# Railroad Tycoon Prototyp

1.0.0

Erzeugt von Doxygen 1.8.17

# Inhaltsverzeichnis

# Kapitel 1

# SWT Praktikum

Hier ist eine kleine Anleitung wie man das Projekt auf seinem eigenen Rechner synchronisiert:

1. git installieren
2. >> git clone https://github.com/davidtraum/swt/
3. >> cd swt

Wenn man was geändert hat:

(0. Ins Basisverzeichnis vom Projekt gehen)

1. >> git add *

1. >> git commit -m "Kurze Nachricht was man gemacht hat"
2. >> git push origin master (Oder eigenen Branch angeben)

## 1.1 Changelog

| Datum | Funktion |
|---|---|
| 28.10. | Start Changelog |
| 28.10. | Animation beim Klick auf Städte |
| 28.10. | Übersichtskarte mit Taste O |
| 29.10. | Statuspanel hinzugefügt |

# Kapitel 2

# Verzeichnis der Namensbereiche

## 2.1 Liste aller Namensbereiche

Liste aller Namensbereiche mit Kurzbeschreibung:

# Kapitel 3

# Hierarchie-Verzeichnis

## 3.1 Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

# Kapitel 4

# Klassen-Verzeichnis

## 4.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

# Kapitel 5

# Datei-Verzeichnis

## 5.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

# Kapitel 6

# Dokumentation der Namensbereiche

## 6.1 Ui-Namensbereichsreferenz

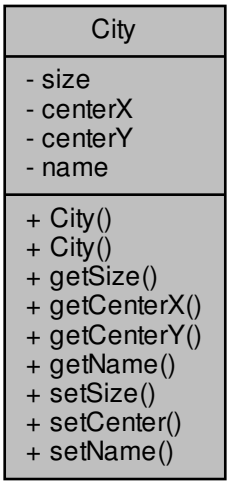
# Kapitel 7

# Klassen-Dokumentation

## 7.1 City Klassenreferenz

`#include <city.h>`

Zusammengehörigkeiten von City:




### Öffentliche Methoden

- City (int pX, int pY, int pSize)

  *City::City Erzeugt eine Stadt mit vorgegebenen Parametern.*
- City ()

  *City::City Erzeugt eine leere Stadt.*
- int getSize ()

*City::getSize Gibt die Anzahl der Felder zurück die zur Stadt gehören.*

- int getCenterX ()

  *City::getCenterX Gibt den X-Index des Mittelpunktes.*
- int getCenterY ()

  *City::getCenterX Gibt den Y-Index des Mittelpunktes.*
- std::string getName ()

  *City::getName Gibt den Namen der Stadt.*
- void setSize (int pSize)

  *City::setSize Gibt die Größe der Stadt zurück (Anzahl der Gebäude)*
- void setCenter (int pX, int pY)

  *City::setCenter Setzt den Mittelpunkt der Stadt.*
- void setName (std::string pName)

  *City::setName Setzt den Namen der Stadt.*

## Private Attribute

- int size
- int centerX
- int centerY
- std::string name

### 7.1.1 Beschreibung der Konstruktoren und Destruktoren

#### 7.1.1.1 City() [1/2]

```
City::City (
            int pX,
            int pY,
            int pSize )
```

City::City Erzeugt eine Stadt mit vorgegebenen Parametern.

**Parameter**

| | |
|---|---|
| *pX* | Der X-Index des Mittelpunktes. |
| *pY* | Der Y-Index des Mittelpunktes. |
| *pSize* | Die größe der Stadt. |

#### 7.1.1.2 City() [2/2]

```
City::City ( )
```

City::City Erzeugt eine leere Stadt.

## 7.1.2 Dokumentation der Elementfunktionen

### 7.1.2.1 getCenterX()

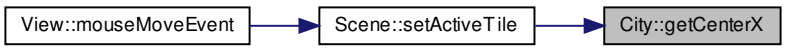
```
int City::getCenterX ( )
```

City::getCenterX Gibt den X-Index des Mittelpunktes.

