a3-4-georgerapeanu

Generated by Doxygen 1.8.17

1 Assignment 02-03

1.1 Requirements

- Each student will be given one of the problems below.
- The solution must use the **C language**.
- The problem should be solved over 2 iterations, due `Week 3` and `Week 4`:

1.1.1 Week 3

- Solve at least requirements **a** and **b**.
- The vector used in the repository can be statically allocated.

1.1.2 Week 4

- Solve all problem requirements.
- Define a vector structure with specific operations using a dynamically allocated array.
- Use modular programming.
- Source code must be specified and include tests for all non-UI functions
- The program must not leak memory!
- Use a layered architecture for your application (domain, repository, controller, UI). User interface, domain and data access elements will be stored in different modules. The user interface module will only contain the user interface part.
- Have at least 10 entries available at application startup.
- Handle user input errors gracefully (replace program crashes with nice error messages :blush:).

1.2 Bonus possibilities

1. Implement the following requirements using function pointers **[deadline: week 4, bonus: 0.1p]**:
   - For requirement **b**, add a different type of filtering (of your choice).
   - For requirement **c**, add descending sorting. The user should choose the type of sort and the program will show the list of entities accordingly.
2. Provide 2 different implementations for the undo_operation/redo_operation functionality: one using a list of operations (this approach is a precursor of the `Command design pattern`) and one where each state of the repository is recorded (this approach is not unlike the `Memento design pattern`). Implement your dynamic array generically, such that it can contain any type of elements (use void∗). Use this structure for your repository, as well as to implement both undo_operation/redo_operation functionalities **[deadline: week 5, bonus: 0.1p]**.

1.3 Problem Statements

1.3.1 Pharmacy

John is the administrator of the *"Smiles"* Pharmacy. He needs a software application to help him manage his pharmacy's medicine stocks. Each **Medicine** has the following attributes: `name`, `concentration`, `quantity` and `price`. John wants the application to help him in the following ways:\ ∗∗(a)∗∗ Add, delete or update a medicine. A medicine is uniquely identified by its name and concentration. If a product that already exists is added, its quantity will be updated (the new quantity is added to the existing one).\ ∗∗(b)∗∗ See all available medicines containing a given string (if the string is empty, all the available medicines will be considered), sorted ascending by medicine name.\ ∗∗(c)∗∗ See only those medicines that are in short supply (quantity less than `X` items, where the value of `X` is user-provided).\ ∗∗(d)∗∗ Provide multiple undo_operation and redo_operation functionality. Each step will undo_operation/redo_operation the previous operation performed by the user.

1.3.2 Bakery

Mary runs her family's bakery, *"Bread'n Bagel"*. Every day she struggles with keeping up to date with available stocks of raw materials and would like a program to help her manage the business more effectively. Each **Material** used in the bakery must have: a `name`, a `supplier`, a `quantity` and the `expiration date`. Mary wants a software application that helps her in the following ways:\ ∗∗(a)∗∗ Add, delete and update a material. A raw material is uniquely identified by its name, supplier and expiration date. If a material that already exists is added, its quantity will be modified (the new quantity is added to the existing one).\ ∗∗(b)∗∗ See all available materials past their expiration date, containing a given string (if the string is empty, all materials past their expiration date will be considered).\ ∗∗(c)∗∗ Display all materials from a given supplier, which are in short supply (quantity less than a user-provided value), sorted ascending by their quantities.\ ∗∗(d)∗∗ Provide multiple undo_operation and redo_operation functionality. Each step will undo_operation/redo_operation the previous operation performed by the user.

1.3.3 Tourism Agency

The employees of *"Happy Holidays"* need an application to manage the offers that the agency has. Each **Offer** has a `type` (one of `seaside, mountain or city break`), a `destination`, a `departure date` and a `price`. The employees need the application to help them in the following ways:\ ∗∗(a)∗∗ Add, delete and update an offer. An offer is uniquely identified by its destination and departure dates.\ ∗∗(b)∗∗ Display all tourism offers whose destinations contain a given string (if the string is empty, all destinations are considered), and show them sorted ascending by price.\ ∗∗(c)∗∗ Display all offers of a given type, having their departure after a given date.\ ∗∗(d)∗∗ Provide multiple undo_operation and redo_operation functionality. Each step will undo_operation/redo_operation the previous operation performed by the user.

1.3.4 Real Estate Agency

Evelyn owns a real estate agency. Being also the only employee, she needs an application to help her manage all the real estates of her clients. Each **Estate** has a type (one of `house, apartment or penthouse`), an `address`, a `surface` and a `price`. Evelyn needs the application to help her in the following ways:\ **(a)** Add, delete or update an estate. An estate is uniquely identified by its address.\ **(b)** Display all estates whose address contains a given string (if the string is empty, all estates will be considered), sorted ascending by their price.\ **(c)** See all estates of a given type, having the surface greater than a user provided value.\ **(d)** Provide multiple undo_operation and redo_operation functionality. Each step will undo_operation/redo_operation the previous operation performed by the user.

1.3.5 Intelligent Refrigerator

The company *"Home SmartApps"* have decided to design a new intelligent refrigerator. Besides the hardware, they need a software application to manage the refrigerator. Each **Product** that the fridge can store has a `name`, a `category` (one of `dairy, sweets, meat or fruit`), a `quantity` and an `expiration date`. The software application will provide the following functionalities:\ **(a)** Add, delete or update a product. A product is uniquely identified by name and category. If a product that already exists is added, its quantity will be updated (the new quantity is added to the existing one).\ **(b)** Display all products whose name contains a given string (if the string is empty, all products from the refrigerator are considered), and show them sorted ascending by the existing quantity.\ **(c)** Display all products of a given category (if the category is empty, all types of food will be considered) whose expiration dates are close (expire in the following `X` days, where the value of `X` is user-provided).\ **(d)** Provide multiple undo_operation and redo_operation functionality. Each step will undo_operation/redo_↩ operation the previous operation performed by the user.

1.3.6 World Population Monitoring

The World Population Monitoring Organisation needs an application to help keep track of countries' populations. Each **Country** has a unique `name`, the `continent` it belongs to (one of `Europe, America, Africa, Australia or Asia`), and a population (stored in millions). The employees of the organisation need the application to help them in the following ways:\ **(a)** Add, delete or update a country. Updating must also consider the case of migration: a given number of people leave one country to migrate to another.\ **(b)** Display all countries whose name contains a given string (if the string is empty, all the countries should be considered).\ **(c)** Display all countries on a given continent (if the continent is empty, all countries will be considered), whose populations are greater than a given value, sorted ascending by population.\ **(d)** Provide multiple undo_↩ operation and redo_operation functionality. Each step will undo_operation/redo_operation the previous operation performed by the user.

I have to do `World Population Monitoring`

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

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

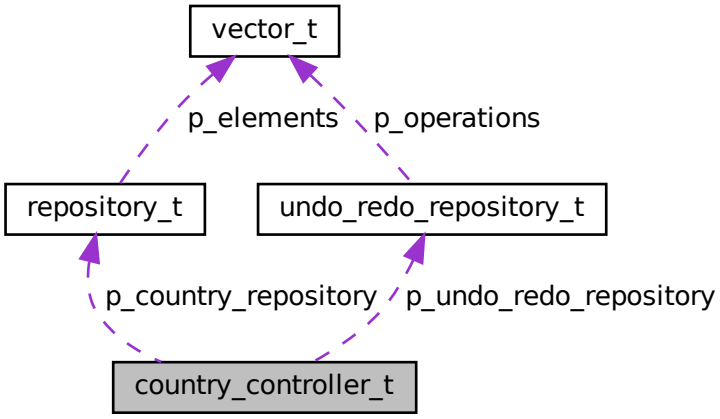
4 Class Documentation

4.1 country_controller_t Struct Reference

This struct contains the model for a country_controller.

`#include <country_controller.h>`

Collaboration diagram for country_controller_t:



Public Attributes

- repository_t * p_country_repository
  A pointer to the repository which stores the countries.
- undo_redo_repository_t * **p_undo_redo_repository**

4.1.1 Detailed Description

This struct contains the model for a country_controller.

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

- include/controller/country_controller.h

4.2 country_t Struct Reference

This struct defined in order to hold all the necessary information for a country.

`#include <country.h>`

Public Attributes

- char * name
  This holds a pointer to the start of a char array which contains the name of the country.
- char * continent
  This holds a pointer to the start of a char array which holds the name of the continent of the country.
- int population
  This holds the population of the (real) country. Due to the application designed, this will always pe printed in millions.

4.2.1 Detailed Description

This struct defined in order to hold all the necessary information for a country.

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

- include/domain/country.h

4.3 error_t Struct Reference

`#include <errors.h>`

Public Attributes

- char * **error_message**
- int is_fatal

  1 if the current error is fatal 0 otherwise

4.3.1 Detailed Description

this structure is used for errors

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

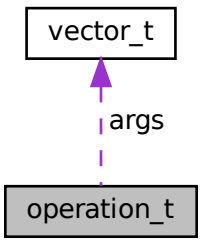
- include/domain/errors.h

4.4 operation_t Struct Reference

This struct contains the model for the operation_t struct.

`#include <operation.h>`

Collaboration diagram for operation_t:



Public Attributes

- void(∗ function)(vector_t ∗args, int ∗p_error)
  the function to be applied
- vector_t ∗ args
  the args with which it is called
- void(∗ free_args_data)(vector_t ∗args)
  the function which frees the extra data allocated for args

4.4.1 Detailed Description

This struct contains the model for the operation_t struct.

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

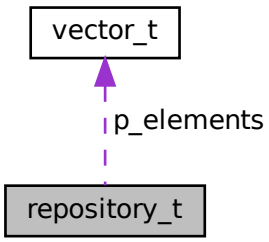
- include/domain/operation.h

4.5 repository_t Struct Reference

This is the model for a repository instance.

`#include <repository.h>`

Collaboration diagram for repository_t:



Public Attributes

- vector_t ∗ p_elements
  a pointer to the vector_t object which holds the elements

4.5.1 Detailed Description

This is the model for a repository instance.

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

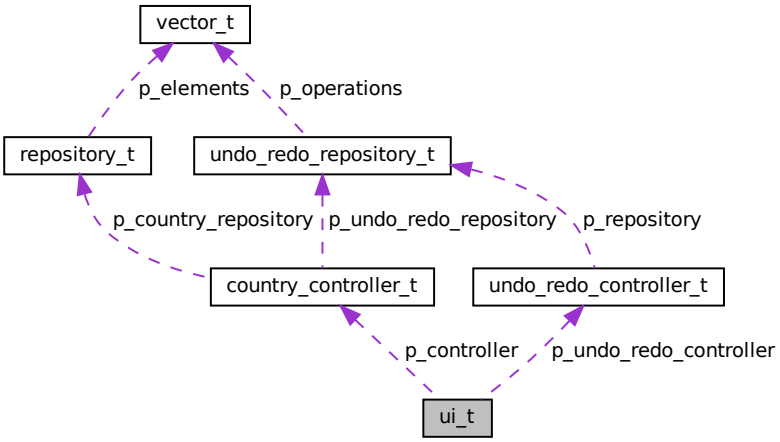
- include/repository/repository.h

4.6 ui_t Struct Reference

this is a model for the ui_t structure

`#include <ui.h>`

Collaboration diagram for ui_t:



Public Attributes

- country_controller_t * p_controller

  a pointer to the controller used by the ui
- undo_redo_controller_t * **p_undo_redo_controller**

4.6.1 Detailed Description

this is a model for the ui_t structure

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

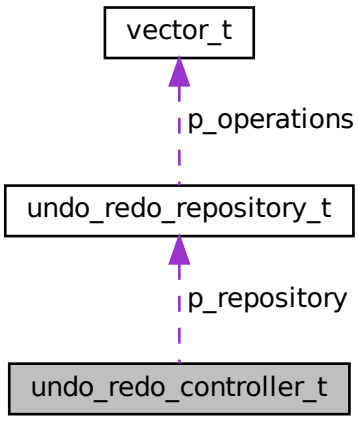
- include/ui/ui.h

4.7 undo_redo_controller_t Struct Reference

this struct implements the model for the undo_operation-redo_operation controller The struct will receive operations indirectly. This means, that any controllers that need undo_repository/redo_repository services will push operations directly to the repository used by this controller

`#include <undo_redo_controller.h>`

Collaboration diagram for undo_redo_controller_t:



Public Attributes

- undo_redo_repository_t * p_repository

  the repository containing undo_operation-redo_operation operations

4.7.1 Detailed Description

this struct implements the model for the undo_operation-redo_operation controller The struct will receive operations indirectly. This means, that any controllers that need undo_repository/redo_repository services will push operations directly to the repository used by this controller

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

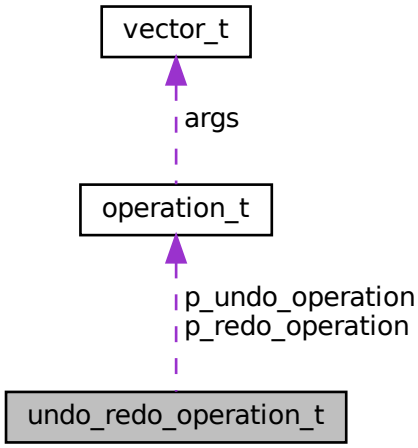
- include/controller/undo_redo_controller.h

4.8 undo_redo_operation_t Struct Reference

this is the model for an undo_operation-redo_operation operation

```
#include <undo_redo_operation.h>
```

Collaboration diagram for undo_redo_operation_t:



Public Attributes

- operation_t * p_undo_operation

  a pointer to the undo_operation
- operation_t * p_redo_operation

  a pointer to the redo_operation operation

4.8.1 Detailed Description

this is the model for an undo_operation-redo_operation operation

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

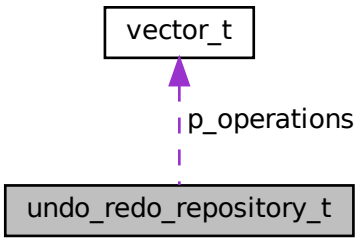
- include/domain/undo_redo_operation.h

4.9 undo_redo_repository_t Struct Reference

This is the model for an undo_redo_repository_t instance.

```
#include <undo_redo_repository.h>
```

Collaboration diagram for undo_redo_repository_t:



Public Attributes

- vector_t * p_operations

  a pointer to the vector_t object which holds the operations
- int head_idx

  this keeps track of the current operation

4.9.1 Detailed Description

This is the model for an undo_redo_repository_t instance.

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

- include/repository/undo_redo_repository.h

4.10 vector_t Struct Reference

Implements a dynamic array. It is generic, so you can hold any type in it.

