest (1-12).

Returns

The number of days in the specified month of the specified year.

4.20.2.5 dayToString()

```
std::string Utils::dayToString (
             Day day )  [static]
```

Converts a Day enum to a string representation.

This method converts a Day enum (Current or Next) to its string representation.

Parameters

in	*day*	The Day enum to be converted.

Returns

The string representation of the specified day.

4.20.2.6 getDuration()

```
int Utils::getDuration (
             const std::string & string_lat1,
             const std::string & string_lon1,
             const std::string & string_lat2,
             const std::string & string_lon2 )  [static]
```

Calculates the duration between two geographical points in seconds.

This method computes the duration based on the geographic distance between two sets of latitude and longitude coordinates, expressed as strings.

Parameters

in	*string_lat1*	Latitude of the first point as a string.
in	*string_lon1*	Longitude of the first point as a string.
in	*string_lat2*	Latitude of the second point as a string.
in	*string_lon2*	Longitude of the second point as a string.

Returns

The duration in seconds.

4.20.2.7 getFirstWord()

```
std::string Utils::getFirstWord (
             const std::string & str )  [static]
```

Retrieves the first word in a string.

This method extracts and returns the first word from a given string, stopping at the first space.

Parameters

in	*str*	The input string.

Returns

The first word in the string.

4.20.2.8 isNumber()

```
bool Utils::isNumber (
            const std::string & str )  [static]
```

Checks if a string represents a valid number.

This method checks whether the input string can be interpreted as a valid numerical value.

Parameters

in	*str*	The input string to be checked.

Returns

True if the string is a valid number, false otherwise.

4.20.2.9 manhattan()

```
double Utils::manhattan (
            const double & lat1,
            const double & lon1,
            const double & lat2,
            const double & lon2 )  [static]
```

Computes the Manhattan distance between two geographical points.

This method calculates the Manhattan (or "taxicab") distance between two points given their latitude and longitude in decimal degrees. This distance is useful for certain types of grid-based calculations.

Parameters

in	*lat1*	Latitude of the first point.
in	*lon1*	Longitude of the first point.
in	*lat2*	Latitude of the second point.
in	*lon2*	Longitude of the second point.

Returns

The Manhattan distance between the two points.

4.20.2.10 secondsToTime()

```
std::string Utils::secondsToTime (
            std::optional< int > seconds )  [static]
```

Converts a time in seconds to a string format (HH:MM:SS).

This method converts a given time in seconds into a formatted string representing the time in the "HH:MM:SS" format.

Parameters

in	*seconds*	The time in seconds.

Returns

A string representation of the time in "HH:MM:SS" format.

4.20.2.11 split()

```
std::vector< std::string > Utils::split (
            const std::string & str,
            char delimiter )  [static]
```

Splits a string into a vector of substrings based on a delimiter.

This method splits a string into parts wherever a specified delimiter appears.

Parameters

in	*str*	The input string to be split.
in	*delimiter*	The delimiter character to split the string by.

Returns

A vector of substrings split from the input string.

4.20.2.12 timeToSeconds() [1/2]

```
int Utils::timeToSeconds (
            const std::string & timeStr )  [static]
```

Converts a time string to the equivalent number of seconds.

This method converts a time string (e.g., "12:30:00") to the total number of seconds.

Parameters

in	*timeStr*	A time string in the "HH:MM:SS" format.

Returns

The total time in seconds.

4.20.2.13 timeToSeconds() [2/2]

```
int Utils::timeToSeconds (
             const Time & time )   [static]
```

Converts a Time object to the equivalent number of seconds.

This method converts a Time object to the total number of seconds since midnight.

Parameters

in	*time*	A Time object representing a specific time.

Returns

The total time in seconds.

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

- src/Utils.h
- src/Utils.cpp

Chapter 5

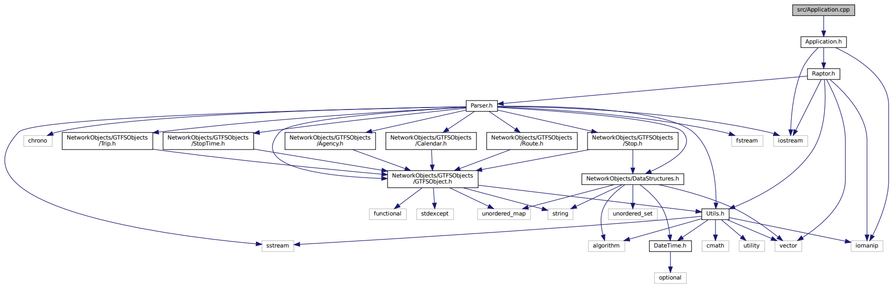
File Documentation

5.1 src/Application.cpp File Reference

Application class implementation.

```
#include "Application.h"
```

Include dependency graph for Application.cpp:


src/Application.cpp
Application.h
Raptor.h
Parser.h
chrono
NetworkObjects/GTFSObjects /Trip.h
NetworkObjects/GTFSObjects /StopTime.h
NetworkObjects/GTFSObjects /Agency.h
NetworkObjects/GTFSObjects /Calendar.h
NetworkObjects/GTFSObjects /Route.h
NetworkObjects/GTFSObjects /Stop.h
fstream
iostream
NetworkObjects/GTFSObjects /GTFSObject.h
NetworkObjects/DataStructures.h
functional
stdexcept
unordered_map
string
unordered_set
Utils.h
sstream
algorithm
DateTime.h
cmath
utility
vector
iomanip
optional

5.1.1 Detailed Description

Application class implementation.

This file contains the implementation of the Application class, which manages the initialization and execution of the RAPTOR application.

@autor Maria

Date

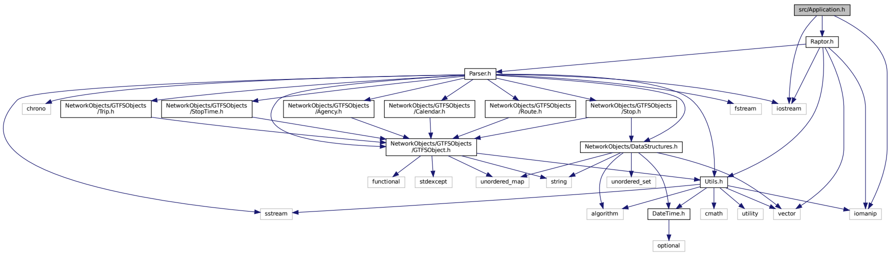
11/11/2024

5.2 src/Application.h File Reference

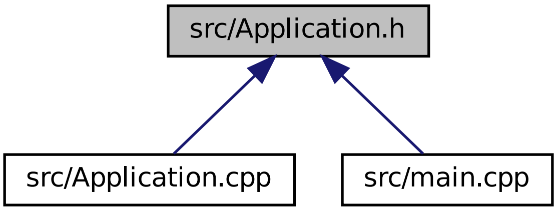
Defines the Application class, which manages the initialization and execution of the RAPTOR application.

```
#include "Raptor.h"
#include <iostream>
#include <iomanip>
```

Include dependency graph for Application.h:


src/Application.h
Raptor.h
Parser.h
chrono
NetworkObjects/GTFSObjects /Trip.h
NetworkObjects/GTFSObjects /StopTime.h
NetworkObjects/GTFSObjects /Agency.h
NetworkObjects/GTFSObjects /Calendar.h
NetworkObjects/GTFSObjects /Route.h
NetworkObjects/GTFSObjects /Stop.h
fstream
iostream
NetworkObjects/GTFSObjects /GTFSObject.h
NetworkObjects/DataStructures.h
functional
stdexcept
unordered_map
string
unordered_set
Utils.h
sstream
algorithm
DateTime.h
cmath
utility
vector
iomanip
optional

This graph shows which files directly or indirectly include this file:



Classes

- class Application

  The main application class for managing the RAPTOR transit algorithm.