**Rückgabe**

Der X-Index des Mittelpunktes der Stadt.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




### 7.1.2.2 getCenterY()

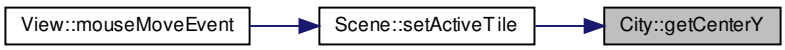
```
int City::getCenterY ( )
```

City::getCenterX Gibt den Y-Index des Mittelpunktes.

**Rückgabe**

Der Y-Index des Mittelpunktes der Stadt.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 7.1.2.3 getName()

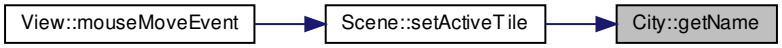
```
std::string City::getName ( )
```

City::getName Gibt den Namen der Stadt.

**Rückgabe**

Der Name der Stadt.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.1.2.4 getSize()

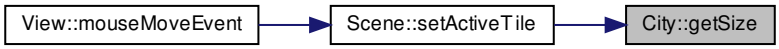
```
int City::getSize ( )
```

City::getSize Gibt die Anzahl der Felder zurück die zur Stadt gehören.

**Rückgabe**

Die Größe.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.1.2.5 setCenter()

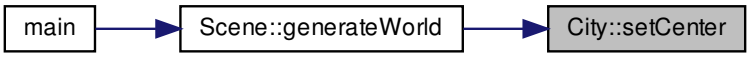
```
void City::setCenter (
            int pX,
            int pY )
```

City::setCenter Setzt den Mittelpunkt der Stadt.

**Parameter**

| | |
|---|---|
| *pX* | Der X-Index. |
| *pY* | Der Y-Index. |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.1.2.6 setName()

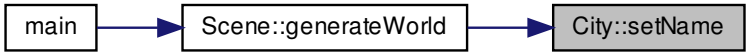
```
void City::setName (
            std::string pName )
```

City::setName Setzt den Namen der Stadt.

**Parameter**

| | |
|---|---|
| *pName* | Der neue Name. |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.1.2.7 setSize()

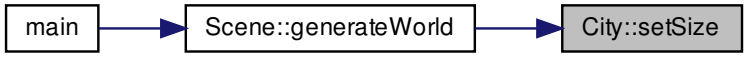
```
void City::setSize (
            int pSize )
```

City::setSize Gibt die Größe der Stadt zurück (Anzahl der Gebäude)

**Parameter**

| | |
|---|---|
| *pSize* | Die Größe der Stadt |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




### 7.1.3 Dokumentation der Datenelemente

#### 7.1.3.1 centerX

```
int City::centerX  [private]
```

#### 7.1.3.2 centerY

```
int City::centerY  [private]
```

#### 7.1.3.3 name

```
std::string City::name  [private]
```

#### 7.1.3.4 size

```
int City::size  [private]
```

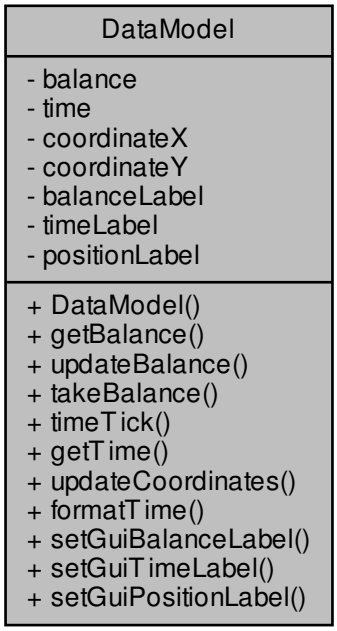
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/city.h
- src/application/city.cpp

## 7.2 DataModel Klassenreferenz

`#include <datamodel.h>`

Zusammengehörigkeiten von DataModel:




### Öffentliche Methoden

- DataModel ()

  *DataModel::DataModel Diese Klasse verwaltet alle globalen Daten rund um den Spielverlauf, z.B. den Kontostand.*
- int getBalance ()