`#include <vector.h>`

Public Attributes

- int element_size

  the size of an element of the array. This value assures that the structure is re-usable with any type
- void * elements

  pointer to a bytes buffer which holds the data for the structure. It should be casted to the specific type in order to correctly use the [] operator
- int size

  the size of the vector(the number of elements it currently holds)
- int capacity

  the capacity of the vector(the maximum number of elements it currently can hold)
- void(* free_elem_data)(void *p_elem)

  a function which is responsible for freeing any memory allocated on the heap for an element.

4.10.1 Detailed Description

Implements a dynamic array. It is generic, so you can hold any type in it.

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

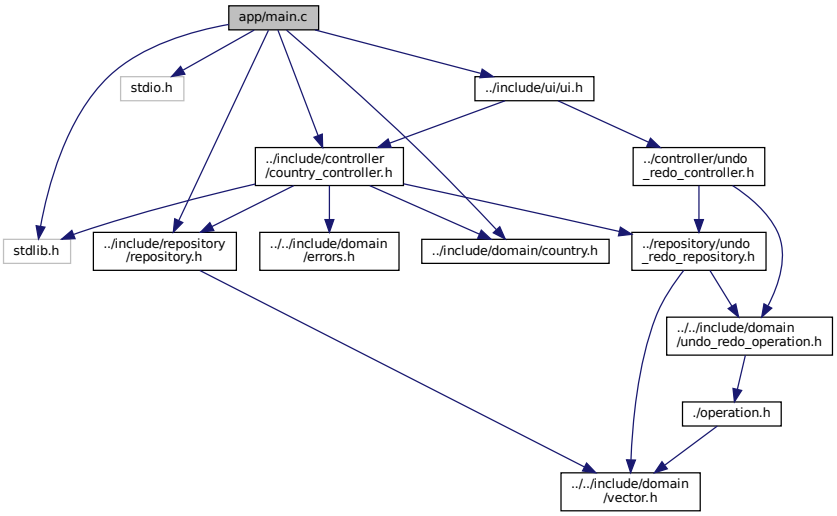
- include/domain/vector.h

5 File Documentation

5.1 app/main.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "../include/repository/repository.h"
#include "../include/domain/country.h"
#include "../include/controller/country_controller.h"
#include "../include/ui/ui.h"
```

Include dependency graph for main.c:



Functions

- int **main** ()

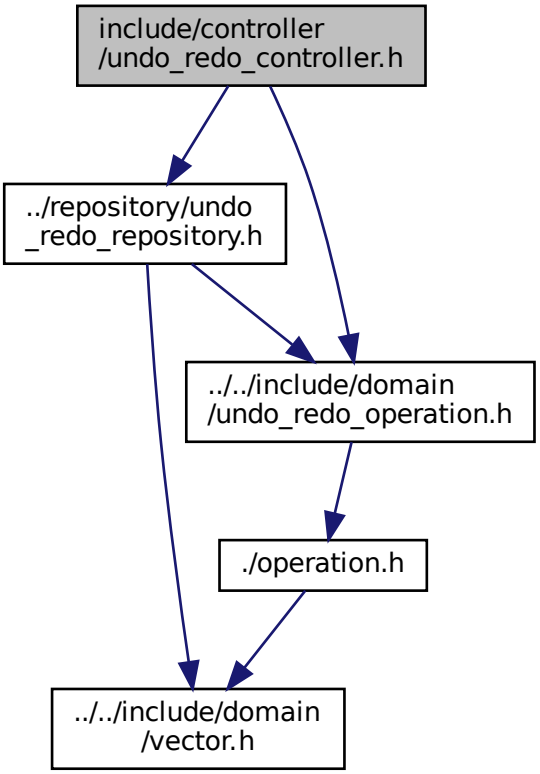
5.1.1 Detailed Description

this is where the application starts

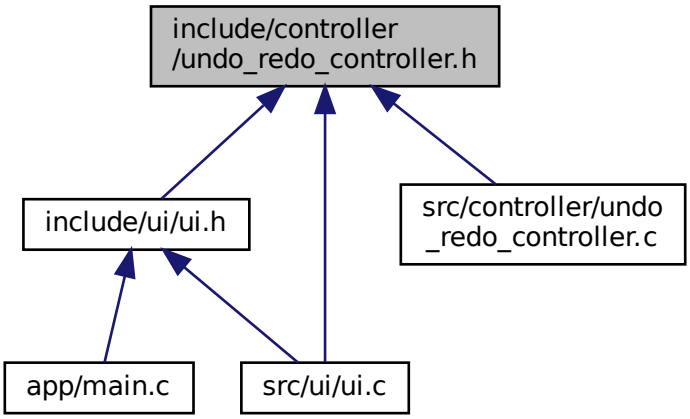
5.2 include/controller/undo_redo_controller.h File Reference

```
#include "../repository/undo_redo_repository.h"
#include "../domain/undo_redo_operation.h"
```

Include dependency graph for undo_redo_controller.h:



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



Classes

- struct undo_redo_controller_t

  this struct implements the model for the undo_operation-redo_operation controller The struct will receive operations indirectly. This means, that any controllers that need undo_repository/redo_repository services will push operations directly to the repository used by this controller

Functions

- undo_redo_controller_t * create_undo_redo_controller (undo_redo_repository_t *p_repository, int *p_error)

  this function creates an undo_repository-redo_repository controller instance
- void delete_undo_redo_controller (undo_redo_controller_t *p_undo_redo_controller)

  this function deletes an undo_repository redo_repository controller
- void undo (undo_redo_controller_t *p_undo_redo_controller, int *p_error)

  this function undoes an operation
- void redo (undo_redo_controller_t *p_undo_redo_controller, int *p_error)

  this function redoes an operation

5.2.1 Detailed Description

this file containing the model for implementing the undo_operation-redo_operation controller This controller is responsible for successfully applying undo_operation and redo_operation functions

5.2.2 Function Documentation

5.2.2.1 create_undo_redo_controller()

```
undo_redo_controller_t* create_undo_redo_controller (
        undo_redo_repository_t * p_repository,
        int * p_error )
```

this function creates an undo_repository-redo_repository controller instance

Parameters

p_repository	a pointer to the repository which should be used for storing operations
p_error	a pointer to the variable receiving the error code. Can be left NULL.

Returns

a pointer to an undo_repository-redo_repository controller instance

5.2.2.2 delete_undo_redo_controller()

```
void delete_undo_redo_controller (
        undo_redo_controller_t * p_undo_redo_controller )
```

this function deletes an undo_repository redo_repository controller

Parameters

p_undo_redo_controller	a pointer to the undo_repository redo_repository controller

5.2.2.3 redo()

```
void redo (
        undo_redo_controller_t * p_undo_redo_controller,
        int * p_error )
```

this function redoes an operation

Parameters

p_undo_redo_controller	a pointer to the undo_repository-redo_repository controller
p_error	a pointer to the varialbe which receives the error code. Can be left NULL.

5.2.2.4 undo()

```
void undo (
        undo_redo_controller_t * p_undo_redo_controller,
        int * p_error )
```

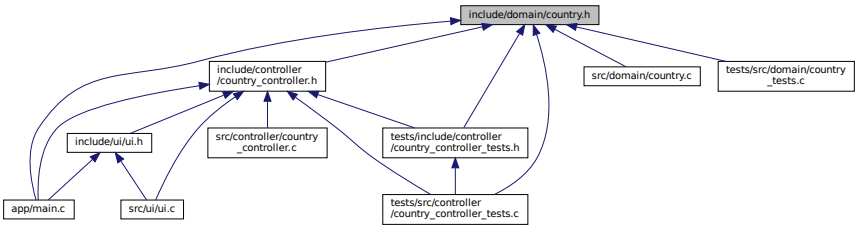
this function undoes an operation

Parameters

p_undo_redo_controller	a pointer to the undo_repository-redo_repository controller
p_error	a pointer to the variable which receives the error code. Can be left NULL.

5.3 include/domain/country.h File Reference

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



Classes

- struct country_t

  This struct defined in order to hold all the necessary information for a country.

Functions

- country_t * create_country (const char *name, const char *continent, int population, int *p_error)

  This creates a country using the necessary data. It stores its name in lowercase.
- void delete_country (country_t *p_country)

  This frees the dynamically allocated memory for a given country.
- void delete_country_data (country_t *p_country)

  this free the data for a country(but not the country itself)

5.3.1 Detailed Description

This file contains the definitions for the country model.

5.3.2 Function Documentation

5.3.2.1 create_country()

```
country_t* create_country (
        const char * name,
        const char * continent,
        int population,
        int * p_error )
```

This creates a country using the necessary data. It stores its name in lowercase.

Parameters

name	A pointer to the start of a char array containing the name of the country. The content of the char array is copied in a new dynamically-allocated char array.
continent	A pointer to the start of a char array containing the name of the continent of the country. The content of the char array is copied in a new dynamically-allocated char array.
population	The population of the country
p_error	This is a pointer to the value which receives the error code for this operation(if any). This can be set to NULL if you do not want the error code.

Returns

a pointer to the new created country

5.3.2.2 delete_country() `void delete_country (`

`country_t * p_country )`

This frees the dynamically allocated memory for a given country.

Parameters

p_country	a pointer to the country for which the memory is freed.

5.3.2.3 delete_country_data() `void delete_country_data (`

`country_t * p_country )`

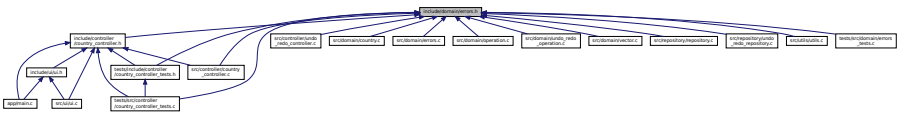
this free the data for a country(but not the country itself)

Parameters

p_country	a pointer to the country for which the memory is freed.

5.4 include/domain/errors.h File Reference

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



Classes

- struct error_t

Functions

- void set_error (int *p_error, int error_code)

  This function sets the error code to a given pointer.

Variables

- const error_t errors [ ]

  Array containing all the corresponding messages for each error code.

5.4.1 Detailed Description

This file contains the corresponding error messages for various error codes.

5.4.2 Function Documentation

5.4.2.1 set_error() `void set_error (`

```
int * p_error,
int error_code )
```

This function sets the error code to a given pointer.

Parameters

p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.
error_code	the error code

5.4.3 Variable Documentation

5.4.3.1 errors `const error_t errors[]`

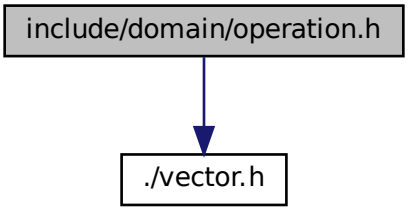
Array containing all the corresponding messages for each error code.

Array containing all the corresponding messages for each error code.

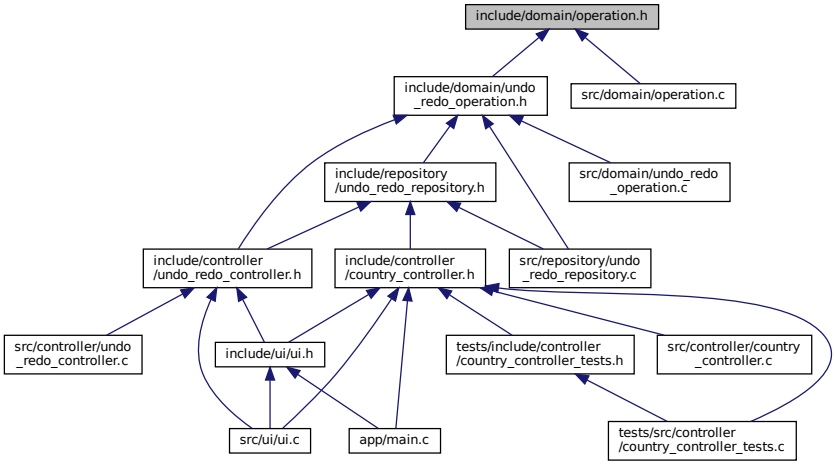
5.5 include/domain/operation.h File Reference

```
#include "./vector.h"
```

Include dependency graph for operation.h:



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



Classes

- struct operation_t

  This struct contains the model for the operation_t struct.

Functions

- operation_t ∗ create_operation (void(∗function)(vector_t ∗args, int ∗p_error), vector_t ∗args, void(∗free_↩ args_data)(vector_t ∗args), int ∗p_error)

  this function creates an operation instance
- void delete_operation (operation_t ∗p_operation)

  this function frees the memory for an operation
- void delete_operation_data (operation_t ∗p_operation)

  this function frees the memory allocated for an operation, but not the operation itself
- void apply (operation_t ∗p_operation, int ∗p_error)

  applies the operation(runs the function with given args).

5.5.1 Detailed Description

this file containg the model for a struct that can perform a single operation(a 'from_args' function call with parameters)

5.5.2 Function Documentation

5.5.2.1 apply()

```
void apply (
        operation_t * p_operation,
        int * p_error )
```

applies the operation(runs the function with given args).

Parameters

p_operation	a pointer to the operation
p_error	a pointer to the variable which receives the error. Can be left NULL.

5.5.2.2 create_operation()

```
operation_t* create_operation (
        void(*)(vector_t *args, int *p_error) function,
        vector_t * args,
        void(*)(vector_t *args) free_args_data,
        int * p_error )
```

this function creates an operation instance

Parameters

function	the function which is to be applied
args	the args of the function
free_args_data	the functions which frees the data for given args. Can be left NULL if there is no need for one.
p_error	the pointer to the variable that receives the error. Can be left NULL

Returns

a pointer to the new operation

5.5.2.3 delete_operation()

```
void delete_operation (
        operation_t * p_operation )
```

this function frees the memory for an operation

Parameters

p_operation	a pointer to the operation to be freed

5.5.2.4 delete_operation_data() `void delete_operation_data ( operation_t * p_operation )`

this function frees the memory allocated for an operation, but not the operation itself

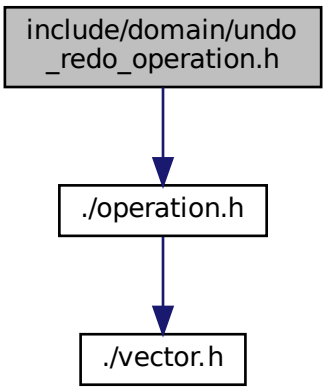
Parameters

p_operation	a pointer to the operation

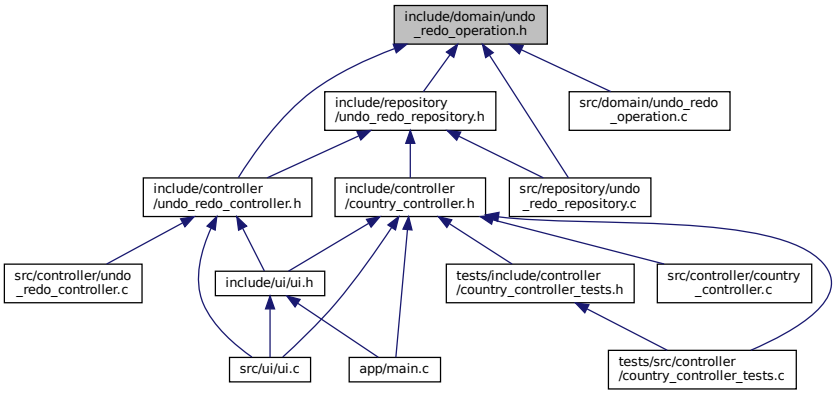
5.6 include/domain/undo_redo_operation.h File Reference