5.2.1 Detailed Description

Defines the Application class, which manages the initialization and execution of the RAPTOR application.

This header provides declarations for the main application class, including methods for running the application, handling user queries, and interacting with the RAPTOR algorithm.

@autor Maria

Date

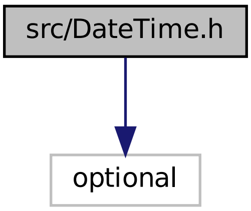
11/11/2024

5.3 src/DateTime.h File Reference

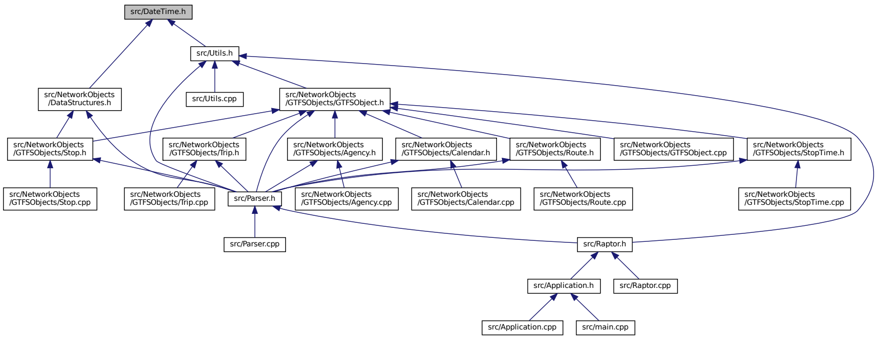
Provides data structures for representing dates and times in the RAPTOR application.

```
#include <optional>
```

Include dependency graph for DateTime.h:



This graph shows which files directly or indirectly include this file:


src/DateTime.h
src/Utils.h
src/NetworkObjects /DataStructures.h
src/Utils.cpp
src/NetworkObjects /GTFSObjects/GTFSObject.h
src/NetworkObjects /GTFSObjects/Stop.h
src/NetworkObjects /GTFSObjects/Trip.h
src/NetworkObjects /GTFSObjects/Agency.h
src/NetworkObjects /GTFSObjects/Calendar.h
src/NetworkObjects /GTFSObjects/Route.h
src/NetworkObjects /GTFSObjects/GTFSObject.cpp
src/NetworkObjects /GTFSObjects/StopTime.h
src/NetworkObjects /GTFSObjects/Stop.cpp
src/NetworkObjects /GTFSObjects/Trip.cpp
src/Parser.h
src/NetworkObjects /GTFSObjects/Agency.cpp
src/NetworkObjects /GTFSObjects/Calendar.cpp
src/NetworkObjects /GTFSObjects/Route.cpp
src/NetworkObjects /GTFSObjects/StopTime.cpp
src/Parser.cpp
src/Raptor.h
src/Application.h
src/Raptor.cpp
src/Application.cpp
src/main.cpp

Classes

- struct Date

  Represents a specific date in the Gregorian calendar.

- struct Time

  Represents a specific time of day in hours, minutes, and seconds.

Enumerations

- enum class Day { CurrentDay , NextDay }

  Represents the current or the next day for calculations.

Variables

- static constexpr int MIDNIGHT = 24 ∗ 3600

  Represents midnight in seconds (24 hours ∗ 3600 seconds per hour).
- constexpr const char ∗ weekdays_names [ ] = {"sunday", "monday", "tuesday", "wednesday", "thursday", "friday", "saturday"}

  Names of the weekdays starting from Sunday.

5.3.1 Detailed Description

Provides data structures for representing dates and times in the RAPTOR application.

This header defines the Date and Time structures, along with auxiliary enums and constants, for handling and manipulating temporal data.

5.3.2 Enumeration Type Documentation

5.3.2.1 Day

```
enum Day  [strong]
```

Represents the current or the next day for calculations.

Enumerator

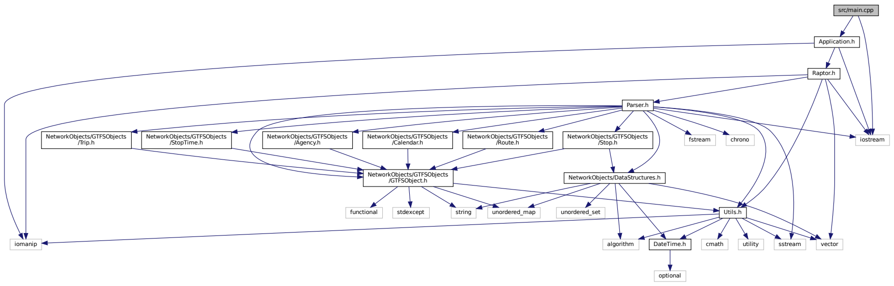
CurrentDay	Refers to the current day.
NextDay	Refers to the next day.

5.4 src/main.cpp File Reference

Entry point for the RAPTOR application.

```
#include <iostream>
#include "Application.h"
```

Include dependency graph for main.cpp:


src/main.cpp
Application.h
Raptor.h
Parser.h
NetworkObjects/GTFSObjects /Trip.h
NetworkObjects/GTFSObjects /StopTime.h
NetworkObjects/GTFSObjects /Agency.h
NetworkObjects/GTFSObjects /Calendar.h
NetworkObjects/GTFSObjects /Route.h
NetworkObjects/GTFSObjects /Stop.h
fstream
chrono
iostream
NetworkObjects/GTFSObjects /GTFSObject.h
NetworkObjects/DataStructures.h
functional
stdexcept
string
unordered_map
unordered_set
Utils.h
iomanip
algorithm
DateTime.h
cmath
utility
sstream
vector
optional

Functions

- int main (int argc, char ∗argv[ ])

  Main function for the RAPTOR application.

5.4.1 Detailed Description

Entry point for the RAPTOR application.

This file initializes the application, parses input directories, and starts the main event loop for processing user queries.

5.4.2 Function Documentation

5.4.2.1 main()

```
int main (
            int argc,
            char * argv[ ] )
```

Main function for the RAPTOR application.

This function parses command-line arguments or prompts the user for GTFS input directories, initializes the application, and starts the interactive event loop.

Parameters

argc	Number of command-line arguments.
argv	Array of command-line arguments.

Returns

Exit status of the application.

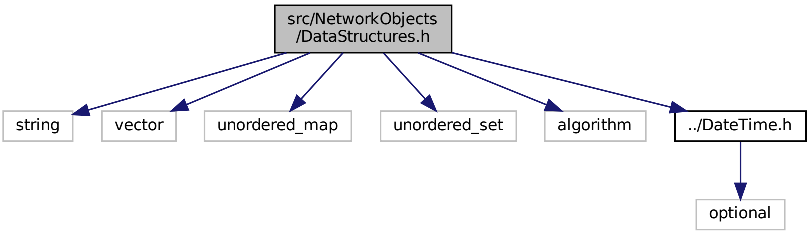
5.5 src/NetworkObjects/DataStructures.h File Reference

Defines core data structures and utility classes for the RAPTOR project.