  *DataModel::getBalance Liefert den aktuellen Kontostand zurück.*
- void updateBalance (int pBalance)

  *DataModel::updateBalance Aktualisiert den Kontostand. Auch in Anzeigen etc.*
- bool takeBalance (int pAmount)

  *DataModel::takeBalance Zieht Geld ab falls noch genug da ist.*
- void timeTick ()

  *DataModel::timeTick Wird aufgerufen wenn eine Zeiteinheit verstrichen ist. Erhöht den Timecode.*
- long getTime ()

  *DataModel::getTime Liefert die aktuelle Zeit als Timecode. (Zahl die je nach Geschwindigkeit wächst)*
- void updateCoordinates (int pX, int pY)

  *DataModel::updateCoordinates Aktualisiert die Koordinaten des fokussierten Quadrantens.*
- std::string formatTime (long pTime)

  *DataModel::formatTime Formattiert einen Timecode als String.*
- void setGuiBalanceLabel (QLabel ∗label)

*DataModel::setGuiBalanceLabel Setzt das Label in welchem der Kontostand dargestellt wird.*

- void setGuiTimeLabel (QLabel ∗label)

  *DataModel::setGuiTimeLabel Setzt das Label in welchem die Zeit dargestellt wird.*

- void setGuiPositionLabel (QLabel ∗label)

  *DataModel::setGuiTimeLabel Setzt das Label in welchem die Koordinate dargestellt wird.*

## Private Attribute

- int balance
- long time
- int coordinateX
- int coordinateY
- QLabel ∗ balanceLabel
- QLabel ∗ timeLabel
- QLabel ∗ positionLabel

### 7.2.1 Beschreibung der Konstruktoren und Destruktoren

#### 7.2.1.1 DataModel()

```
DataModel::DataModel ( )
```

DataModel::DataModel Diese Klasse verwaltet alle globalen Daten rund um den Spielverlauf, z.B. den Kontostand.

### 7.2.2 Dokumentation der Elementfunktionen

#### 7.2.2.1 formatTime()

```
std::string DataModel::formatTime (
            long pTime )
```

DataModel::formatTime Formattiert einen Timecode als String.

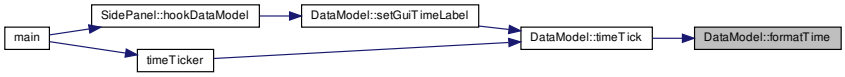
**Parameter**

| | |
|---|---|
| *pTime* | Der Timecode. |

**Rückgabe**

Der Text.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.2.2.2 getBalance()

```
int DataModel::getBalance ( )
```

DataModel::getBalance Liefert den aktuellen Kontostand zurück.

**Rückgabe**

Der aktuelle Kontostand.

#### 7.2.2.3 getTime()

```
long DataModel::getTime ( )
```

DataModel::getTime Liefert die aktuelle Zeit als Timecode. (Zahl die je nach Geschwindigkeit wächst)

**Rückgabe**

Der Timecode.

#### 7.2.2.4 setGuiBalanceLabel()

```
void DataModel::setGuiBalanceLabel (
            QLabel * label )
```

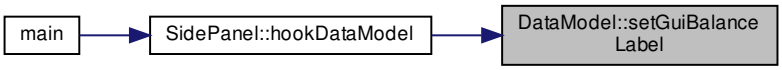
DataModel::setGuiBalanceLabel Setzt das Label in welchem der Kontostand dargestellt wird.

**Parameter**

| | |
|---|---|
| *label* | Ein Pointer auf ein QLabel Objekt. |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:


Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.2.2.5 setGuiPositionLabel()

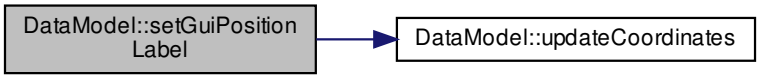
```
void DataModel::setGuiPositionLabel (
            QLabel * label )
```

DataModel::setGuiTimeLabel Setzt das Label in welchem die Koordinate dargestellt wird.