```
#include "./operation.h"
```

Include dependency graph for undo_redo_operation.h:



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



Classes

- struct undo_redo_operation_t

  this is the model for an undo_operation-redo_operation operation

Functions

- undo_redo_operation_t * create_undo_redo_operation (operation_t *p_undo, operation_t *p_redo, int *p_error)

  this function creates an undo_operation-redo_operation operation
- void delete_undo_redo_operation (undo_redo_operation_t *p_operation)

  this function deletes the memory allocated for an undo_operation-redo_operation operation
- void delete_undo_redo_operation_data (undo_redo_operation_t *p_operation)

  this function deletes the memory allocated for an undo_operation-redo_operation operation's data, but not the operation itself
- void undo_operation (undo_redo_operation_t *p_operation, int *p_error)

  this function applies the undo_operation operation of a given operation
- void redo_operation (undo_redo_operation_t *p_operation, int *p_error)

  this function applies the redo_operation operation of a given operation

5.6.1 Detailed Description

this file contains the model for an undo_redo operation. This operation consists of 2 operations, one which can undo_operation and one which can redo_operation.

5.6.2 Function Documentation

5.6.2.1 create_undo_redo_operation()

```
undo_redo_operation_t* create_undo_redo_operation (
        operation_t * p_undo,
        operation_t * p_redo,
        int * p_error )
```

this function creates an undo_operation-redo_operation operation

Parameters

p_undo	a pointer to the undo_operation operation
p_redo	a pointer to the redo_operation operation
p_error	a pointer to the variable which receives the error. Can be left NULL.

Returns

a pointer to the undo_operation-redo_operation operation

5.6.2.2 delete_undo_redo_operation()

```
void delete_undo_redo_operation (
        undo_redo_operation_t * p_operation )
```

this function deletes the memory allocated for an undo_operation-redo_operation operation

Parameters

p_operation	a pointer to the operation

5.6.2.3 delete_undo_redo_operation_data()

```
void delete_undo_redo_operation_data (
        undo_redo_operation_t * p_operation )
```

this function deletes the memory allocated for an undo_operation-redo_operation operation's data, but not the operation itself

Parameters

p_operation	

5.6.2.4 redo_operation()

```
void redo_operation (
        undo_redo_operation_t * p_operation,
        int * p_error )
```

this function applies the redo_operation operation of a given operation

Parameters

p_operation	the pointer to the undo_operation-redo_operation operation
p_error	a pointer to the variable which receives the error. Can be left NULL.

5.6.2.5 undo_operation()

```
void undo_operation (
        undo_redo_operation_t * p_operation,
        int * p_error )
```

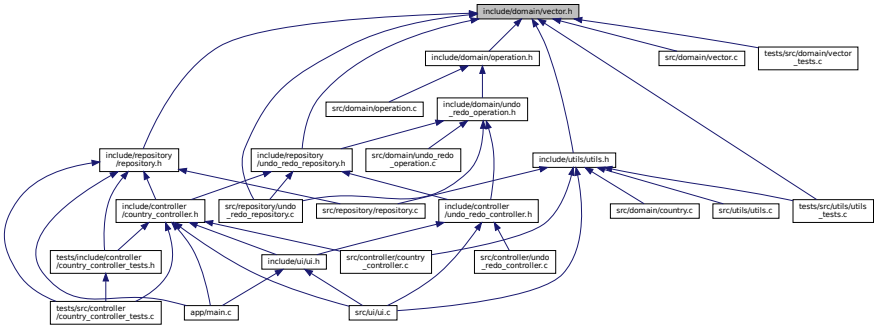
this function applies the undo_operation operation of a given operation

Parameters

p_operation	the pointer to the undo_operation-redo_operation operation
p_error	a pointer to the variable which receives the error. Can be left NULL.

5.7 include/domain/vector.h File Reference

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



Classes

- struct vector_t

  Implements a dynamic array. It is generic, so you can hold any type in it.

Functions

- vector_t ∗ create_vector (int element_size, int initial_size, void(∗free_elem_data)(), int ∗p_error)

  This function creates a pointer to a new vector_t object.
- void delete_vector (vector_t ∗p_v)

  This function frees the memory for a vector_t object.
- void resize (vector_t ∗p_v, int size, int ∗p_error)

  This function resizes the current vector. It also changes its capacity.
- void change_capacity_vector (vector_t ∗p_v, int capacity, int ∗p_error)

  This function changes the capacity of the current vector.
- void push_back (vector_t ∗p_v, const void ∗p_element, int ∗p_error)

  This function resizes a given vector. It changes both size, and capacity. Deletes the any elements which do not fit in the new size. Any newly created elements are not initialized to any default value.
- void pop_back (vector_t ∗p_v, int ∗p_error)

  This function removes the last element of the vector. It changes only size.
- void ∗ get_position_vector (const vector_t ∗p_v, int pos, int ∗p_error)

  gets a pointer to the element from specified position in the given vector
- void set_position_vector (vector_t ∗p_v, int pos, const void ∗p_element, int ∗p_error)

  sets an element on a given position in the vector to a given element

5.7.1 Detailed Description

This file contains the model for a dynamic array. This array is generic, so it holds a void* pointer to a large buffer.

5.7.2 Function Documentation

5.7.2.1 change_capacity_vector()

```
void change_capacity_vector (
        vector_t * p_v,
        int capacity,
        int * p_error )
```

This function changes the capacity of the current vector.

Parameters

p_v	the pointer to the vector to be resized
capacity	the new size of the vector
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL

5.7.2.2 create_vector()

```
vector_t* create_vector (
        int element_size,
        int initial_size,
        void(*)() free_elem_data,
        int * p_error )
```

This function creates a pointer to a new vector_t object.

Parameters

element_size	the size of a vector element(int bytes)
initial_size	the initial number of elements of the vector
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL
free_elem_data	a pointer to a function responsible for freeing data for a given element(but not the element itself). Can be NULL if no freeing should be applied.

Returns

a pointer to the newly created vector

5.7.2.3 delete_vector()

```
void delete_vector (
        vector_t * p_v )
```

This function frees the memory for a vector_t object.

Parameters

p_v	the pointer to the vector to be freed

5.7.2.4 get_position_vector()

```
void* get_position_vector (
        const vector_t * p_v,
        int pos,
        int * p_error )
```

gets a pointer to the element from specified position in the given vector

Parameters

p_v	pointer to the vector that is accessed
pos	the position that is wanted
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.7.2.5 pop_back()

```
void pop_back (
        vector_t * p_v,
        int * p_error )
```

This function removes the last element of the vector. It changes only size.

Parameters

p_v	a pointer to the vector from which the last element is removed
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.7.2.6 push_back()

```
void push_back (
        vector_t * p_v,
        const void * p_element,
        int * p_error )
```

This function resizes a given vector. It changes both size, and capacity. Deletes the any elements which do not fit in the new size. Any newly created elements are not initialized to any default value.

Parameters

p_v	the pointer to the vector on which to push back the element.
p_element	a pointer to the element to be pushed to the back of the vector.
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.7.2.7 resize() `void resize (`

```
vector_t * p_v,
int size,
int * p_error )
```

This function resizes the current vector. It also changes its capacity.

Parameters

p_v	the pointer to the vector to be resized
size	the new size of the vector
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL

5.7.2.8 set_position_vector() `void set_position_vector (`

```
vector_t * p_v,
int pos,
const void * p_element,
int * p_error )
```

sets an element on a given position in the vector to a given element

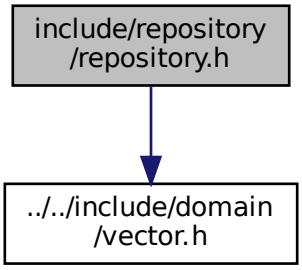
Parameters

p_v	pointer to the vector that is modified
pos	the position that will be modified
p_element	pointer to the elemenet with which the position is updated
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

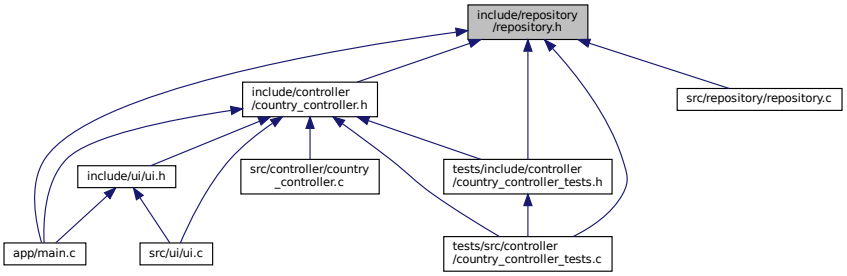
5.8 include/repository/repository.h File Reference

```
#include "../../include/domain/vector.h"
```

Include dependency graph for repository.h:



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



Classes

- struct repository_t

  This is the model for a repository instance.

Functions

- repository_t * create_repository (int element_size, void(*free_element_data)(void *), int *p_error)

  creates a repository object and returns a pointer to it
- void delete_repository (repository_t *p_repository)

  frees the memory allocated for a given repository
- void add_repository (repository_t *p_repository, void *p_element, int *p_error)

  adds an element to a given repository
- vector_t * filter_repository (repository_t *p_repository, int(*p_filter_function)(void *repository_element, void **args), void **filter_function_args, int *p_error)

  returns a list of the indexes of elements which match a given filter function.
- void update_repository (repository_t *p_repository, int position, void *p_new_element, int *p_error)

  updates an element from the repository
- void remove_repository (repository_t *p_repository, int position, int *p_error)

removes an element from the repository

- void * get_position_repository (repository_t *p_repository, int position, int *p_error)

  gets an element from a specified position

- int get_repository_size (repository_t *p_repository)

  returns the number of elements the repository currently has

- int get_repository_element_size (repository_t *p_repository)

  returns the size(in bytes) of a repository element

5.8.1 Detailed Description

this file contains the model for a general list-based repository

5.8.2 Function Documentation

5.8.2.1 add_repository()

```
void add_repository (
        repository_t * p_repository,
        void * p_element,
        int * p_error )
```

adds an element to a given repository

Parameters

p_repository	a pointer to the repository
p_element	a pointer to the element which is adds to the repository
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.8.2.2 create_repository()

```
repository_t* create_repository (
        int element_size,
        void(*)(void *) free_element_data,
        int * p_error )
```

creates a repository object and returns a pointer to it

Parameters

element_size	the size of a repository element(in bytes)
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.
free_element_data	a pointer to a function that frees the data for an element(but not the element itself)

Returns

pointer to the newly created repository

5.8.2.3 delete_repository()

```
void delete_repository (
        repository_t * p_repository )
```

frees the memory allocated for a given repository

Parameters

p_repository	a pointer to the repository

5.8.2.4 filter_repository()

```
vector_t* filter_repository (
        repository_t * p_repository,
        int(*)(void *repository_element, void **args) p_filter_function,
        void ** filter_function_args,
        int * p_error )
```

returns a list of the indexes of elements which match a given filter function.

Parameters

p_repository	a pointer to the repository
p_filter_function	a filter function which decides which elements are returned. If the function returns 1, the element is added to the answer. The filter function should have 2 arguments, the first one should be the pointer to the repository element, the second one should be any custom function arguments.
filter_function_args	any custom filter function arguments
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

Returns

pointer to a vector that contains the indexes of the elements which are matching a given filter function.

5.8.2.5 get_position_repository()

```
void* get_position_repository (
        repository_t * p_repository,
        int position,
        int * p_error )
```

gets an element from a specified position

Parameters

p_repository	a pointer to the repository
position	the position wanted

Returns

a pointer to the element

5.8.2.6 get_repository_element_size() `int get_repository_element_size (`

```
repository_t * p_repository )
```

returns the size(in bytes) of a repository element

Parameters

p_repository	a pointer to the repository

Returns

the size of a repository element

5.8.2.7 get_repository_size() `int get_repository_size (`

```
repository_t * p_repository )
```

returns the number of elements the repository currently has

Parameters

p_repository	a pointer to the repository

Returns

the number of elements the repository contains

5.8.2.8 remove_repository() `void remove_repository (`

```
repository_t * p_repository,
int position,
int * p_error )
```

removes an element from the repository

Parameters

p_repository	a pointer to the repository
position	the position of the element which is to be removed
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.8.2.9 update_repository()

```
void update_repository (
        repository_t * p_repository,
        int position,
        void * p_new_element,
        int * p_error )
```

updates an element from the repository

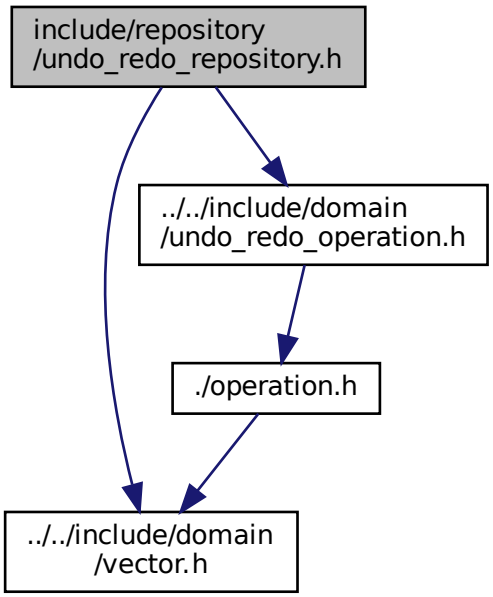
Parameters

p_repository	a pointer to the repository
position	the position of the element which is to be updated
p_new_element	a pointer to the new value of the updated element
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

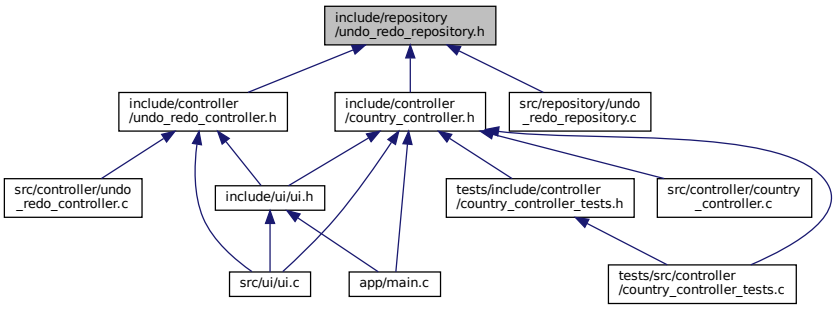
5.9 include/repository/undo_redo_repository.h File Reference