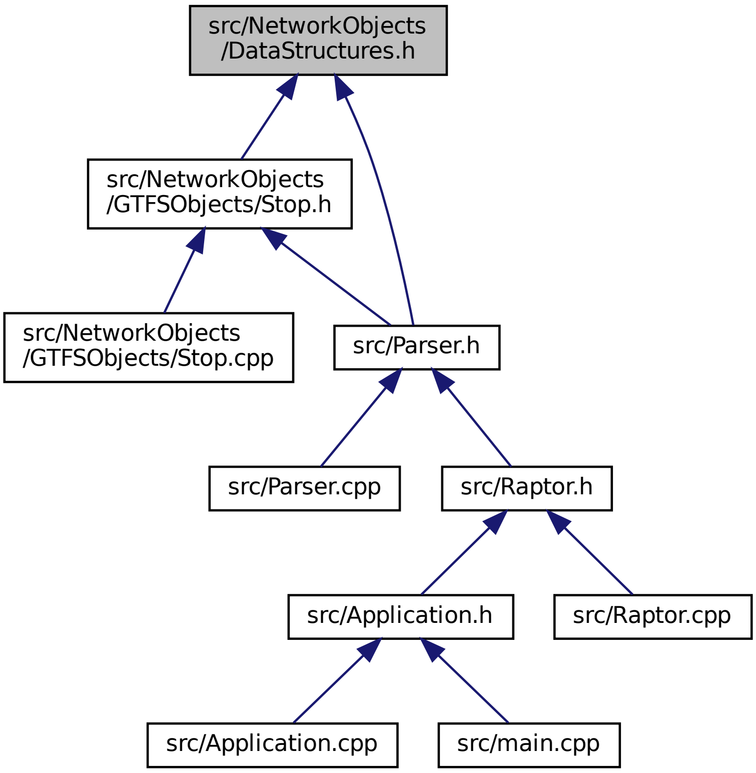
```
#include <string>
#include <vector>
#include <unordered_map>
#include <unordered_set>
#include <algorithm>
```

```
#include "../DateTime.h"
```

Include dependency graph for DataStructures.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Query

    Represents a transit query.

- struct StopInfo

    Represents information about a transit stop during a journey.

- struct JourneyStep

    Represents a single step in a journey.
- struct Journey

    Represents an entire journey consisting of multiple steps.
- struct pair_hash

    Hash function for a pair of strings.
- struct nested_pair_hash

    Hash function for nested pairs of strings.

5.5.1 Detailed Description

Defines core data structures and utility classes for the RAPTOR project.

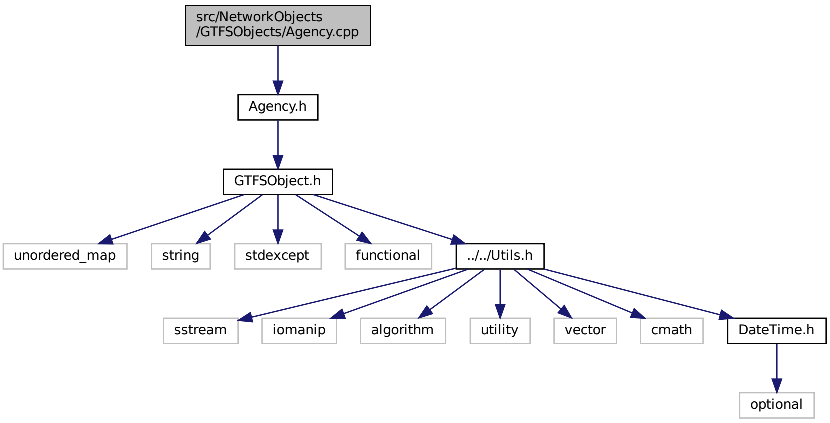
This header file includes declarations for structs like `Query`, `StopInfo`, `JourneyStep`, and `Journey`, which are used to represent transit queries, stop information, and journey details. It also provides hash functions for specific pair-based keys.

5.6 src/NetworkObjects/GTFSObjects/Agency.cpp File Reference

Implements the Agency class.

```
#include "Agency.h"
```

Include dependency graph for Agency.cpp:



5.6.1 Detailed Description

Implements the Agency class.

This file contains the implementation of the Agency class, which represents transit agencies in the GTFS dataset.

Note

> Currently, this file serves as a placeholder for future extensions.

@autor Maria

Date

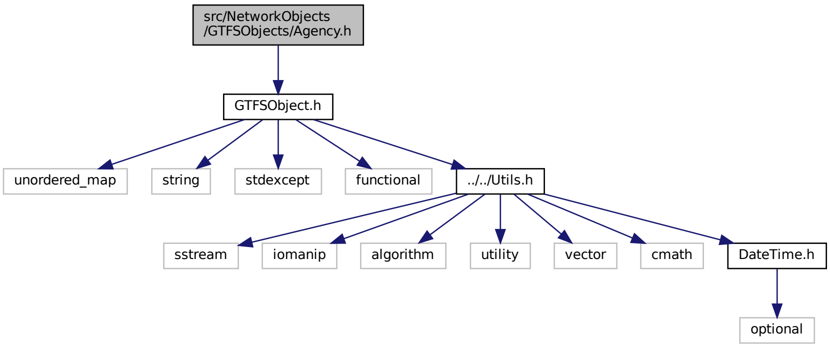
> 11/20/2024

5.7 src/NetworkObjects/GTFSObjects/Agency.h File Reference

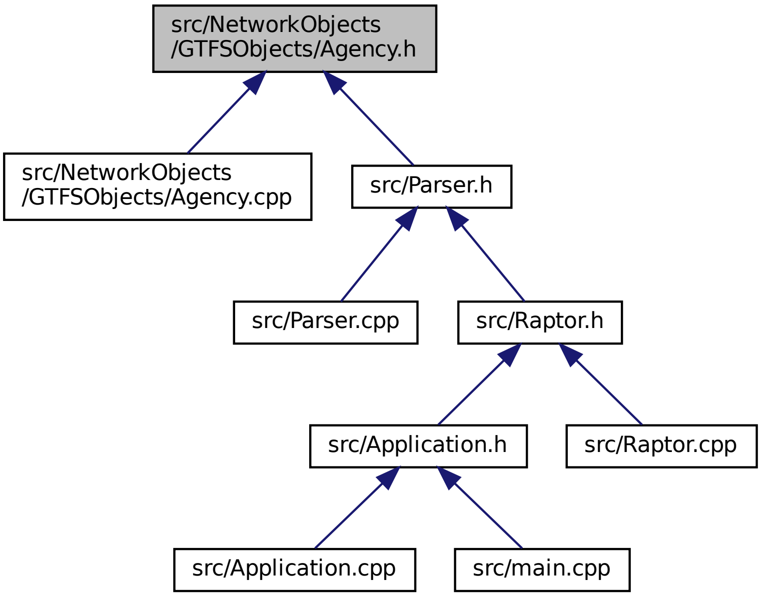
Defines the Agency class, representing transit agencies in the GTFS dataset.

```
#include "GTFSObject.h"
```

Include dependency graph for Agency.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Agency

  Represents a transit agency in the GTFS data.

5.7.1 Detailed Description

Defines the Agency class, representing transit agencies in the GTFS dataset.

This header file declares the Agency class, which inherits from GTFSObject. The class serves as a representation of the GTFS "agency.txt" file, storing information about transit agencies.

Author

Maria

Date

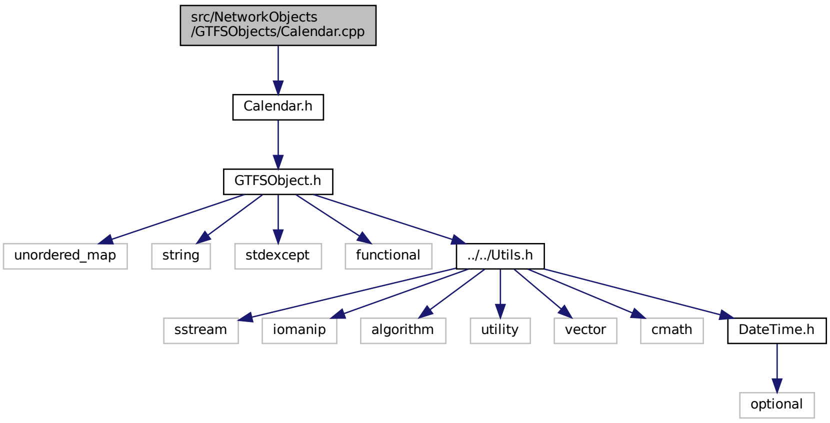
11/20/2024

5.8 src/NetworkObjects/GTFSObjects/Calendar.cpp File Reference

Implements the Calendar class.

```
#include "Calendar.h"
```

Include dependency graph for Calendar.cpp:



5.8.1 Detailed Description

Implements the Calendar class.