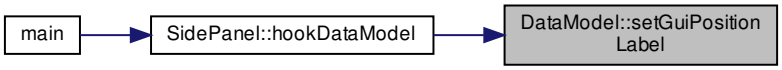
**Parameter**

| | |
|---|---|
| *label* | Ein Pointer auf ein QLabel Objekt. |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.2.2.6 setGuiTimeLabel()

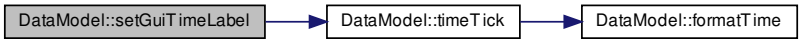
```
void DataModel::setGuiTimeLabel (
            QLabel * label )
```

DataModel::setGuiTimeLabel Setzt das Label in welchem die Zeit dargestellt wird.

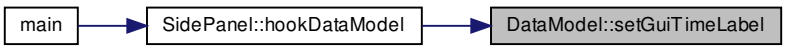
**Parameter**

| | |
|---|---|
| *label* | Ein Pointer auf ein QLabel Objekt. |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:




Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.2.2.7 takeBalance()

```
bool DataModel::takeBalance (
            int pAmount )
```

DataModel::takeBalance Zieht Geld ab falls noch genug da ist.

**Parameter**

| | |
|---|---|
| *pAmount* | Die Geldzahl zum Entfernen. |

**Rückgabe**

true wenn genug Geld da war und entfernt wurde. false wenn nicht genug Geld da ist.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

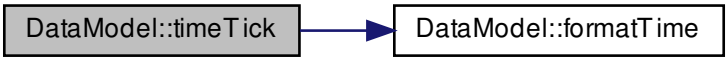

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:


#### 7.2.2.8 timeTick()

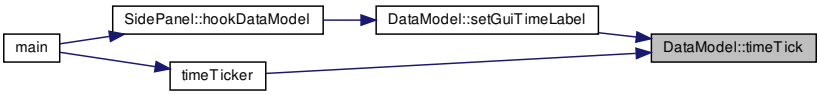
```
void DataModel::timeTick ( )
```

DataModel::timeTick Wird aufgerufen wenn eine Zeiteinheit verstrichen ist. Erhöht den Timecode.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.2.2.9 updateBalance()

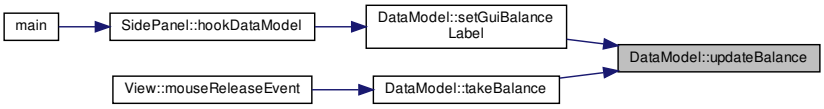
```
void DataModel::updateBalance (
            int pBalance )
```

DataModel::updateBalance Aktualisiert den Kontostand. Auch in Anzeigen etc.

**Parameter**

| | |
|---|---|
| *pBalance* | Der neue Kontostand. |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.2.2.10 updateCoordinates()

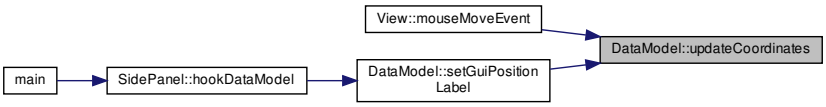
```
void DataModel::updateCoordinates (
            int pX,
            int pY )
```

DataModel::updateCoordinates Aktualisiert die Koordinaten des fokussierten Quadrantens.

**Parameter**

| | |
|---|---|
| *pX* | Die X Koordinate. |
| *pY* | Die Y Koordinate. |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