```
#include "../../include/domain/vector.h"
#include "../../include/domain/undo_redo_operation.h"
```

Include dependency graph for undo_redo_repository.h:



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



Classes

- struct undo_redo_repository_t

  This is the model for an undo_redo_repository_t instance.

Functions

- undo_redo_repository_t * create_undo_redo_repository (int *p_error)

  this function creates an undo redo repository
- void delete_undo_redo_repository (undo_redo_repository_t *p_undo_redo_repository)

  this function frees the memory for a given undo redo repository
- void add_undo_repository (undo_redo_repository_t *p_undo_redo_repository, undo_redo_operation_t *p↩ _operation, int *p_error)

  add an operation to the repository
- void undo_repository (undo_redo_repository_t *p_undo_redo_repository, int *p_error)

  this function undoes an operation
- void redo_repository (undo_redo_repository_t *p_undo_redo_repository, int *p_error)

  this function redoes an operation

5.9.1 Detailed Description

this file contains the model for a repository responsible for managing undo-redo operations

5.9.2 Function Documentation

5.9.2.1 add_undo_repository()

```
void add_undo_repository (
        undo_redo_repository_t * p_undo_redo_repository,
        undo_redo_operation_t * p_operation,
        int * p_error )
```

add an operation to the repository

Parameters

p_undo_redo_repository	a pointer to the repository
p_operation	a pointer to the undo redo operation
p_error	a pointer to the variable that receives the error code. Can be left NULL.

5.9.2.2 create_undo_redo_repository()

```
undo_redo_repository_t* create_undo_redo_repository (
        int * p_error )
```

this function creates an undo redo repository

Parameters

p_error	a pointer to the variable that receives the error code. Can be left NULL.

Returns

a pointer to the undo redo repository created

5.9.2.3 delete_undo_redo_repository()

```
void delete_undo_redo_repository (
        undo_redo_repository_t * p_undo_redo_repository )
```

this function frees the memory for a given undo redo repository

Parameters

p_undo_redo_repository	a pointer to the undo redo repository

5.9.2.4 redo_repository()

```
void redo_repository (
        undo_redo_repository_t * p_undo_redo_repository,
        int * p_error )
```

this function redoes an operation

Parameters

p_undo_redo_repository	
p_error	a pointer to the variable that receives the error code. Can be left NULL.

5.9.2.5 undo_repository()

```
void undo_repository (
        undo_redo_repository_t * p_undo_redo_repository,
        int * p_error )
```

this function undoes an operation

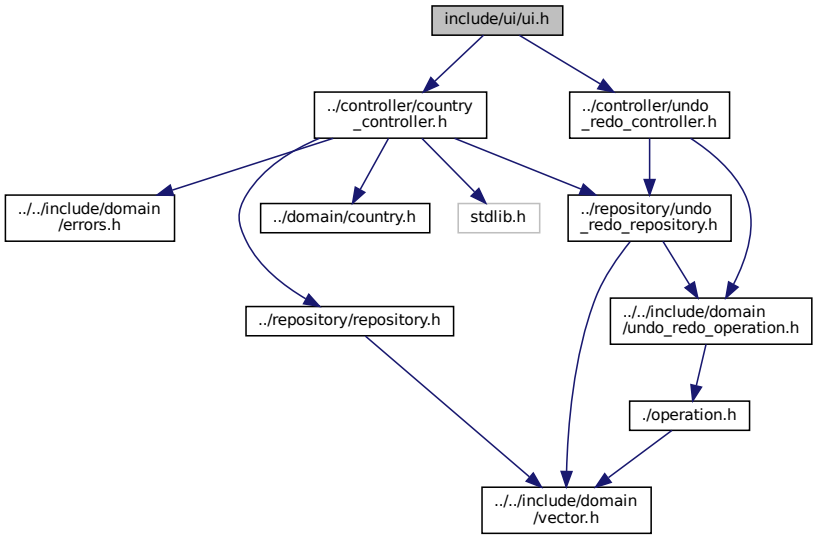
Parameters

p_undo_redo_repository	a pointer to the undo redo repository
p_error	a pointer to the variable that receives the error code. Can be left NULL.

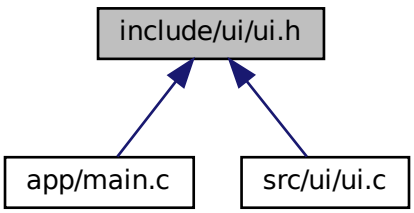
5.10 include/ui/ui.h File Reference

```
#include "../controller/country_controller.h"
#include "../controller/undo_redo_controller.h"
```

Include dependency graph for ui.h:



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



Classes

- struct ui_t

  this is a model for the ui_t structure

Functions

- ui_t * create_ui (country_controller_t *p_controller, undo_redo_controller_t *p_undo_redo_controller, int *p_error)

  creates ui object
- void delete_ui (ui_t *p_ui)
- void run (ui_t *p_ui)

  the main loop of the ui

5.10.1 Detailed Description

this file contains the model for an ui structure

5.10.2 Function Documentation

5.10.2.1 create_ui()

```
ui_t* create_ui (
        country_controller_t * p_controller,
        undo_redo_controller_t * p_undo_redo_controller,
        int * p_error )
```

creates ui object

Parameters

p_controller	pointer to the controller
p_error	pointer to the variable receiving the error code. Can be left NULL.

5.10.2.2 delete_ui()

```
void delete_ui (
        ui_t * p_ui )
```

frees the memory of a ui_t object

Parameters

p_ui	pointer to the ui object

5.10.2.3 run() `void run ( ui_t * p_ui )`

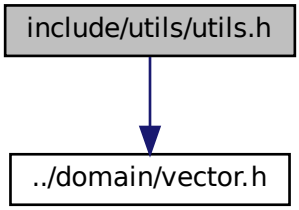
the main loop of the ui

Parameters

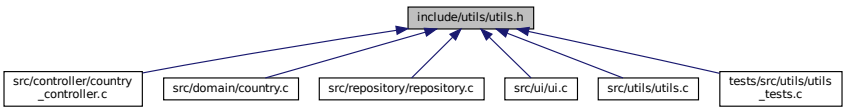
p_ui	the pointer to the ui

5.11 include/utils/utils.h File Reference

`#include "../domain/vector.h"`
Include dependency graph for utils.h:



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



Functions

- char * get_lowercase (const char *p_s, int *p_error)

  gets a pointer to a new char string having the lowercase version of a given string
- void sort (vector_t *p_v, int l, int r, int(*cmp)(const void **a, const void **b))

  this function sorts a given vector using the compare function on the range [l,r)
- void swap (void *p_a, void *p_b, int cnt_bytes)

  swaps the next cnt_bytes from a and b
- vector_t * split_string (char *p_s, char delim, int *p_error)

  this function returns a vector structure containing pointers to all tokens in a given string.

5.11.1 Detailed Description

contains the definitions of various general purpose functions.

5.11.2 Function Documentation

5.11.2.1 get_lowercase()

```
char* get_lowercase (
        const char * p_s,
        int * p_error )
```

gets a pointer to a new char string having the lowercase version of a given string

Parameters

p_s	the pointer to the string
p_error	a pointer to the variable that gets the error

Returns

pointer to the new string

5.11.2.2 sort()

```
void sort (
        vector_t * p_v,
        int l,
        int r,
        int(*)(const void **a, const void **b) cmp )
```

this function sorts a given vector using the compare function on the range [l,r)

Parameters

p_v	a pointer to the vector to be sorted
l	the left index
r	the right index
cmp	the compare index which returns 1 if a $<$ b

5.11.2.3 split_string()

```
vector_t* split_string (
        char * p_s,
        char delim,
        int * p_error )
```

this function returns a vector structure containing pointers to all tokens in a given string.

Parameters

p_s	a pointer to the string being processed
delim	a character which is the 'deliminator'
p_error	a pointer to the variable that gets the error

Returns

a pointer to a vector containing pointers to new strings containing the arguments

5.11.2.4 swap()

```
void swap (
        void * p_a,
        void * p_b,
        int cnt_bytes )
```

swaps the next cnt_bytes from a and b

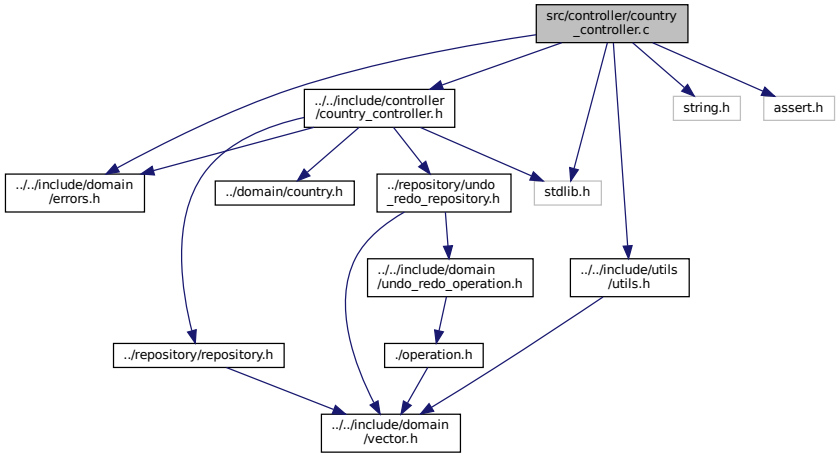
Parameters

p_a	pointer to the first address
p_b	pointer to the second address
cnt_bytes	the number of bytes to swap

5.12 src/controller/country_controller.c File Reference

```
#include "../../include/domain/errors.h"
#include "../../include/controller/country_controller.h"
#include "../../include/utils/utils.h"
#include <string.h>
#include <stdlib.h>
#include <assert.h>
```

Include dependency graph for country_controller.c:



Functions

- country_controller_t * create_country_controller (repository_t *p_repository, undo_redo_repository_t *p_↩ undo_redo_repository, int *p_error)

  creates a country controller and returns the pointer to it
- void delete_country_controller (country_controller_t *p_controller)

  frees the memory allocated for a country_controller
- int filter_by_exact_name (void *p_country, void **args)

  filters countries by checking their exact name
- int get_index_from_name (country_controller_t *p_controller, const char *p_name, int *p_error)

  returns the index of a country by name in repository.
- void add_country (country_controller_t *p_controller, const country_t *p_country, int *p_error)

  adds a country to the controller.
- void remove_country (country_controller_t *p_controller, const int index, int *p_error)

  removes a country from the controller.
- void update_country (country_controller_t *p_controller, const int index, const country_t *p_new_country, int *p_error)

  updates a country from the controller.
- void migrate_country (country_controller_t *p_controller, const int src_index, const int dst_index, const int population, int *p_error)

  migrates population from a country to another
- void free_add_country_args (vector_t *args)
- void free_remove_country_args (vector_t *args)

  this function frees the memory allocated for the arguments a remove from args operation
- void free_update_country_args (vector_t *args)

  this function frees the memory allocated for arguments for a update from args operation
- void **free_migrate_country_args** (vector_t *args)
- void add_country_ui (country_controller_t *p_controller, const char *p_country_name, const char *p_↩ country_continent, const int population, int make_undo_redo, int *p_error)

  Adds a country to the controller. This function should be used by UI only.
- void add_country_from_args (vector_t *args, int *p_error)

Adds a country using arguments contained in a single parameter. This is intended to be used only for undo_↩ operation/redo_operation purposes.

- void remove_country_ui (country_controller_t *p_controller, const char *p_country_name, int make_undo↩ _redo, int *p_error)

  Removes a country based on its name. This is intended to be used by UI only.

- void remove_country_from_args (vector_t *args, int *p_error)

  Removes a country using arguments contained in a single parameter. This is intended to be used only for undo_↩ operation/redo_operation purposes.

- void update_country_ui (country_controller_t *p_controller, const char *p_country_name, const int new_↩ population, int make_undo_redo_operation, int *p_error)

  Updates a country based on its name. This is intended to be used by UI only.

- void update_country_from_args (vector_t *args, int *p_error)

  Updates a country using arguments contained in a single parameter. This is intended to be used for undo_↩ operation/redo_operation purposes only.

- void migrate_ui (country_controller_t *p_controller, const char *p_country_source_name, const char *p_↩ country_destination_name, const int population, int make_undo_redo_operation, int *p_error)

  migrates population from a country to another. This is intended to be used by UI only.

- void migrate_from_args (vector_t *args, int *p_error)

  migrates population from a country to another. This is intended to be used for undo_operation/redo_operation purposes only.

- vector_t * get_p_countries_from_p_indexes (const country_controller_t *p_controller, vector_t *p_indexes, int *p_error)

  returns a vector containing the pointers to the countries corresponding to the indexes from a given vector

- int filter_contains_string (country_t *p_country, void **args)

  returns 1 if the given country contains a given string in its name and 0 otherwise

- vector_t * get_countries_containing_string (country_controller_t *p_controller, const char *p_name, int *p↩ _error)

  This function returns all countries which have given string as substring in their name.

- int filter_at_least (country_t *p_country, void **args)

  returns 1 if the given country has more population than a given amount

- vector_t * get_countries_with_at_least (country_controller_t *p_controller, const int population, int *p_error)

  This function returns all countries which have at least a specified population.

- int filter_matches_string_continent_and_population (country_t *p_country, void **args)

  returns 1 if the given country's continent matches a string, and 0 otherwise

- int cmp_ascending (const country_t **a, const country_t **b)

  comparator for sorting countries ascendingly according to population

- int cmp_descending (const country_t **a, const country_t **b)

  comparator for sorting countries descendingly according to population

- vector_t * get_countries_from_continent (country_controller_t *p_controller, const char *p_continent_name, const int population, const char *p_sort_order, int *p_error)

  This functions returns all countries from a continent with at least a specified population.

5.12.1 Detailed Description

This file contains the implementation for the functions from country_controller.h

5.12.2 Function Documentation

5.12.2.1 add_country()

```
void add_country (
        country_controller_t * p_controller,
        const country_t * p_country,
        int * p_error )
```

adds a country to the controller.

Parameters

p_controller	a pointer to the country controller
p_country	a pointer to the country to be added.
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

5.12.2.2 add_country_from_args()