This file contains the implementation of the Calendar class, which represents active days of a calendar in the GTFS dataset.

Note

Currently, this file serves as a placeholder for future extensions.

@autor Maria

Date

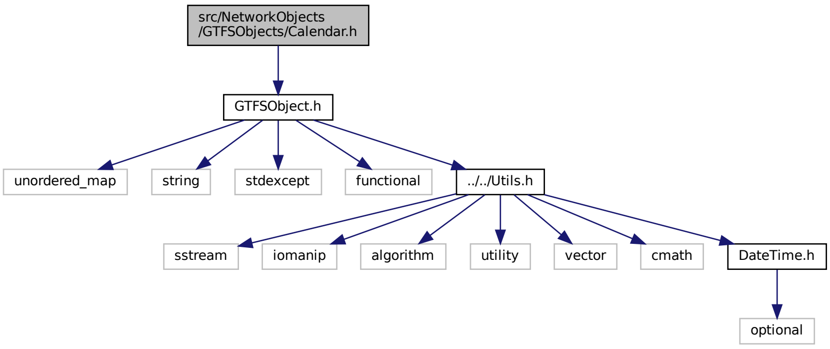
11/20/2024

5.9 src/NetworkObjects/GTFSObjects/Calendar.h File Reference

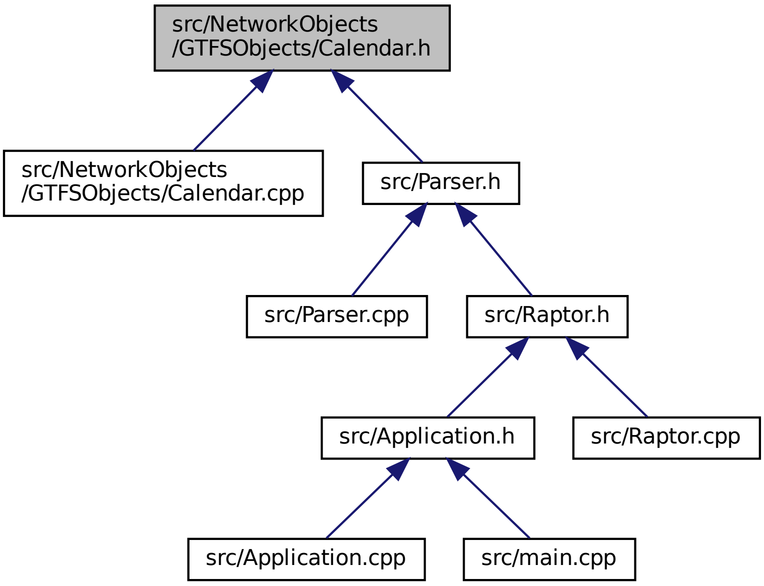
Defines the Calendar class, representing active weekdays for calendar in the GTFS dataset.

```
#include "GTFSObject.h"
```

Include dependency graph for Calendar.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Calendar

  Represents active weekdays for calendar in the GTFS data.

5.9.1 Detailed Description

Defines the Calendar class, representing active weekdays for calendar in the GTFS dataset.

This header file declares the Calendar class, which inherits from GTFSObject. The class serves as a representation of the GTFS "calendar.txt" file, storing information about active days of a calendar.

Author

Maria

Date

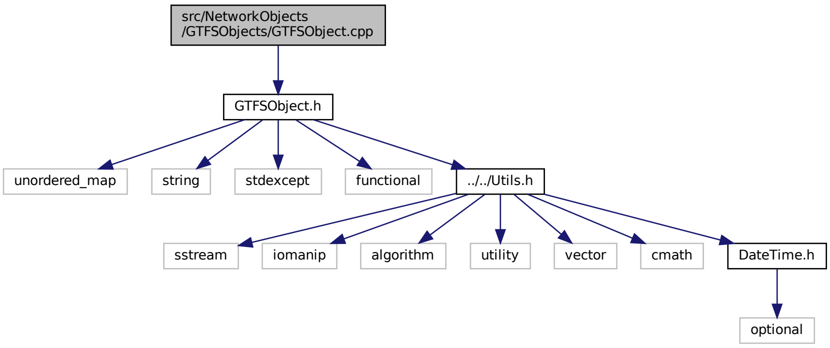
11/20/2024

5.10 src/NetworkObjects/GTFSObjects/GTFSObject.cpp File Reference

Implements the GTFSObject class.

```
#include "GTFSObject.h"
```

Include dependency graph for GTFSObject.cpp:



5.10.1 Detailed Description

Implements the GTFSObject class.

This file contains the implementation of the GTFSObject class, which represents a generic GTFS object.

@autor Maria

Date

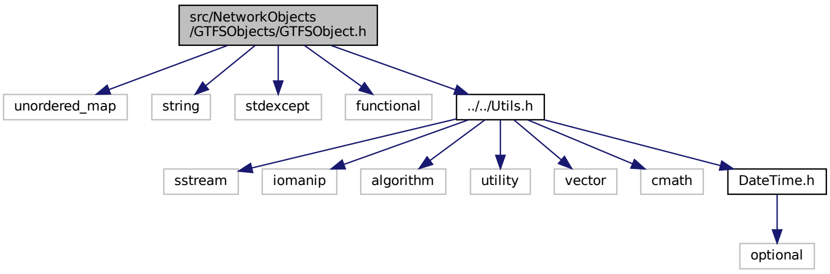
11/20/2024

5.11 src/NetworkObjects/GTFSObjects/GTFSObject.h File Reference

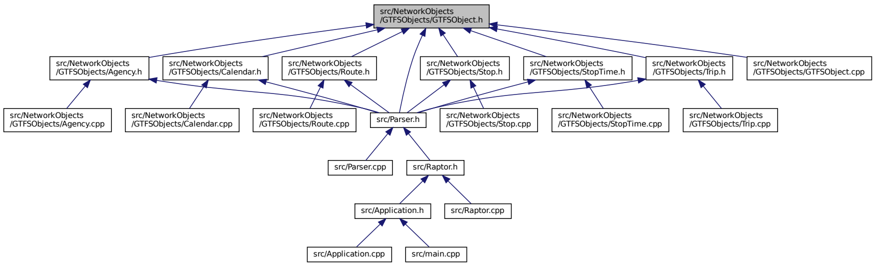
Defines the GTFSObject class, representing a generic GTFS object.

```
#include <unordered_map>
#include <string>
#include <stdexcept>
#include <functional>
#include "../../Utils.h"
```

Include dependency graph for GTFSObject.h:



This graph shows which files directly or indirectly include this file:



Classes

- class GTFSObject

  Represents a generic GTFS object.

5.11.1 Detailed Description

Defines the GTFSObject class, representing a generic GTFS object.

This header file declares the GTFSObject class, which serves as a base class for all GTFS objects.