## 7.2.3 Dokumentation der Datenelemente

### 7.2.3.1 balance

```
int DataModel::balance  [private]
```

### 7.2.3.2 balanceLabel

```
QLabel* DataModel::balanceLabel  [private]
```

### 7.2.3.3 coordinateX

```
int DataModel::coordinateX  [private]
```

### 7.2.3.4 coordinateY

```
int DataModel::coordinateY  [private]
```

### 7.2.3.5 positionLabel

```
QLabel* DataModel::positionLabel  [private]
```

#### 7.2.3.6 time

```
long DataModel::time  [private]
```

#### 7.2.3.7 timeLabel

```
QLabel* DataModel::timeLabel  [private]
```

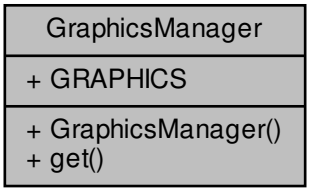
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/datamodel.h
- src/application/datamodel.cpp

## 7.3 GraphicsManager Klassenreferenz

```
#include <graphicsmanager.h>
```

Zusammengehörigkeiten von GraphicsManager:




### Öffentliche Methoden

- GraphicsManager ()

  *GraphicsManager::GraphicsManager Lädt alle Grafiken. Neue bitte im selben Stil ergänzen.*
- QPixmap get (std::string key)

  *GraphicsManager::get Liefert eine Grafik mit einem bestimmten Namen.*

### Öffentliche Attribute

- std::map< std::string, QPixmap > GRAPHICS

### 7.3.1 Beschreibung der Konstruktoren und Destruktoren

#### 7.3.1.1 GraphicsManager()

```
GraphicsManager::GraphicsManager ( )
```

GraphicsManager::GraphicsManager Lädt alle Grafiken. Neue bitte im selben Stil ergänzen.

### 7.3.2 Dokumentation der Elementfunktionen

#### 7.3.2.1 get()

```
QPixmap GraphicsManager::get (
            std::string key )
```

GraphicsManager::get Liefert eine Grafik mit einem bestimmten Namen.

**Parameter**

| | |
|---|---|
| *key* | Name der Grafik. |

**Rückgabe**

Die Grafik.

### 7.3.3 Dokumentation der Datenelemente

#### 7.3.3.1 GRAPHICS

```
std::map<std::string, QPixmap> GraphicsManager::GRAPHICS
```

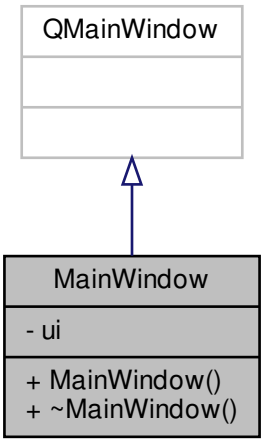
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/graphicsmanager.h
- src/application/graphicsmanager.cpp

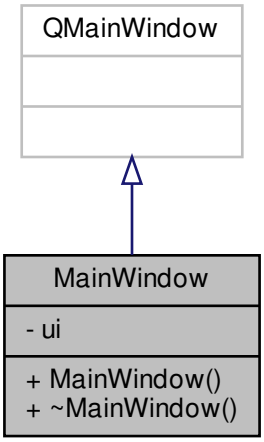
## 7.4 MainWindow Klassenreferenz

`#include <mainwindow.h>`

Klassendiagramm für MainWindow:




Zusammengehörigkeiten von MainWindow:




### Öffentliche Methoden

- MainWindow (QWidget ∗parent=nullptr)
  *MainWindow::MainWindow.*
- ∼MainWindow ()
  *MainWindow::∼MainWindow.*

### Private Attribute

- Ui::MainWindow * ui

## 7.4.1 Beschreibung der Konstruktoren und Destruktoren

### 7.4.1.1 MainWindow()

```
MainWindow::MainWindow (
            QWidget * parent = nullptr )
```

MainWindow::MainWindow.

**Parameter**

| | |
|---|---|
| *parent* | |

### 7.4.1.2 ~MainWindow()

```
MainWindow::~MainWindow ( )
```

MainWindow::~MainWindow.