```
void add_country_from_args (
        vector_t * args,
        int * p_error )
```

Adds a country using arguments contained in a single parameter. This is intended to be used only for undo_↩ operation/redo_operation purposes.

Parameters

args	a pointer to the vector of arguments this function would normally use
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

5.12.2.3 add_country_ui()

```
void add_country_ui (
        country_controller_t * p_controller,
        const char * p_country_name,
        const char * p_country_continent,
        const int population,
        int make_undo_redo,
        int * p_error )
```

Adds a country to the controller. This function should be used by UI only.

Parameters

p_controller	a pointer to the country controller
p_country_name	a pointer to the country name string
p_country_continent	a pointer to the country continent string
population	the population of the country
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

this frees the memory for the operation because its data is stored in the repository

5.12.2.4 cmp_ascending() `int cmp_ascending (`

```
const country_t ** a,
const country_t ** b )
```

comparator for sorting countries ascendingly according to population

Parameters

a	pointer to the first country
b	pointer to the second country

Returns

return 0 if a has more population than b, 1 otherwise

5.12.2.5 cmp_descending() `int cmp_descending (`

```
const country_t ** a,
const country_t ** b )
```

comparator for sorting countries descendingly according to population

Parameters

a	pointer to a pointer to the first country
b	pointer to a pointer to the second country

Returns

return 0 if a has more population than b, 1 otherwise

5.12.2.6 create_country_controller() `country_controller_t* create_country_controller (`

```
repository_t * p_repository,
undo_redo_repository_t * p_undo_redo_repository,
int * p_error )
```

creates a country controller and returns the pointer to it

Parameters

p_controller	a pointer to the repository containing the countries.
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

Returns

pointer to the country controller

5.12.2.7 delete_country_controller()

```
void delete_country_controller (
        country_controller_t * p_controller )
```

frees the memory allocated for a country_controller

Parameters

p_controller	

5.12.2.8 filter_at_least()

```
int filter_at_least (
        country_t * p_country,
        void ** args )
```

returns 1 if the given country has more population than a given amount

Parameters

p_country	pointer to the country
args	a pointer to an array of pointers containing a single element which points to an int which is the population to be checked against

Returns

1 if the string is included, 0 otherwise

5.12.2.9 filter_by_exact_name()

```
int filter_by_exact_name (
        void * p_country,
        void ** args )
```

filters countries by checking their exact name

Parameters

p_country	pointer to a country
args	void** pointer containing only one element which is a pointer to the name to be checked against

Returns

1 if they are the same, 0 otherwise

5.12.2.10 filter_contains_string() `int filter_contains_string (`

```
country_t * p_country,
void ** args )
```

returns 1 if the given country contains a given string in its name and 0 otherwise

Parameters

p_country	pointer to the country
args	a pointer to an array of pointers containing a single element which points to the string which is to be checked against

Returns

1 if the string is included, 0 otherwise

5.12.2.11 filter_matches_string_continent_and_population() `int filter_matches_string_continent_↩ and_population (`

```
country_t * p_country,
void ** args )
```

returns 1 if the given country's continent matches a string, and 0 otherwise

Parameters

p_country	pointer to the country
args	a pointer to an array of pointers containing a single element which points to the string which is to be checked against

Returns

1 if the string is included, 0 otherwise

5.12.2.12 free_add_country_args() `void free_add_country_args (`

```
vector_t * args )
```

frees the memory allocated for the arguments of an add operation

Parameters

args	a pointer to the vector containing the arguments

5.12.2.13 free_remove_country_args() `void free_remove_country_args (`

```
vector_t * args )
```

this function frees the memory allocated for the arguments a remove from args operation

Parameters

args	pointer to the arguments

5.12.2.14 free_update_country_args()

```
void free_update_country_args (
    vector_t * args )
```

this function frees the memory allocated for arguments for a update from args operation

Parameters

args	pointer to the arguments

5.12.2.15 get_countries_containing_string()

```
vector_t* get_countries_containing_string (
    country_controller_t * p_controller,
    const char * p_name,
    int * p_error )
```

This function returns all countries which have given string as substring in their name.

Parameters

p_controller	a pointer to the country controller
p_name	a pointer to the string containing the name
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

Returns

a pointer to a vector containing pointer to the valid countries.

5.12.2.16 get_countries_from_continent()

```
vector_t* get_countries_from_continent (
    country_controller_t * p_controller,
    const char * p_continent_name,
    const int population,
    const char * p_sort_order,
    int * p_error )
```

This functions returns all countries from a continent with at least a specified population.

Parameters

p_controller	a pointer to the country controller
p_continent_name	a pointer to the string containing the continent name
population	the minimum population a country should have
p_sort_order	a pointer to the string containing the order
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

Returns

a pointer to a vector containing pointer to all valid countries.

5.12.2.17 get_countries_with_at_least()

```
vector_t* get_countries_with_at_least (
    country_controller_t * p_controller,
    const int population,
    int * p_error )
```

This function returns all countries which have at least a specified population.

Parameters

p_controller	a pointer to the country controller.
population	the minimum population a country should have.
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

Returns

a pointer to a vector containing pointers to all valid countries.

5.12.2.18 get_index_from_name()

```
int get_index_from_name (
    country_controller_t * p_controller,
    const char * p_name,
    int * p_error )
```

returns the index of a country by name in repository.

Parameters

p_controller	a pointer to the country controller
p_name	a pointer to the string containing the name of the country
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

Returns

the country index if it exists, and -1 otherwise(but it also sets p_error to 11)

5.12.2.19 get_p_countries_from_p_indexes()

```
vector_t* get_p_countries_from_p_indexes (
    const country_controller_t * p_controller,
    vector_t * p_indexes,
    int * p_error )
```

returns a vector containing the pointers to the countries corresponding to the indexes from a given vector

Parameters

p_controller	pointer to the country controller
p_indexes	pointer to the vector containing the indexes
p_error	pointer to the variable that receives the error variable. Can be left NULL

Returns

pointer to a vector containing pointers to countries.

5.12.2.20 migrate_country()

```
void migrate_country (
    country_controller_t * p_controller,
    const int src_index,
    const int dst_index,
    const int population,
    int * p_error )
```

migrates population from a country to another

Parameters

p_controller	a pointer to the country controller
src_index	the index of the source country
dst_index	the index of the destination country
population	the population that is migrated
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

5.12.2.21 migrate_from_args()

```
void migrate_from_args (
    vector_t * args,
    int * p_error )
```

migrates population from a country to another. This is intended to be used for undo_operation/redo_operation purposes only.

Parameters

args	a pointer to the vector of arguments this function would normally use
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

5.12.2.22 migrate_ui()

```
void migrate_ui (
    country_controller_t * p_controller,
    const char * p_country_source_name,
    const char * p_country_destination_name,
    const int population,
    int make_undo_redo,
    int * p_error )
```

migrates population from a country to another. This is intended to be used by UI only.

Parameters

p_controller	a pointer to the country controller
p_country_source_name	a pointer to the string containing the source country name
p_country_destination_name	a pointer to the string containing the destination country name
population	the population that migrates
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

this frees the memory for the operation because its data is stored in the repository

5.12.2.23 remove_country()

```
void remove_country (
    country_controller_t * p_controller,
    const int index,
    int * p_error )
```

removes a country from the controller.

Parameters

p_controller	a pointer to the country controller
index	the index of the country which is removed
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

5.12.2.24 remove_country_from_args()

```
void remove_country_from_args (
    vector_t * args,
    int * p_error )
```

Removes a country using arguments contained in a single parameter. This is intended to be used only for undo_↩ operation/redo_operation purposes.

Parameters

args	a pointer to the vector of arguments this function would normally use
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

5.12.2.25 remove_country_ui()

```
void remove_country_ui (
    country_controller_t * p_controller,
    const char * p_country_name,
    int make_undo_redo,
    int * p_error )
```

Removes a country based on its name. This is intended to be used by UI only.

Parameters

p_controller	a pointer to the country controller
p_country_name	a pointer to the string containing the country name
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

this frees the memory for the operation because its data is stored in the repository

5.12.2.26 update_country()

```
void update_country (
    country_controller_t * p_controller,
    const int index,
    const country_t * p_new_country,
    int * p_error )
```

updates a country from the controller.

Parameters

p_controller	a pointer to the country controller
index	the index of the country which is removed
p_new_country	a pointer to a country object containing the new data
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

5.12.2.27 update_country_from_args()

```
void update_country_from_args (
    vector_t * args,
    int * p_error )
```

Updates a country using arguments contained in a single parameter. This is intended to be used for undo_↩ operation/redo_operation purposes only.

Parameters

args	a pointer to the vector of arguments this function would normally use
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

5.12.2.28 update_country_ui()

```
void update_country_ui (
    country_controller_t * p_controller,
    const char * p_country_name,
    const int new_population,
    int make_undo_redo,
    int * p_error )
```

Updates a country based on its name. This is intended to be used by UI only.

Parameters

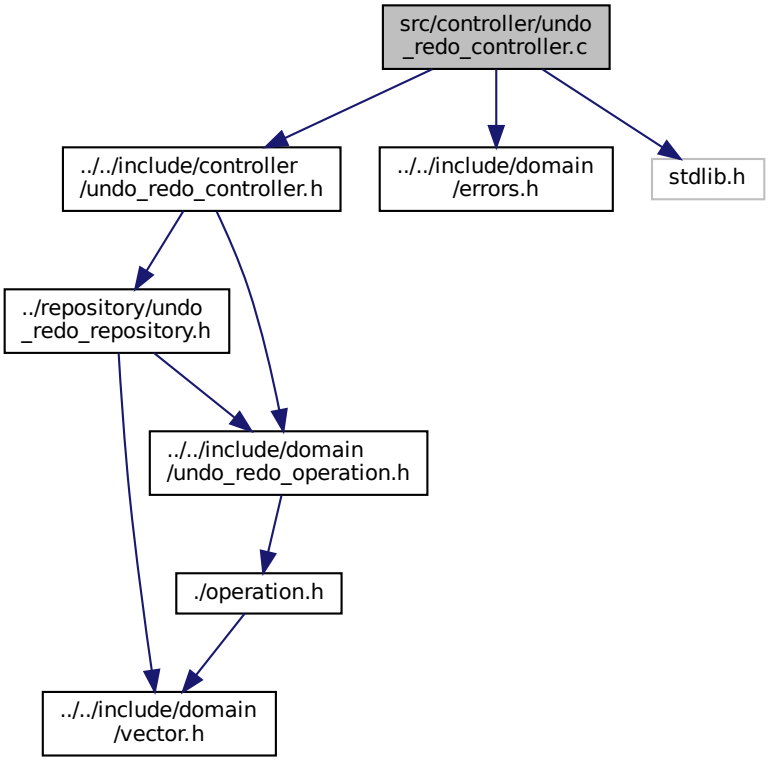
p_controller	a pointer to the country controller
p_country_name	a pointer to the string containing the country name
new_population	the new population of the country.
p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.

this frees the memory for the operation because its data is stored in the repository

5.13 src/controller/undo_redo_controller.c File Reference

```
#include "../../include/controller/undo_redo_controller.h"
#include "../../include/domain/errors.h"
#include <stdlib.h>
```

Include dependency graph for undo_redo_controller.c:



Functions

- undo_redo_controller_t * create_undo_redo_controller (undo_redo_repository_t *p_repository, int *p_error)

  this function creates an undo_repository-redo_repository controller instance
- void delete_undo_redo_controller (undo_redo_controller_t *p_undo_redo_controller)

  this function deletes an undo_repository redo_repository controller
- void undo (undo_redo_controller_t *p_undo_redo_controller, int *p_error)

  this function undoes an operation
- void redo (undo_redo_controller_t *p_undo_redo_controller, int *p_error)

  this function redoes an operation

5.13.1 Detailed Description

this file containing the implementation for implementing the undo_operation-redo_operation controller This controller is responsible for successfully applying undo_operation and redo_operation functions

5.13.2 Function Documentation

5.13.2.1 create_undo_redo_controller()

```
undo_redo_controller_t* create_undo_redo_controller (
        undo_redo_repository_t * p_repository,
        int * p_error )
```

this function creates an undo_repository-redo_repository controller instance

Parameters

p_repository	a pointer to the repository which should be used for storing operations
p_error	a pointer to the variable receiving the error code. Can be left NULL.

Returns

a pointer to an undo_repository-redo_repository controller instance

5.13.2.2 delete_undo_redo_controller()

```
void delete_undo_redo_controller (
        undo_redo_controller_t * p_undo_redo_controller )
```

this function deletes an undo_repository redo_repository controller

Parameters

p_undo_redo_controller	a pointer to the undo_repository redo_repository controller

5.13.2.3 redo()

```
void redo (
        undo_redo_controller_t * p_undo_redo_controller,
        int * p_error )
```

this function redoes an operation

Parameters

p_undo_redo_controller	a pointer to the undo_repository-redo_repository controller
p_error	a pointer to the varialbe which receives the error code. Can be left NULL.

5.13.2.4 undo()

```
void undo (
        undo_redo_controller_t * p_undo_redo_controller,
        int * p_error )
```

this function undoes an operation

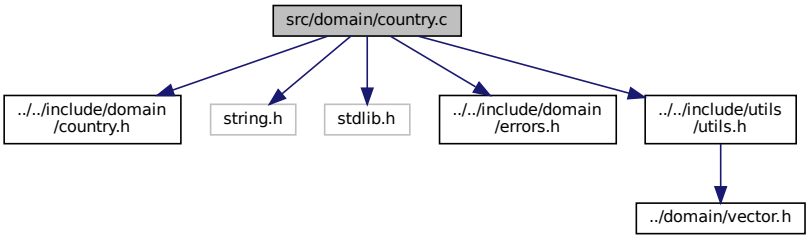
Parameters

p_undo_redo_controller	a pointer to the undo_repository-redo_repository controller
p_error	a pointer to the variable which receives the error code. Can be left NULL.

5.14 src/domain/country.c File Reference

```
#include "../../include/domain/country.h"
#include <string.h>
#include <stdlib.h>
#include "../../include/domain/errors.h"
#include "../../include/utils/utils.h"
```

Include dependency graph for country.c:



Functions

- country_t * create_country (const char *name, const char *continent, int population, int *p_error)

  This creates a country using the necessary data. It stores its name in lowercase.
- void delete_country (country_t *p_country)

  This frees the dynamically allocated memory for a given country.
- void delete_country_data (country_t *p_country)

  this free the data for a country(but not the country itself)

5.14.1 Detailed Description

This file contains the implementation of the defined functions for Country.h

5.14.2 Function Documentation

5.14.2.1 create_country()