Author

Maria

Date

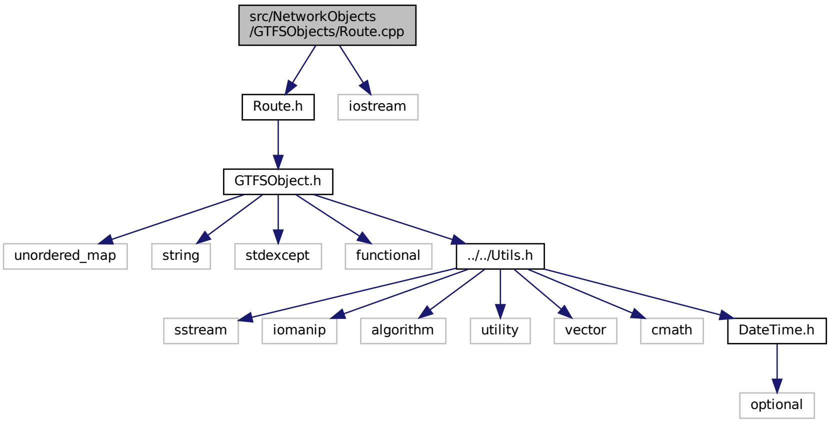
11/20/2024

5.12 src/NetworkObjects/GTFSObjects/Route.cpp File Reference

Route class implementation.

```
#include "Route.h"
#include <iostream>
```

Include dependency graph for Route.cpp:



5.12.1 Detailed Description

Route class implementation.

This file contains the implementation of the Route class, which represents a route in the GTFS dataset.

@autor Maria

Date

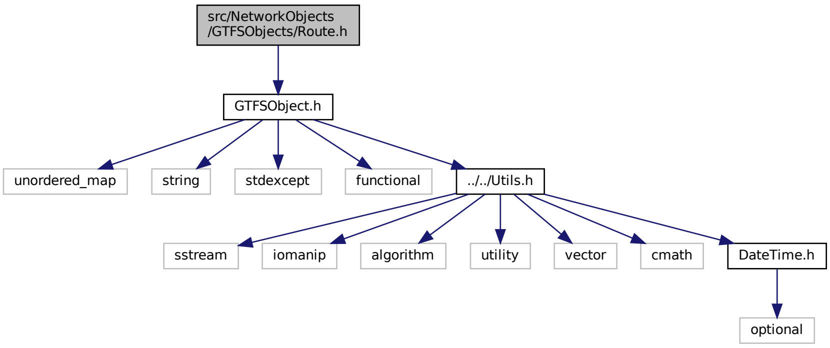
11/20/2024

5.13 src/NetworkObjects/GTFSObjects/Route.h File Reference

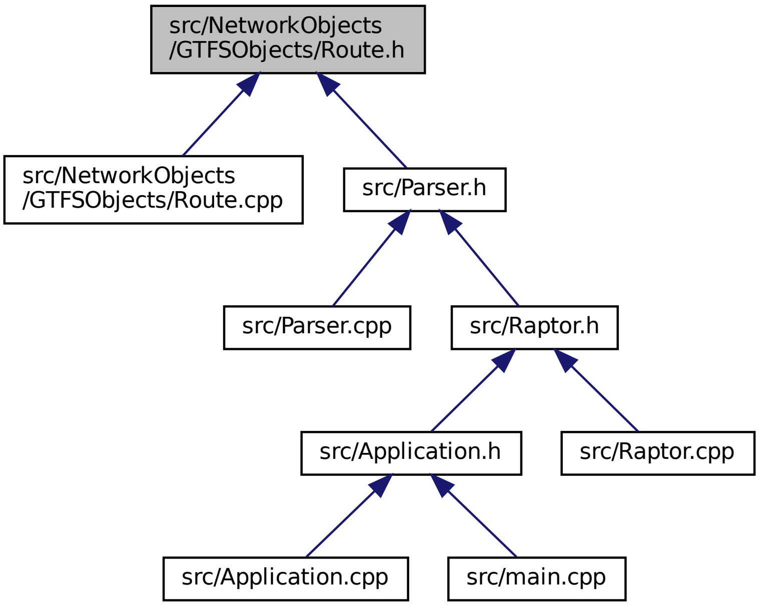
Defines the Route class, representing a route in the GTFS dataset.

```
#include "GTFSObject.h"
```

Include dependency graph for Route.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Route

    Represents a route in the GTFS data.

5.13.1 Detailed Description

Defines the Route class, representing a route in the GTFS dataset.

This header file declares the Route class, which inherits from GTFSObject. The class serves as a representation of the GTFS "route.txt" file, storing information about a route.

Author

Maria

Date

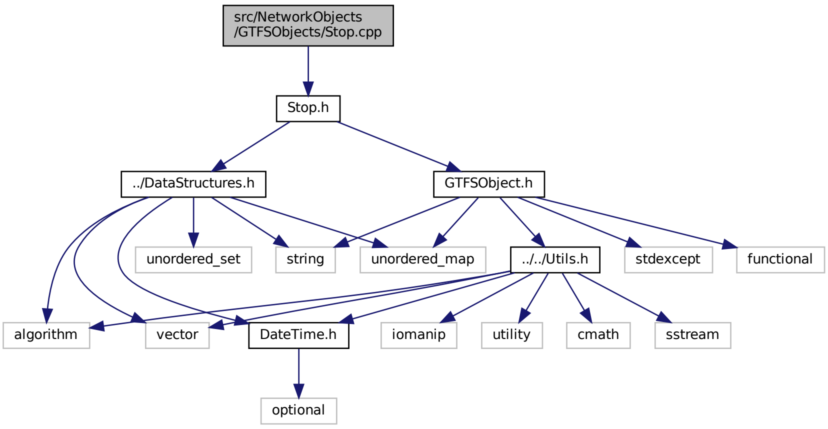
11/20/2024

5.14 src/NetworkObjects/GTFSObjects/Stop.cpp File Reference

Stop class implementation.

```
#include "Stop.h"
```

Include dependency graph for Stop.cpp:



5.14.1 Detailed Description

Stop class implementation.

This file contains the implementation of the Stop class, which represents a stop in the GTFS dataset.

@autor Maria

Date

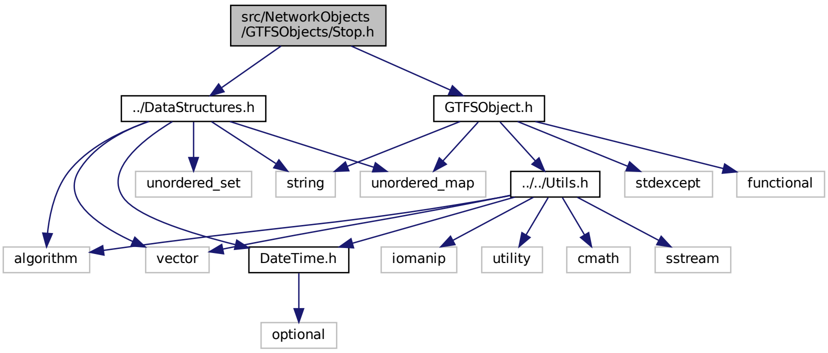
11/20/2024

5.15 src/NetworkObjects/GTFSObjects/Stop.h File Reference

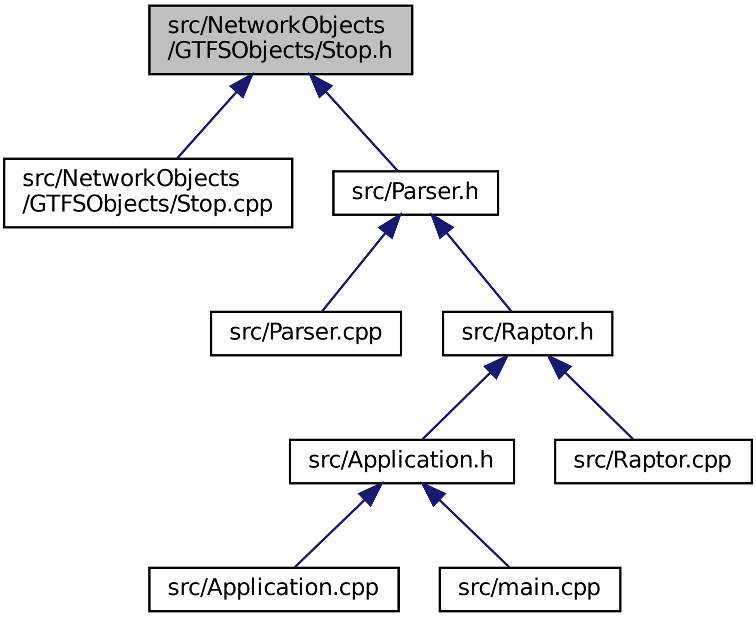
Defines the Stop class, representing a stop in the GTFS dataset.

```
#include "GTFSObject.h"
#include "../DataStructures.h"
```

Include dependency graph for Stop.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Stop

  Represents a stop in the GTFS data.