## 7.4.2 Dokumentation der Datenelemente

### 7.4.2.1 ui

```
Ui::MainWindow* MainWindow::ui  [private]
```

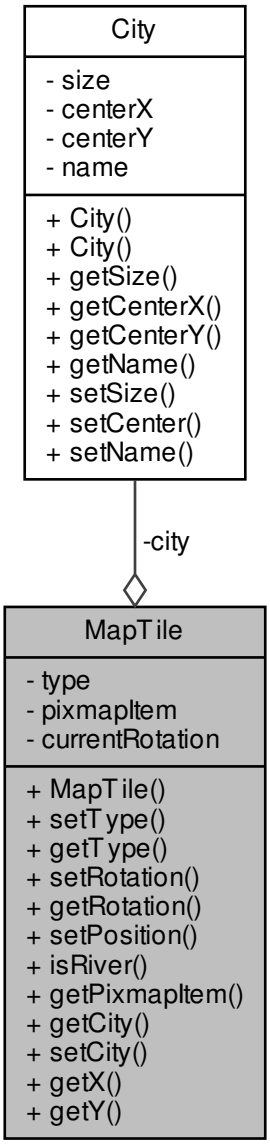
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/mainwindow.h
- src/application/mainwindow.cpp

## 7.5 MapTile Klassenreferenz

`#include <maptile.h>`

Zusammengehörigkeiten von MapTile:




### Öffentliche Typen

- enum TYPE {
  GRASS, FORREST, CITY, CITY_CENTER,
  RIVER_H, RIVER_V, RIVER_LB, RIVER_LT,
  RIVER_RT, RIVER_RB, RAIL_H, RAIL_CURVE }

## Öffentliche Methoden

- MapTile ()

  *MapTile::MapTile Konstruktor.*
- void setType (TYPE pType)

  *MapTile::setType Setzt den Typ der Kachel.*
- TYPE getType ()

  *MapTile::getType Liefert den Typ des Quadranten.*
- void setRotation (int pRotation)

  *MapTile::setRotation Hilfsfunktion zur Rotation im Quadrat.*
- int getRotation ()

  *MapTile::getRotation Liefert die aktuelle Rotation. (Himmelsrichtung)*
- void setPosition (int posX, int posY)

  *MapTile::setPosition Setzt die Position der Kachel. (In Pixeln)*
- bool isRiver ()

  *MapTile::isRiver Checkt ob die Kachel ein Fluss ist.*
- QGraphicsPixmapItem ∗ getPixmapItem ()

  *MapTile::getPixmapItem Liefert das Pixmap Item.*
- City ∗ getCity ()

  *MapTile::getCity Die Informationen. Falls keine Stadt: null.*
- void setCity (City ∗pCity)

  *MapTile::setCity.*
- int getX ()

  *MapTile::getX.*
- int getY ()

  *MapTile::getY.*

## Private Attribute

- TYPE type
- QGraphicsPixmapItem ∗ pixmapItem
- int currentRotation
- City ∗ city

### 7.5.1 Dokumentation der Aufzählungstypen

#### 7.5.1.1 TYPE

`enum MapTile::TYPE`

**Aufzählungswerte**

| | |
|---|---|
| GRASS | |
| FORREST | |
| CITY | |
| CITY_CENTER | |
| RIVER_H | |

**Aufzählungswerte**

| | |
|---|---|
| RIVER_V | |
| RIVER_LB | |
| RIVER_LT | |
| RIVER_RT | |
| RIVER_RB | |
| RAIL_H | |
| RAIL_CURVE | |

## 7.5.2 Beschreibung der Konstruktoren und Destruktoren

### 7.5.2.1 MapTile()

```
MapTile::MapTile ( )
```

MapTile::MapTile Konstruktor.

## 7.5.3 Dokumentation der Elementfunktionen

### 7.5.3.1 getCity()

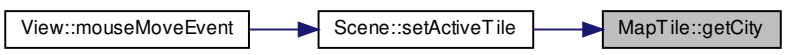
```
City * MapTile::getCity ( )
```

MapTile::getCity Die Informationen. Falls keine Stadt: null.