```
country_t* create_country (
        const char * name,
        const char * continent,
        int population,
        int * p_error )
```

This creates a country using the necessary data. It stores its name in lowercase.

Parameters

name	A pointer to the start of a char array containing the name of the country. The content of the char array is copied in a new dynamically-allocated char array.
continent	A pointer to the start of a char array containing the name of the continent of the country. The content of the char array is copied in a new dynamically-allocated char array.
population	The population of the country
p_error	This is a pointer to the value which receives the error code for this operation(if any). This can be set to NULL if you do not want the error code.

Returns

a pointer to the new created country

5.14.2.2 delete_country() `void delete_country ( country_t * p_country )`

This frees the dynamically allocated memory for a given country.

Parameters

p_country	a pointer to the country for which the memory is freed.

5.14.2.3 delete_country_data() `void delete_country_data ( country_t * p_country )`

this free the data for a country(but not the country itself)

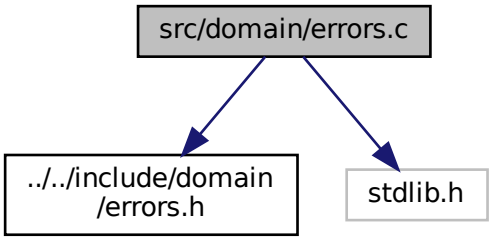
Parameters

p_country	a pointer to the country for which the memory is freed.

5.15 src/domain/errors.c File Reference

```
#include "../../include/domain/errors.h"
#include <stdlib.h>
```

Include dependency graph for errors.c:



Functions

- void set_error (int *p_error, int error_code)

  This function sets the error code to a given pointer.

Variables

- const error_t errors [ ]

  this is a vector containing all error codes

5.15.1 Detailed Description

contains the implementation for errors.h

5.15.2 Function Documentation

5.15.2.1 set_error() `void set_error ( int * p_error, int error_code )`

This function sets the error code to a given pointer.

Parameters

p_error	a pointer to the variable that receives the error code. It can be NULL in case no error is wanted.
error_code	the error code

5.15.3 Variable Documentation

5.15.3.1 errors `const error_t errors[]`

Initial value:

```
= {
{"Success!", 0},
{"Unable to allocate more memory(bad_alloc)", 1},
{"Vector element size invalid", 1},
{"Vector size invalid", 1},
{"Vector new capacity is less than old capacity", 1},
{"A pop was attempted on an empty vector", 1},
{"Attempted to access an index which is not in vector", 1},
{"Invalid index in repository", 1},
{"Country with specified name already exists", 0},
{"Country population cannot be negative", 0},
{"Continent does not exist", 0},
{"Country with specified name doesn't exist", 0},
{"Source country cannot be the same with destination country", 0},
{"Source country population is less than migrated population", 0},
{"New population cannot be negative", 0},
{"Sort order invalid", 0},
{"Invalid address", 1},
{"Command does not follow format", 0},
{"Args has wrong size", 1},
{"Cannot undo any further", 0},
{"Cannot redo any further", 0}
}
```

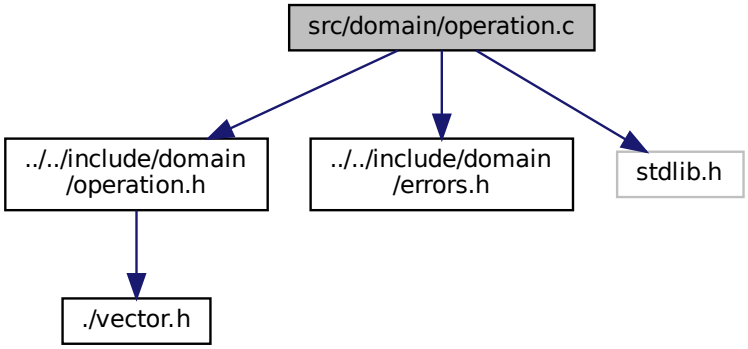
this is a vector containing all error codes

Array containing all the corresponding messages for each error code.

5.16 src/domain/operation.c File Reference

```
#include "../../include/domain/operation.h"
#include "../../include/domain/errors.h"
#include <stdlib.h>
```

Include dependency graph for operation.c:



Functions

- operation_t * create_operation (void(*function)(vector_t *args, int *p_error), vector_t *args, void(*free_↩ args_data)(vector_t *args), int *p_error)

  this function creates an operation instance
- void delete_operation (operation_t *p_operation)

  this function frees the memory for an operation
- void delete_operation_data (operation_t *p_operation)

  this function frees the memory allocated for an operation, but not the operation itself
- void apply (operation_t *p_operation, int *p_error)

  applies the operation(runs the function with given args).

5.16.1 Detailed Description

this file contains the implementation of the operation structure and its methods

5.16.2 Function Documentation

5.16.2.1 apply()

```
void apply (
    operation_t * p_operation,
    int * p_error )
```

applies the operation(runs the function with given args).

Parameters

p_operation	a pointer to the operation
p_error	a pointer to the variable which receives the error. Can be left NULL.

5.16.2.2 create_operation()

```
operation_t* create_operation (
    void(*)(vector_t *args, int *p_error) function,
    vector_t * args,
    void(*)(vector_t *args) free_args_data,
    int * p_error )
```

this function creates an operation instance

Parameters

function	the function which is to be applied
args	the args of the function
free_args_data	the functions which frees the data for given args. Can be left NULL if there is no need for one.
p_error	the pointer to the variable that receives the error. Can be left NULL

Returns

a pointer to the new operation

5.16.2.3 delete_operation() `void delete_operation (`

`operation_t * p_operation )`

this function frees the memory for an operation

Parameters

p_operation	a pointer to the operation to be freed

5.16.2.4 delete_operation_data() `void delete_operation_data (`

`operation_t * p_operation )`

this function frees the memory allocated for an operation, but not the operation itself

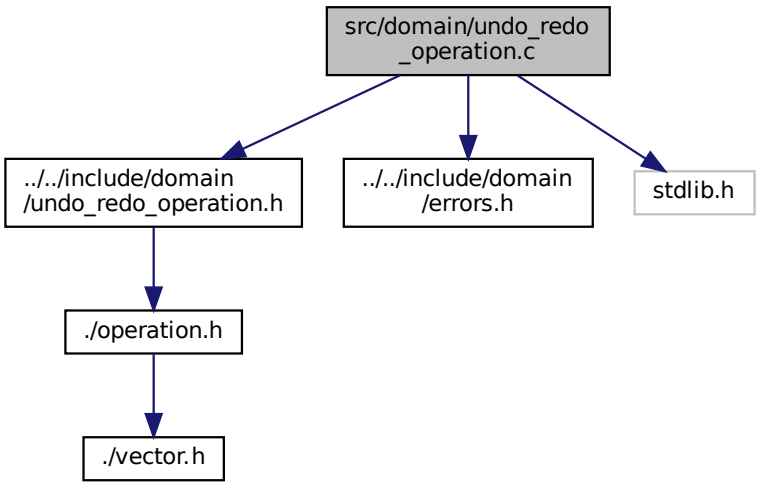
Parameters

p_operation	a pointer to the operation

5.17 src/domain/undo_redo_operation.c File Reference

```
#include "../../include/domain/undo_redo_operation.h"
#include "../../include/domain/errors.h"
#include <stdlib.h>
```

Include dependency graph for undo_redo_operation.c:



Functions

- undo_redo_operation_t * create_undo_redo_operation (operation_t *p_undo, operation_t *p_redo, int *p↩ _error)

  this function creates an undo_operation-redo_operation operation
- void delete_undo_redo_operation (undo_redo_operation_t *p_operation)

  this function deletes the memory allocated for an undo_operation-redo_operation operation
- void delete_undo_redo_operation_data (undo_redo_operation_t *p_operation)

  this function deletes the memory allocated for an undo_operation-redo_operation operation's data, but not the operation itself
- void undo_operation (undo_redo_operation_t *p_operation, int *p_error)

  this function applies the undo_operation operation of a given operation
- void redo_operation (undo_redo_operation_t *p_operation, int *p_error)

  this function applies the redo_operation operation of a given operation

5.17.1 Detailed Description

this file contains the model for an undo_redo operation. This operation consists of 2 operations, one which can undo_operation and one which can redo_operation.

5.17.2 Function Documentation

5.17.2.1 create_undo_redo_operation()

```
undo_redo_operation_t* create_undo_redo_operation (
        operation_t * p_undo,
        operation_t * p_redo,
        int * p_error )
```

this function creates an undo_operation-redo_operation operation

Parameters

p_undo	a pointer to the undo_operation operation
p_redo	a pointer to the redo_operation operation
p_error	a pointer to the variable which receives the error. Can be left NULL.

Returns

a pointer to the undo_operation-redo_operation operation

5.17.2.2 delete_undo_redo_operation()

```
void delete_undo_redo_operation (
        undo_redo_operation_t * p_operation )
```

this function deletes the memory allocated for an undo_operation-redo_operation operation

Parameters

p_operation	a pointer to the operation

5.17.2.3 delete_undo_redo_operation_data()

```
void delete_undo_redo_operation_data (
        undo_redo_operation_t * p_operation )
```

this function deletes the memory allocated for an undo_operation-redo_operation operation's data, but not the operation itself

Parameters

p_operation	

5.17.2.4 redo_operation()

```
void redo_operation (
        undo_redo_operation_t * p_operation,
        int * p_error )
```

this function applies the redo_operation operation of a given operation

Parameters

p_operation	the pointer to the undo_operation-redo_operation operation
p_error	a pointer to the variable which receives the error. Can be left NULL.

5.17.2.5 undo_operation()

```
void undo_operation (
        undo_redo_operation_t * p_operation,
        int * p_error )
```

this function applies the undo_operation operation of a given operation

Parameters

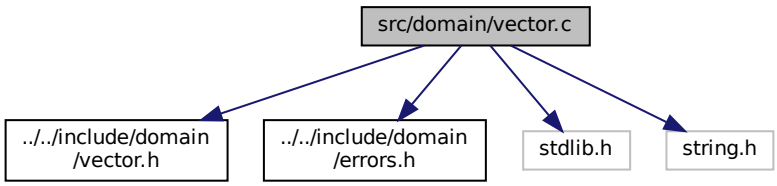
p_operation	the pointer to the undo_operation-redo_operation operation
p_error	a pointer to the variable which receives the error. Can be left NULL.

5.18 src/domain/vector.c File Reference

This file contains the implementation for the vector structure implemented in vector.h.

```
#include "../../include/domain/vector.h"
#include "../../include/domain/errors.h"
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for vector.c:



Functions

- vector_t * create_vector (int element_size, int initial_size, void(*free_element_data)(), int *p_error)

  This function creates a pointer to a new vector_t object.
- void delete_vector (vector_t *p_v)

  This function frees the memory for a vector_t object.
- void resize (vector_t *p_v, int size, int *p_error)

  This function resizes the current vector. It also changes its capacity.
- void change_capacity_vector (vector_t *p_v, int capacity, int *p_error)

  This function changes the capacity of the current vector.
- void push_back (vector_t *p_v, const void *p_element, int *p_error)

  This function resizes a given vector. It changes both size, and capacity. Deletes the any elements which do not fit in the new size. Any newly created elements are not initialized to any default value.
- void pop_back (vector_t *p_v, int *p_error)

  This function removes the last element of the vector. It changes only size.
- void * get_position_vector (const vector_t *p_v, int pos, int *p_error)

  gets a pointer to the element from specified position in the given vector
- void set_position_vector (vector_t *p_v, int pos, const void *p_element, int *p_error)

  sets an element on a given position in the vector to a given element

5.18.1 Detailed Description

This file contains the implementation for the vector structure implemented in vector.h.

5.18.2 Function Documentation

5.18.2.1 change_capacity_vector()

```
void change_capacity_vector (
        vector_t * p_v,
        int capacity,
        int * p_error )
```

This function changes the capacity of the current vector.

Parameters

p_v	the pointer to the vector to be resized
capacity	the new size of the vector
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL

5.18.2.2 create_vector()

```
vector_t* create_vector (
        int element_size,
        int initial_size,
        void(*)() free_elem_data,
        int * p_error )
```

This function creates a pointer to a new vector_t object.

Parameters

element_size	the size of a vector element(int bytes)
initial_size	the initial number of elements of the vector
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL
free_elem_data	a pointer to a function responsible for freeing data for a given element(but not the element itself). Can be NULL if no freeing should be applied.

Returns

a pointer to the newly created vector

5.18.2.3 delete_vector()

```
void delete_vector (
        vector_t * p_v )
```

This function frees the memory for a vector_t object.

Parameters

p_v	the pointer to the vector to be freed

5.18.2.4 get_position_vector()

```
void* get_position_vector (
    const vector_t * p_v,
    int pos,
    int * p_error )
```

gets a pointer to the element from specified position in the given vector

Parameters

p_v	pointer to the vector that is accessed
pos	the position that is wanted
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.18.2.5 pop_back()

```
void pop_back (
    vector_t * p_v,
    int * p_error )
```

This function removes the last element of the vector. It changes only size.

Parameters

p_v	a pointer to the vector from which the last element is removed
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.18.2.6 push_back()

```
void push_back (
    vector_t * p_v,
    const void * p_element,
    int * p_error )
```

This function resizes a given vector. It changes both size, and capacity. Deletes the any elements which do not fit in the new size. Any newly created elements are not initialized to any default value.

Parameters

p_v	the pointer to the vector on which to push back the element.
p_element	a pointer to the element to be pushed to the back of the vector.
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.18.2.7 resize()

```
void resize (
        vector_t * p_v,
        int size,
        int * p_error )
```

This function resizes the current vector. It also changes its capacity.