5.15.1 Detailed Description

Defines the Stop class, representing a stop in the GTFS dataset.

This header file declares the Stop class, which inherits from GTFSObject. The class serves as a representation of the GTFS "stop.txt" file, storing information about a stop.

Author

Maria

Date

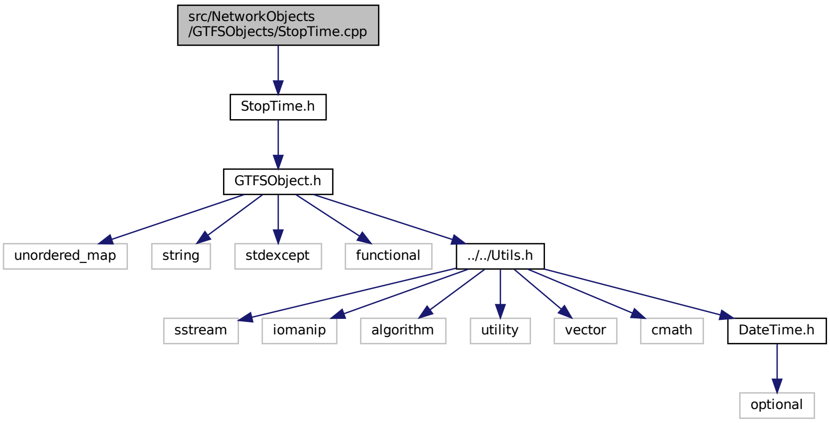
11/20/2024

5.16 src/NetworkObjects/GTFSObjects/StopTime.cpp File Reference

StopTime class implementation.

```
#include "StopTime.h"
```

Include dependency graph for StopTime.cpp:



5.16.1 Detailed Description

StopTime class implementation.

This file contains the implementation of the StopTime class, which represents a stop time in the GTFS dataset.

@autor Maria

Date

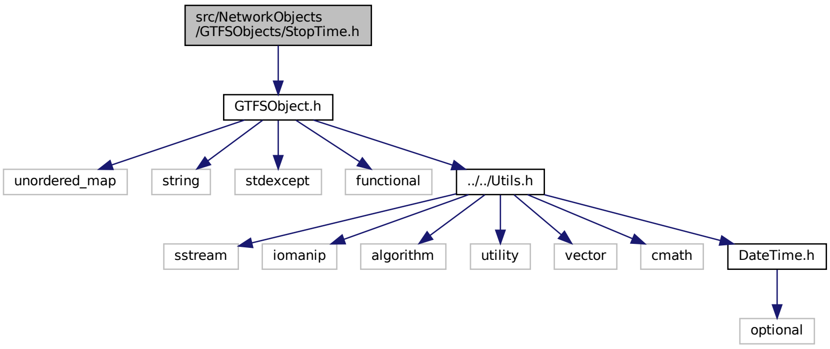
11/20/2024

5.17 src/NetworkObjects/GTFSObjects/StopTime.h File Reference

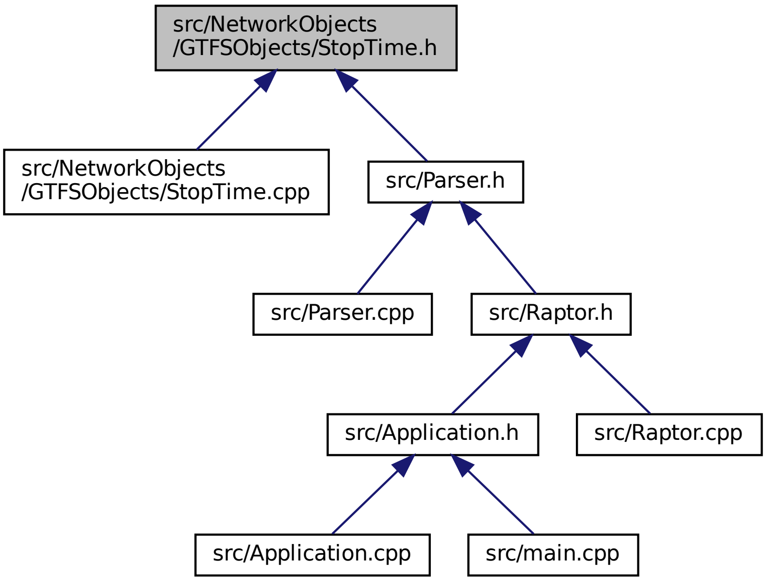
Defines the StopTime class, representing a stop time in the GTFS dataset.

```
#include "GTFSObject.h"
```

Include dependency graph for StopTime.h:



This graph shows which files directly or indirectly include this file:



Classes

- class StopTime

    Represents a stop time in the GTFS data.

5.17.1 Detailed Description

Defines the StopTime class, representing a stop time in the GTFS dataset.

This header file declares the StopTime class, which inherits from GTFSObject. The class serves as a representation of the GTFS "stop_times.txt" file, storing information about a stop time.

Author

Maria

Date

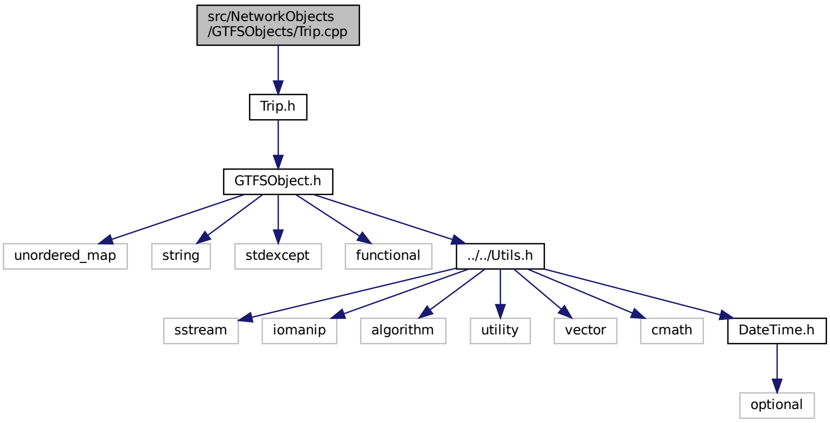
11/20/2024

5.18 src/NetworkObjects/GTFSObjects/Trip.cpp File Reference

Trip class implementation.

```
#include "Trip.h"
```

Include dependency graph for Trip.cpp:



5.18.1 Detailed Description

Trip class implementation.

This file contains the implementation of the Trip class, which represents a trip in the GTFS dataset.