**Rückgabe**

Liefert die Informationen über eine Stadt auf der Kachel.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 7.5.3.2 getPixmapItem()

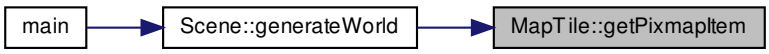
```
QGraphicsPixmapItem * MapTile::getPixmapItem ( )
```

MapTile::getPixmapItem Liefert das Pixmap Item.

**Rückgabe**

Das Pixmap Item.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.5.3.3 getRotation()

```
int MapTile::getRotation ( )
```

MapTile::getRotation Liefert die aktuelle Rotation. (Himmelsrichtung)

**Rückgabe**

Die aktuelle Rotation (0-3)

#### 7.5.3.4 getType()

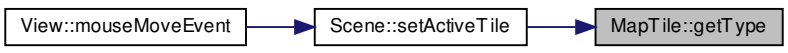
```
MapTile::TYPE MapTile::getType ( )
```

MapTile::getType Liefert den Typ des Quadranten.

**Rückgabe**

Den Typ.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 7.5.3.5 getX()

```
int MapTile::getX ( )
```

MapTile::getX.

**Rückgabe**

Der X Index des Quadranten.

#### 7.5.3.6 getY()

```
int MapTile::getY ( )
```

MapTile::getY.

**Rückgabe**

Der Y Index des Quadranten.

#### 7.5.3.7 isRiver()

```
bool MapTile::isRiver ( )
```

MapTile::isRiver Checkt ob die Kachel ein Fluss ist.

**Rückgabe**

Ob die Kachel ein Fluss ist.

#### 7.5.3.8 setCity()

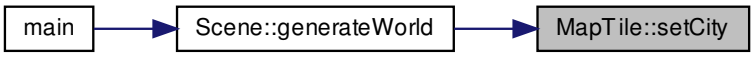
```
void MapTile::setCity (
            City * pCity )
```

MapTile::setCity.

**Parameter**

| | |
|---|---|
| *pCity* | Fügt dem Quadranten Daten über eine Stadt hinzu. |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.5.3.9 setPosition()

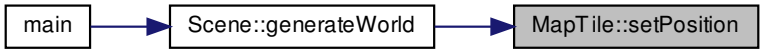
```
void MapTile::setPosition (
            int posX,
            int posY )
```

MapTile::setPosition Setzt die Position der Kachel. (In Pixeln)

**Parameter**

| | |
|---|---|
| *posX* | Die X Koordinate. |
| *posY* | Die Y Koordinate. |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.5.3.10 setRotation()

```
void MapTile::setRotation (
            int pRotation )
```

MapTile::setRotation Hilfsfunktion zur Rotation im Quadrat.

**Parameter**

| | |
|---|---|
| *pRotation* | 0=Ursprung 1=90° Grad 2=180° Grad 3=270° |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:


#### 7.5.3.11 setType()

```
void MapTile::setType (
            MapTile::TYPE pType )
```

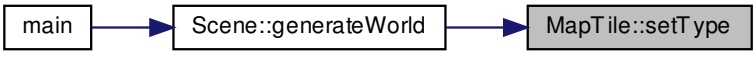
MapTile::setType Setzt den Typ der Kachel.

**Parameter**

| | |
|---|---|
| *pType* | Der Typ. |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:


Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




### 7.5.4 Dokumentation der Datenelemente

#### 7.5.4.1 city

```
City* MapTile::city  [private]
```

#### 7.5.4.2 currentRotation

```
int MapTile::currentRotation  [private]
```

#### 7.5.4.3 pixmapItem

```
QGraphicsPixmapItem* MapTile::pixmapItem  [private]
```

#### 7.5.4.4 type

```
TYPE MapTile::type  [private]
```

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/maptile.h
- src/application/maptile.cpp

## 7.6 River Klassenreferenz

```
#include <river.h>
```

Zusammengehörigkeiten von River:

### Öffentliche Methoden

- River ()