Parameters

p_v	the pointer to the vector to be resized
size	the new size of the vector
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL

5.18.2.8 set_position_vector()

```
void set_position_vector (
        vector_t * p_v,
        int pos,
        const void * p_element,
        int * p_error )
```

sets an element on a given position in the vector to a given element

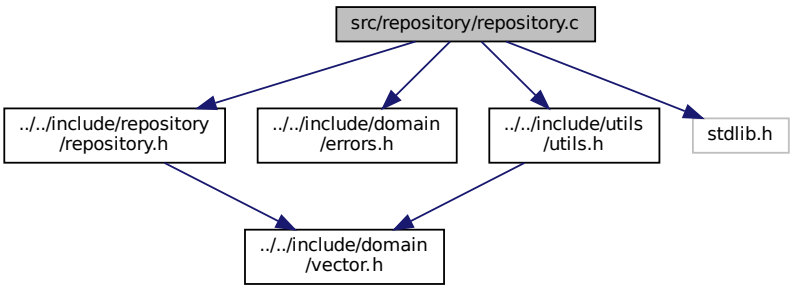
Parameters

p_v	pointer to the vector that is modified
pos	the position that will be modified
p_element	pointer to the elemenet with which the position is updated
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.19 src/repository/repository.c File Reference

```
#include "../../include/repository/repository.h"
#include "../../include/domain/errors.h"
#include "../../include/utils/utils.h"
#include <stdlib.h>
```

Include dependency graph for repository.c:



Functions

- repository_t * create_repository (int element_size, void(*free_element_data)(void *), int *p_error)
  creates a repository object and returns a pointer to it
- void delete_repository (repository_t *p_repository)
  frees the memory allocated for a given repository
- void add_repository (repository_t *p_repository, void *p_element, int *p_error)
  adds an element to a given repository
- vector_t * filter_repository (repository_t *p_repository, int(*p_filter_function)(void *repository_element, void **args), void **filter_function_args, int *p_error)
  returns a list of the indexes of elements which match a given filter function.
- void update_repository (repository_t *p_repository, int position, void *p_new_element, int *p_error)
  updates an element from the repository
- void remove_repository (repository_t *p_repository, int position, int *p_error)
  removes an element from the repository
- void * get_position_repository (repository_t *p_repository, int position, int *p_error)
  gets an element from a specified position
- int get_repository_size (repository_t *p_repository)
  returns the number of elements the repository currently has
- int get_repository_element_size (repository_t *p_repository)
  returns the size(in bytes) of a repository element

5.19.1 Detailed Description

This file contains the implementation for repository.h

5.19.2 Function Documentation

5.19.2.1 add_repository()

```
void add_repository (
        repository_t * p_repository,
        void * p_element,
        int * p_error )
```

adds an element to a given repository

Parameters

p_repository	a pointer to the repository
p_element	a pointer to the element which is adds to the repository
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.19.2.2 create_repository()

```
repository_t* create_repository (
        int element_size,
        void(*)(void *) free_element_data,
        int * p_error )
```

creates a repository object and returns a pointer to it

Parameters

element_size	the size of a repository element(in bytes)
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.
free_element_data	a pointer to a function that frees the data for an element(but not the element itself)

Returns

pointer to the newly created repository

5.19.2.3 delete_repository()

```
void delete_repository (
        repository_t * p_repository )
```

frees the memory allocated for a given repository

Parameters

p_repository	a pointer to the repository

5.19.2.4 filter_repository()

```
vector_t* filter_repository (
        repository_t * p_repository,
        int(*)(void *repository_element, void **args) p_filter_function,
        void ** filter_function_args,
        int * p_error )
```

returns a list of the indexes of elements which match a given filter function.

Parameters

p_repository	a pointer to the repository

Parameters

p_filter_function	a filter function which decides which elements are returned. If the function returns 1, the element is added to the answer. The filter function should have 2 arguments, the first one should be the pointer to the repository element, the second one should be any custom function arguments.
filter_function_args	any custom filter function arguments
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

Returns

pointer to a vector that contains the indexes of the elements which are matching a given filter function.

5.19.2.5 get_position_repository()

```
void* get_position_repository (
        repository_t * p_repository,
        int position,
        int * p_error )
```

gets an element from a specified position

Parameters

p_repository	a pointer to the repository
position	the position wanted

Returns

a pointer to the element

5.19.2.6 get_repository_element_size()

```
int get_repository_element_size (
        repository_t * p_repository )
```

returns the size(in bytes) of a repository element

Parameters

p_repository	a pointer to the repository

Returns

the size of a repository element

5.19.2.7 get_repository_size() int get_repository_size (repository_t * p_repository)

returns the number of elements the repository currently has

Parameters

p_repository	a pointer to the repository

Returns

the number of elements the repository contains

5.19.2.8 remove_repository() void remove_repository (repository_t * p_repository, int position, int * p_error)

removes an element from the repository

Parameters

p_repository	a pointer to the repository
position	the position of the element which is to be removed
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.19.2.9 update_repository() void update_repository (repository_t * p_repository, int position, void * p_new_element, int * p_error)

updates an element from the repository

Parameters

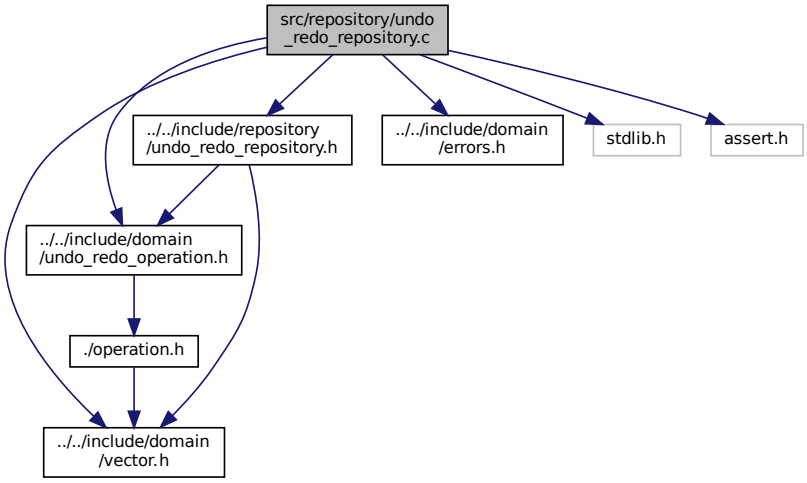
p_repository	a pointer to the repository
position	the position of the element which is to be updated
p_new_element	a pointer to the new value of the updated element
p_error	a pointer to the variable receiving the error code of this function. Can be left NULL.

5.20 src/repository/undo_redo_repository.c File Reference

```
#include "../../include/domain/vector.h"
#include "../../include/domain/undo_redo_operation.h"
```

```
#include "../../include/repository/undo_redo_repository.h"
#include "../../include/domain/errors.h"
#include <stdlib.h>
#include <assert.h>
```

Include dependency graph for undo_redo_repository.c:



Functions

- undo_redo_repository_t * create_undo_redo_repository (int *p_error)
  this function creates an undo redo repository
- void delete_undo_redo_repository (undo_redo_repository_t *p_undo_redo_repository)
  this function frees the memory for a given undo redo repository
- void add_undo_repository (undo_redo_repository_t *p_undo_redo_repository, undo_redo_operation_t *p_operation, int *p_error)
  add an operation to the repository
- void undo_repository (undo_redo_repository_t *p_undo_redo_repository, int *p_error)
  this function undoes an operation
- void redo_repository (undo_redo_repository_t *p_undo_redo_repository, int *p_error)
  this function redoes an operation

5.20.1 Detailed Description

this file contains the implementation for a repository responsible for managing undo-redo_ operations

5.20.2 Function Documentation

5.20.2.1 add_undo_repository()

```
void add_undo_repository (
        undo_redo_repository_t * p_undo_redo_repository,
        undo_redo_operation_t * p_operation,
        int * p_error )
```

add an operation to the repository

Parameters

p_undo_redo_repository	a pointer to the repository
p_operation	a pointer to the undo redo operation
p_error	a pointer to the variable that receives the error code. Can be left NULL.

5.20.2.2 create_undo_redo_repository()

```
undo_redo_repository_t* create_undo_redo_repository (
        int * p_error )
```

this function creates an undo redo repository

Parameters

p_error	a pointer to the variable that receives the error code. Can be left NULL.

Returns

a pointer to the undo redo repository created

5.20.2.3 delete_undo_redo_repository()

```
void delete_undo_redo_repository (
        undo_redo_repository_t * p_undo_redo_repository )
```

this function frees the memory for a given undo redo repository

Parameters

p_undo_redo_repository	a pointer to the undo redo repository

5.20.2.4 redo_repository()

```
void redo_repository (
        undo_redo_repository_t * p_undo_redo_repository,
        int * p_error )
```

this function redoes an operation

Parameters

p_undo_redo_repository	
p_error	a pointer to the variable that receives the error code. Can be left NULL.

5.20.2.5 undo_repository()

```
void undo_repository (
        undo_redo_repository_t * p_undo_redo_repository,
        int * p_error )
```

this function undoes an operation

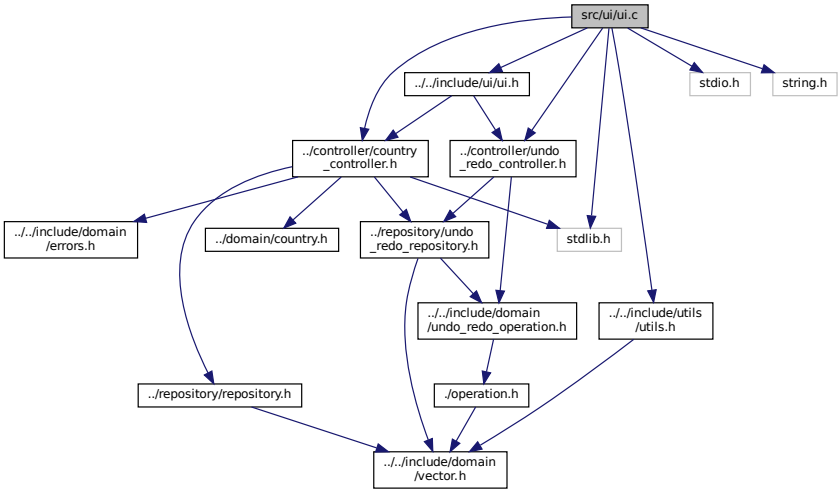
Parameters

p_undo_redo_repository	a pointer to the undo redo repository
p_error	a pointer to the variable that receives the error code. Can be left NULL.

5.21 src/ui/ui.c File Reference

```
#include "../../include/ui/ui.h"
#include "../../include/utils/utils.h"
#include "../../include/controller/country_controller.h"
#include "../../include/controller/undo_redo_controller.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for ui.c:



Functions

- ui_t * create_ui (country_controller_t *p_controller, undo_redo_controller_t *p_undo_redo_controller, int *p_error)

  creates ui object
- void delete_ui (ui_t *p_ui)
- void display_country (ui_t *p_ui, country_t *p_country)

  this function displays a country_t object
- void display_countries (ui_t *p_ui, vector_t *p_countries)

  this function displays a vector of countries

- void display_help (ui_t *p_ui)

  display the help for this app
- void run (ui_t *p_ui)

  the main loop of the ui

5.21.1 Detailed Description

contains the implementation of ui.h

5.21.2 Function Documentation

5.21.2.1 create_ui()

```
ui_t* create_ui (
        country_controller_t * p_controller,
        undo_redo_controller_t * p_undo_redo_controller,
        int * p_error )
```

creates ui object

Parameters

p_controller	pointer to the controller
p_error	pointer to the variable receiving the error code. Can be left NULL.

5.21.2.2 delete_ui()

```
void delete_ui (
        ui_t * p_ui )
```

frees the memory of a ui_t object

Parameters

p_ui	pointer to the ui object

5.21.2.3 display_countries()

```
void display_countries (
        ui_t * p_ui,
        vector_t * p_countries )
```

this function displays a vector of countries

Parameters

p_ui	a pointer to the ui
p_countries	a pointer to the vector containing the countries

5.21.2.4 display_country() `void display_country (`

```
ui_t * p_ui,
country_t * p_country )
```

this function displays a country_t object

Parameters

p_ui	pointer to the ui
p_country	pointer to the country to be displayed

5.21.2.5 display_help() `void display_help (`

```
ui_t * p_ui )
```

display the help for this app

Parameters

p↩ _ui	pointer to the ui

TODO maybe allow change of name and continent

5.21.2.6 run() `void run (`

```
ui_t * p_ui )
```

the main loop of the ui

Parameters

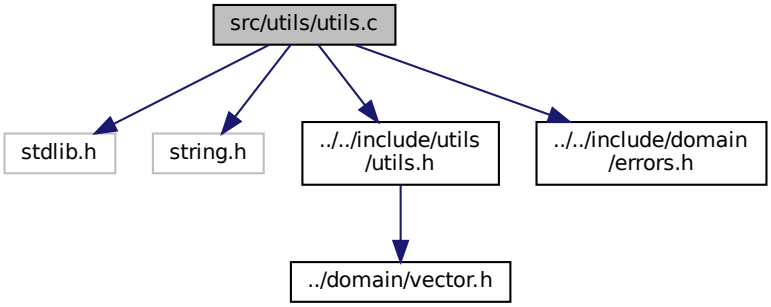
p↩ _ui	the pointer to the ui

5.22 src/utils/utils.c File Reference

```
#include <stdlib.h>
#include <string.h>
```

```
#include "../../include/utils/utils.h"
#include "../../include/domain/errors.h"
```

Include dependency graph for utils.c:



Functions

- char ∗ get_lowercase (const char ∗p_s, int ∗p_error)

  gets a pointer to a new char string having the lowercase version of a given string
- void swap (void ∗p_a, void ∗p_b, int cnt_bytes)

  swaps the next cnt_bytes from a and b
- void sort (vector_t ∗p_v, int l, int r, int(∗cmp)(const void ∗∗a, const void ∗∗b))

  this function sorts a given vector using the compare function on the range [l,r)
- void free_vector_string (void ∗p_v)

  this function is responsible for freeing a string from a vector which holds pointers
- vector_t ∗ split_string (char ∗p_s, char delim, int ∗p_error)

  this function returns a vector structure containing pointers to all tokens in a given string.

5.22.1 Detailed Description

contains the implementations of various general purpose functions.

5.22.2 Function Documentation

5.22.2.1 free_vector_string() `void free_vector_string ( void * p_v )`

this function is responsible for freeing a string from a vector which holds pointers

Parameters

p_v	

5.22.2.2 get_lowercase()

```
char* get_lowercase (
    const char * p_s,
    int * p_error )
```

gets a pointer to a new char string having the lowercase version of a given string

Parameters

p_s	the pointer to the string
p_error	a pointer to the variable that gets the error

Returns

pointer to the new string

5.22.2.3 sort()

```
void sort (
    vector_t * p_v,
    int l,
    int r,
    int(*)(const void **a, const void **b) cmp )
```

this function sorts a given vector using the compare function on the range [l,r)

Parameters

p_v	a pointer to the vector to be sorted
l	the left index
r	the right index
cmp	the compare index which returns 1 if a $<$ b

5.22.2.4 split_string()

```
vector_t* split_string (
    char * p_s,
    char delim,
    int * p_error )
```

this function returns a vector structure containing pointers to all tokens in a given string.

Parameters

p_s	a pointer to the string being processed
delim	a character which is the 'deliminator'
p_error	a pointer to the variable that gets the error

Returns

a pointer to a vector containing pointers to new strings containing the arguments

5.22.2.5 swap()