@autor Maria

Date

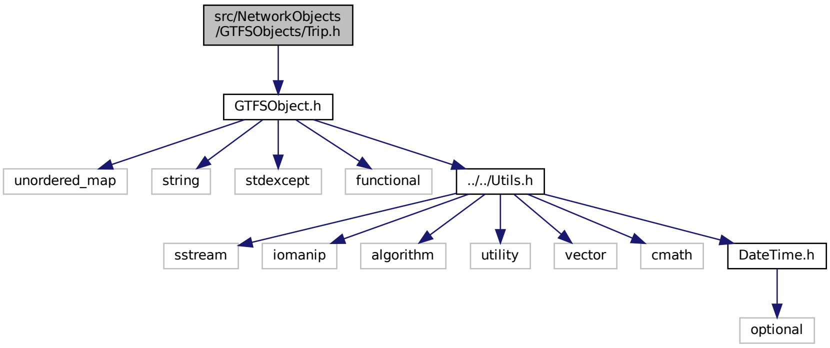
11/20/2024

5.19 src/NetworkObjects/GTFSObjects/Trip.h File Reference

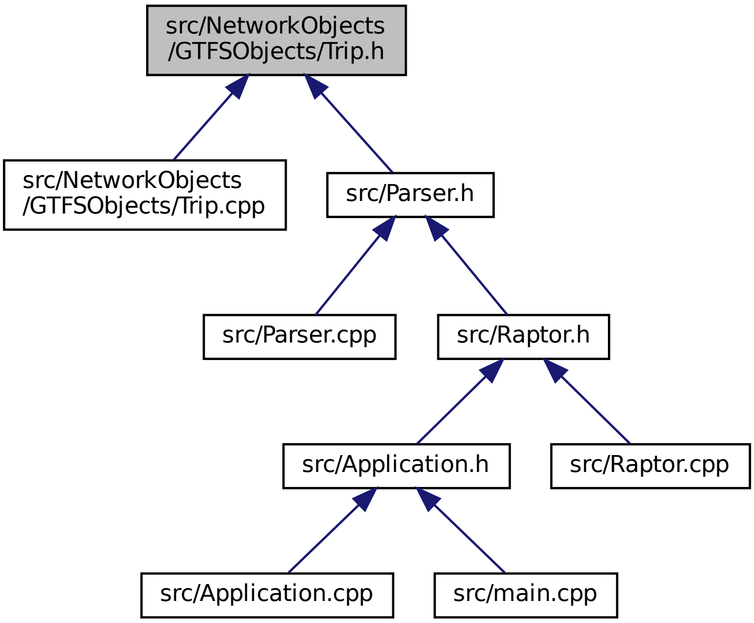
Defines the Trip class, representing a trip in the GTFS dataset.

```
#include "GTFSObject.h"
```

Include dependency graph for Trip.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Trip

    Represents a trip in the GTFS data.

5.19.1 Detailed Description

Defines the Trip class, representing a trip in the GTFS dataset.

This header file declares the Trip class, which inherits from GTFSObject. The class serves as a representation of the GTFS "trip.txt" file, storing information about a trip.

Author

Maria

Date

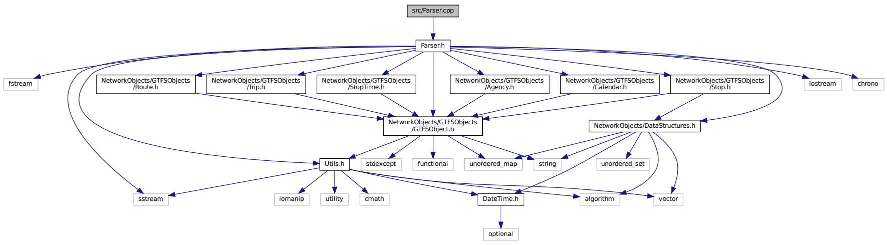
11/20/2024

5.20 src/Parser.cpp File Reference

Implementation of the Parser class.

```
#include "Parser.h"
```

Include dependency graph for Parser.cpp:


src/Parser.cpp
Parser.h
fstream
NetworkObjects/GTFSObjects /Route.h
NetworkObjects/GTFSObjects /Trip.h
NetworkObjects/GTFSObjects /StopTime.h
NetworkObjects/GTFSObjects /Agency.h
NetworkObjects/GTFSObjects /Calendar.h
NetworkObjects/GTFSObjects /Stop.h
iostream
chrono
NetworkObjects/GTFSObjects /GTFSObject.h
NetworkObjects/DataStructures.h
Utils.h
stdexcept
functional
unordered_map
string
unordered_set
sstream
iomanip
utility
cmath
DateTime.h
algorithm
vector
optional

5.20.1 Detailed Description

Implementation of the Parser class.

This file contains the implementation of the Parser class, which is responsible for parsing GTFS data.

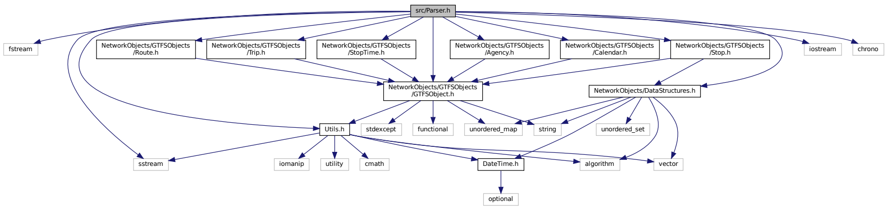
5.21 src/Parser.h File Reference

Provides the Parser class for parsing GTFS data files.

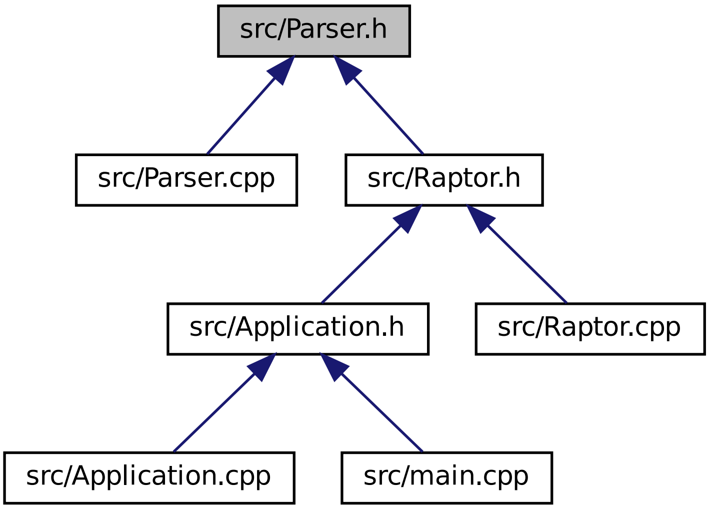
```
#include <fstream>
#include <sstream>
#include <iostream>
#include <chrono>
#include "Utils.h"
#include "NetworkObjects/GTFSObjects/GTFSObject.h"
#include "NetworkObjects/DataStructures.h"
```

```
#include "NetworkObjects/GTFSObjects/Agency.h"
#include "NetworkObjects/GTFSObjects/Calendar.h"
#include "NetworkObjects/GTFSObjects/Route.h"
#include "NetworkObjects/GTFSObjects/Stop.h"
#include "NetworkObjects/GTFSObjects/Trip.h"
#include "NetworkObjects/GTFSObjects/StopTime.h"
```

Include dependency graph for Parser.h:


src/Parser.h
fstream
NetworkObjects/GTFSObjects/Route.h
NetworkObjects/GTFSObjects/Trip.h
NetworkObjects/GTFSObjects/StopTime.h
NetworkObjects/GTFSObjects/Agency.h
NetworkObjects/GTFSObjects/Calendar.h
NetworkObjects/GTFSObjects/Stop.h
iostream
chrono
NetworkObjects/GTFSObjects/GTFSObject.h
NetworkObjects/DataStructures.h
Utils.h
stdexcept
functional
unordered_map
string
unordered_set
sstream
iomanip
utility
cmath
DateTime.h
algorithm
vector
optional

This graph shows which files directly or indirectly include this file:



Classes

- class Parser

  Class for parsing GTFS data files and organizing the information.

5.21.1 Detailed Description

Provides the Parser class for parsing GTFS data files.