```
void swap (
        void * p_a,
        void * p_b,
        int cnt_bytes )
```

swaps the next cnt_bytes from a and b

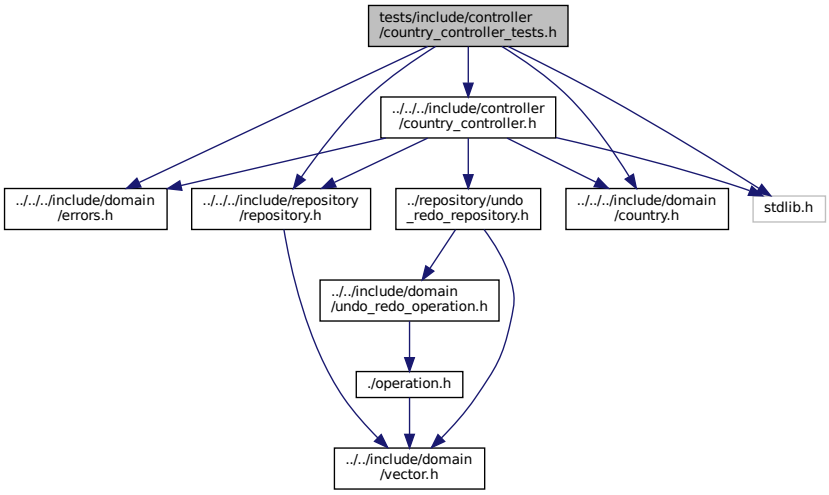
Parameters

p_a	pointer to the first address
p_b	pointer to the second address
cnt_bytes	the number of bytes to swap

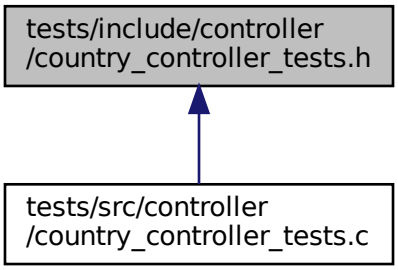
5.23 tests/include/controller/country_controller_tests.h File Reference

```
#include "../../../include/domain/errors.h"
#include "../../../include/repository/repository.h"
#include "../../../include/domain/country.h"
#include "../../../include/controller/country_controller.h"
#include <stdlib.h>
```

Include dependency graph for country_controller_tests.h:



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



Functions

- void country_controller_test_create_delete_country_controller ()
  this function tests the create and delete functionalities for country controller
- void country_controller_test_add_country_ui ()
  this function tests the add from ui country controller function
- void country_controller_test_add_country_from_args ()
  this function tests the add from args country controller function
- void country_controller_test_remove_country_ui ()
  this function tests the remove from ui country controller function
- void country_controller_test_remove_country_from_args ()
  this function tests the remove from args country controller function
- void country_controller_test_update_country_ui ()
  this function tests the remove from args country controller function
- void country_controller_test_update_country_from_args ()
  this function tests update country from args country controller function
- void country_controller_test_migrate_ui ()
  this function tests the migrate ui country controller function
- void country_controller_test_migrate_from_args ()
  this function tests migrate from args country controller function
- void country_controller_test_get_countries_containing_string ()
  this function tests the get countries containing string country controller function
- void country_controller_test_get_countries_with_at_least ()
  this function tests the get countries with at least country controller function
- void country_controller_test_get_countries_from_continent ()
  this function tests the get countries from continent country controller function
- void country_controller_test_test_all ()
  this function runs all country controller tests

5.23.1 Detailed Description

This file contains the models for the test functions for country_controller

5.24 tests/include/controller/undo_redo_tests.h File Reference

Functions

- void undo_redo_test1 ()
  this function runs the first test of the undo redo functionality
- void undo_redo_test_all ()
  this function runs all undo_redo related tests

5.24.1 Detailed Description

this file contains the undo redo related functionalities tests

5.24.2 Function Documentation

5.24.2.1 undo_redo_test1() `void undo_redo_test1 ( )`

this function runs the first test of the undo redo functionality

for test uses

5.25 tests/include/domain/country_tests.h File Reference

Functions

- void country_test_create_delete ()
  this function tests the creation and deletion of country_t objects
- void country_test_all ()
  this function runs all country tests

5.25.1 Detailed Description

this file contains the definitions for the country tests

5.26 tests/include/domain/domain_tests.h File Reference

Functions

- void domain_test_all ()
  this function runs all domain tests

5.26.1 Detailed Description

this file contains the definitions for running all domain tests

5.27 tests/include/domain/errors_tests.h File Reference

Functions

- void errors_test_set_error ()

  this function tests the function set_error
- void errors_test_all ()

  this function runs all 'errors' tests

5.27.1 Detailed Description

this file contains the definition of the functions used for testing errors.h

5.28 tests/include/domain/operation_tests.h File Reference

Functions

- void operation_test_create_delete_operation ()

  this function tests the create and delete functions for an operation
- void operation_test_apply ()

  this function tests the apply function for an operation
- void operation_test_all ()

  this function tests all operation tests

5.28.1 Detailed Description

this file contains the model for the operation_t tests

5.29 tests/include/domain/undo_redo_operation_tests.h File Reference

Functions

- void undo_redo_operation_test_create_delete_undo_redo_operation ()

  this function tests the create and delete undo_operation-redo_operation operation function
- void undo_redo_operation_test_undo_redo ()

  this function tests the undo_repository and redo_repository operations for an undo_operation-redo_operation operation
- void undo_redo_operation_test_all ()

  this function runs all tests for undo_operation-redo_operation operations

5.29.1 Detailed Description

this file contains the model for an undo_redo operation tests

5.30 tests/include/domain/vector_tests.h File Reference

Functions

- void vector_test_set_position_vector ()
  this function tests the set function on a vector
- void vector_test_get_position_vector ()
  this function tests the get function on a vector
- void vector_test_pop_back ()
  this function tests the pop back function on a vector
- void vector_test_push_back ()
  this function tests the push back function on a vector
- void vector_test_change_capacity ()
  this function tests the change capacity function on a vector
- void vector_test_resize ()
  this function tests the resize function on a vector
- void vector_test_create_delete ()
  this function tests the create and delete functions on a vector
- void vector_test_all ()
  this function runs all vector tests

5.30.1 Detailed Description

This file contains the definitions for all vector tests

5.31 tests/include/repository/repository_tests.h File Reference

Functions

- void repository_test_create_delete_repository ()
  this function tests the create and delete repository functions
- void repository_test_add_repository ()
  this function tests the add repository function
- void repository_test_filter_repository ()
  this function tests the filter repository function
- void repository_test_update_repository ()
  this function tests the update repository function
- void repository_test_remove_repository ()
  this function tests the remove repository function
- void repository_test_get_position_repository ()
  this function tests the get position repository function
- void repository_test_get_size ()
  this function tests the get size repository function
- void repository_test_get_element_size ()
  ths function tests the get element size repository function
- void repository_test_all ()
  this function runs all repository tests

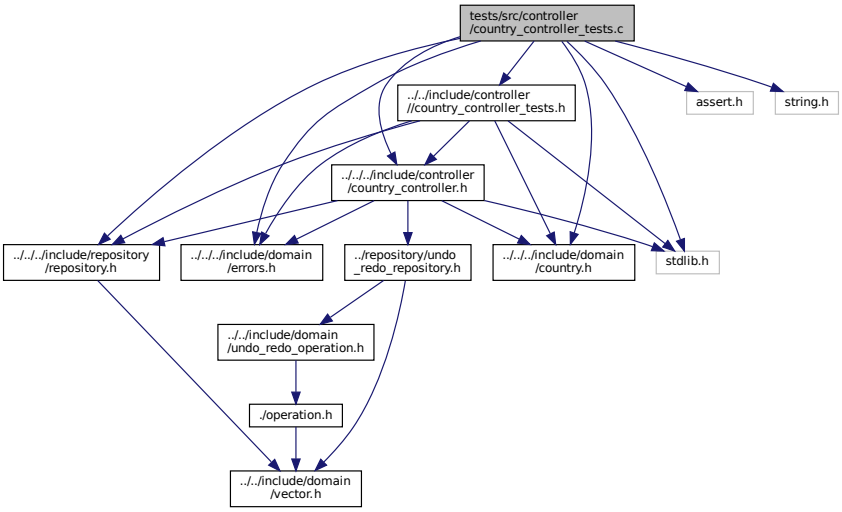
5.31.1 Detailed Description

This file contains the definitions for the tests for the repository

5.32 tests/src/controller/country_controller_tests.c File Reference

```
#include "../../../include/domain/errors.h"
#include "../../../include/repository/repository.h"
#include "../../../include/domain/country.h"
#include "../../../include/controller/country_controller.h"
#include "../../include/controller//country_controller_tests.h"
#include <stdlib.h>
#include <assert.h>
#include <string.h>
```

Include dependency graph for country_controller_tests.c:



Functions

- void country_controller_test_create_delete_country_controller ()

  this function tests the create and delete functionalities for country controller
- void country_controller_test_add_country_ui ()

  this function tests the add from ui country controller function
- void country_controller_test_add_country_from_args ()

  this function tests the add from args country controller function
- void country_controller_test_remove_country_ui ()

  this function tests the remove from ui country controller function
- void country_controller_test_remove_country_from_args ()

  this function tests the remove from args country controller function
- void country_controller_test_update_country_ui ()

  this function tests the remove from args country controller function
- void country_controller_test_update_country_from_args ()

  this function tests update country from args country controller function

- void country_controller_test_migrate_ui ()

  this function tests the migrate ui country controller function
- void country_controller_test_migrate_from_args ()

  this function tests migrate from args country controller function
- void country_controller_test_get_countries_containing_string ()

  this function tests the get countries containing string country controller function
- void country_controller_test_get_countries_with_at_least ()

  this function tests the get countries with at least country controller function
- void country_controller_test_get_countries_from_continent ()

  this function tests the get countries from continent country controller function
- void country_controller_test_test_all ()

  this function runs all country controller tests

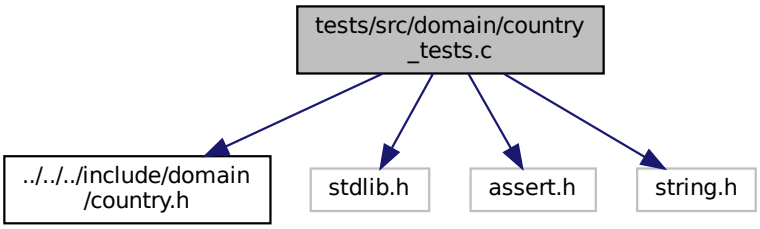
5.32.1 Detailed Description

This file contains the implementations for the test functions for country_controller

5.33 tests/src/domain/country_tests.c File Reference

```
#include "../../../include/domain/country.h"
#include <stdlib.h>
#include <assert.h>
#include <string.h>
```

Include dependency graph for country_tests.c:



Functions

- void country_test_create_delete ()

  this function tests the creation and deletion of country_t objects
- void country_test_all ()

  this function runs all country tests

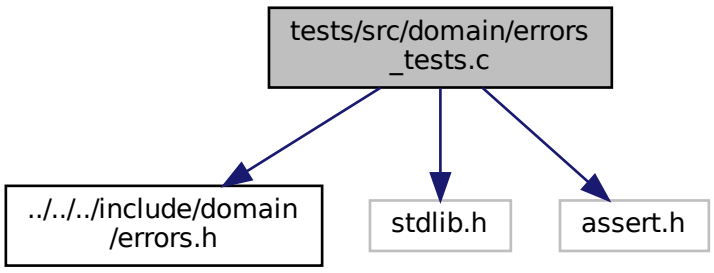
5.33.1 Detailed Description

this file contains the tests for country related functions

5.34 tests/src/domain/errors_tests.c File Reference

```
#include "../../../include/domain/errors.h"
#include <stdlib.h>
#include <assert.h>
```

Include dependency graph for errors_tests.c:



Functions

- void errors_test_set_error ()

  this function tests the function set_error

- void errors_test_all ()

  this function runs all 'errors' tests

5.34.1 Detailed Description

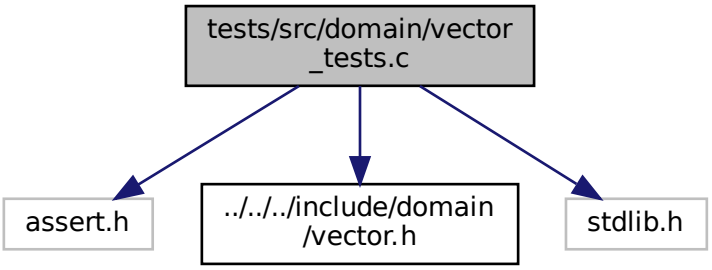
this file contains the implementations of the errors tests

5.35 tests/src/domain/vector_tests.c File Reference

```
#include <assert.h>
#include "../../../include/domain/vector.h"
```

```
#include <stdlib.h>
```

Include dependency graph for vector_tests.c:



Functions

- void vector_test_set_position_vector ()

  this function tests the set function on a vector
- void vector_test_get_position_vector ()

  this function tests the get function on a vector
- void vector_test_pop_back ()

  this function tests the pop back function on a vector
- void vector_test_push_back ()

  this function tests the push back function on a vector
- void vector_test_change_capacity ()

  this function tests the change capacity function on a vector
- void vector_test_resize ()

  this function tests the resize function on a vector
- void vector_test_create_delete ()

  this function tests the create and delete functions on a vector
- void vector_test_all ()

  this function runs all vector tests

5.35.1 Detailed Description

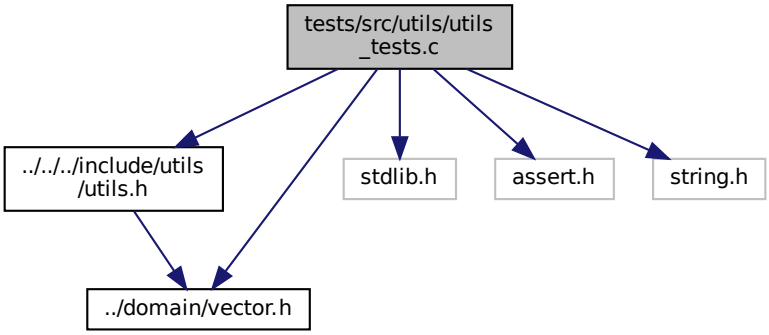
This file contains the implementations for all vector tests

5.36 tests/src/utils/utils_tests.c File Reference

```
#include "../../../include/utils/utils.h"
#include "../../../include/domain/vector.h"
#include <stdlib.h>
#include <assert.h>
```

```
#include <string.h>
```

Include dependency graph for utils_tests.c:



Functions

- void utils_test_get_lowercase ()

  this function tests the get_lowercase function from utils.c
- void utils_test_sort ()

  this function tests the sort function from utils.c
- void utils_test_swap ()

  this function tests the swap function from utils.c
- void utils_test_split_string ()

  this function tests the split string function from utils.h
- void **utils_test_all** ()

5.36.1 Detailed Description

contains the implementation for the utils_test functions

Index