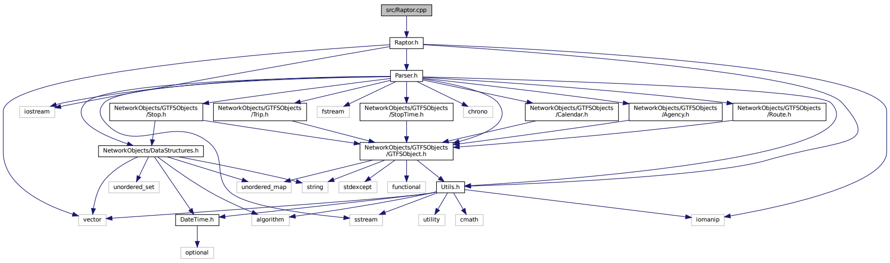
This header file declares the Parser class, which is responsible for parsing GTFS data files and associating the data to construct a transit network.

5.22 src/Raptor.cpp File Reference

Raptor class implementation.

```
#include "Raptor.h"
```

Include dependency graph for Raptor.cpp:


src/Raptor.cpp
Raptor.h
Parser.h
iostream
NetworkObjects/GTFSObjects/Stop.h
NetworkObjects/GTFSObjects/Trip.h
fstream
NetworkObjects/GTFSObjects/StopTime.h
chrono
NetworkObjects/GTFSObjects/Calendar.h
NetworkObjects/GTFSObjects/Agency.h
NetworkObjects/GTFSObjects/Route.h
NetworkObjects/DataStructures.h
NetworkObjects/GTFSObjects/GTFSObject.h
unordered_set
unordered_map
string
stdexcept
functional
Utils.h
vector
DateTime.h
algorithm
sstream
utility
cmath
iomanip
optional

5.22.1 Detailed Description

Raptor class implementation.

This file contains the implementation of the Raptor class, which represents the Round-Based Public Transit Routing algorithm, for journey planning.

@autor Maria

Date

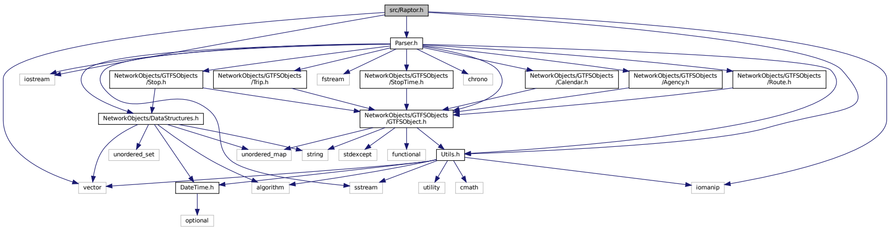
10/28/2024

5.23 src/Raptor.h File Reference

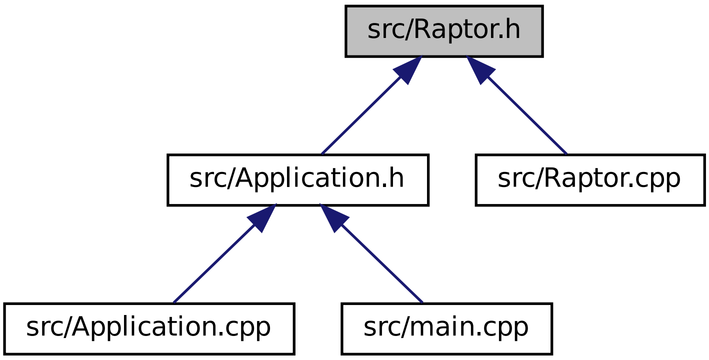
Defines the Raptor class for finding Pareto-optimal journeys in a transit network.

```
#include <iostream>
#include <vector>
#include <iomanip>
#include "Parser.h"
#include "Utils.h"
```

Include dependency graph for Raptor.h:


src/Raptor.h
Parser.h
iostream
NetworkObjects/GTFSObjects/Stop.h
NetworkObjects/GTFSObjects/Trip.h
fstream
NetworkObjects/GTFSObjects/StopTime.h
chrono
NetworkObjects/GTFSObjects/Calendar.h
NetworkObjects/GTFSObjects/Agency.h
NetworkObjects/GTFSObjects/Route.h
NetworkObjects/DataStructures.h
NetworkObjects/GTFSObjects/GTFSObject.h
unordered_set
unordered_map
string
stdexcept
functional
Utils.h
vector
DateTime.h
algorithm
sstream
utility
cmath
iomanip
optional

This graph shows which files directly or indirectly include this file:



Classes

- class Raptor

  Implements the RAPTOR algorithm for finding Pareto-optimal journeys.

5.23.1 Detailed Description

Defines the Raptor class for finding Pareto-optimal journeys in a transit network.

This header file declares the Raptor class, which implements the Round-Based Public Transit Routing algorithm.

The main method involve finding journeys.

The class also contains several private methods for initializing the algorithm, traversing routes, and reconstructing journeys.

Author

Maria

Date

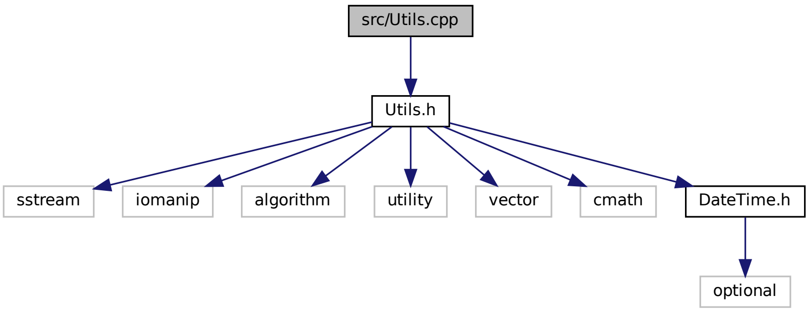
10/28/2024

5.24 src/Utils.cpp File Reference

Provides utility functions for the RAPTOR application.

```
#include "Utils.h"
```

Include dependency graph for Utils.cpp:



5.24.1 Detailed Description

Provides utility functions for the RAPTOR application.

This file contains utility functions for the RAPTOR application, including functions for calculating distances, durations, and time conversions.

Author

Maria

Date

10/28/2024

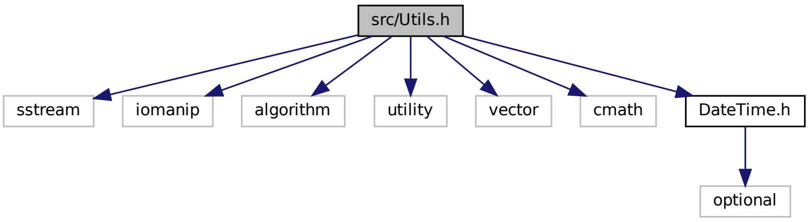
5.25 src/Utils.h File Reference

Provides utility functions for the RAPTOR application.

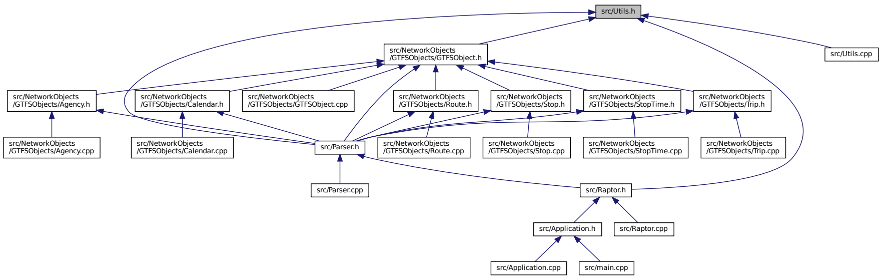
```
#include <sstream>
#include <iomanip>
#include <algorithm>
#include <utility>
#include <vector>
#include <cmath>
```

```
#include "DateTime.h"
```

Include dependency graph for Utils.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Utils

    A utility class providing various helper functions.

5.25.1 Detailed Description

Provides utility functions for the RAPTOR application.

This header file declares utility functions for the RAPTOR application, including functions for calculating distances, durations, and time conversions.

Author

Maria

Date

10/28/2024