### 7.6.1 Beschreibung der Konstruktoren und Destruktoren

#### 7.6.1.1 River()

```
River::River ( )
```

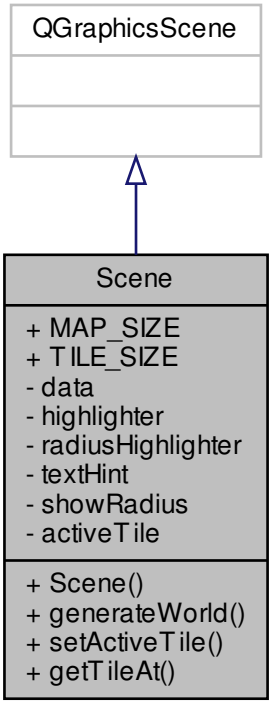
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/river.h
- src/application/river.cpp

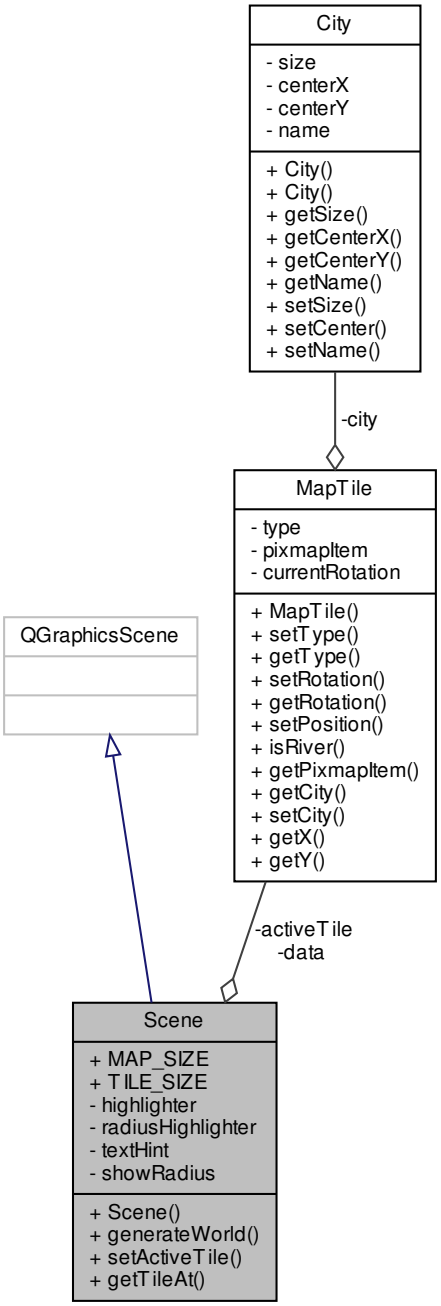
## 7.7 Scene Klassenreferenz

```
#include <scene.h>
```

Klassendiagramm für Scene:



Zusammengehörigkeiten von Scene:




## Öffentliche Methoden

- Scene ()

  *Scene::Scene Konstruktor.*
- void generateWorld ()

  *Scene::generateWorld Diese Methode generiert eine neue Welt.*
- void setActiveTile (QGraphicsItem ∗pItem)

*Scene::setActiveTile Setzt den MapTile über dem die Maus gerade ist. Wird von view aufgerufen.*

- MapTile * getTileAt (int posX, int posY, bool isPixelCoordinate)

  *Scene::getTileAt Liefert ein MapTile anhand der Pixel-Koordinaten.*

## Statische öffentliche Attribute

- const static int MAP_SIZE {300}
- const static int TILE_SIZE {64}

## Private Attribute

- MapTile data [Scene::MAP_SIZE][Scene::MAP_SIZE]
- QGraphicsRectItem * highlighter
- QGraphicsEllipseItem * radiusHighlighter
- QGraphicsTextItem * textHint
- bool showRadius
- MapTile * activeTile

### 7.7.1 Beschreibung der Konstruktoren und Destruktoren

#### 7.7.1.1 Scene()

```
Scene::Scene ( )
```

Scene::Scene Konstruktor.

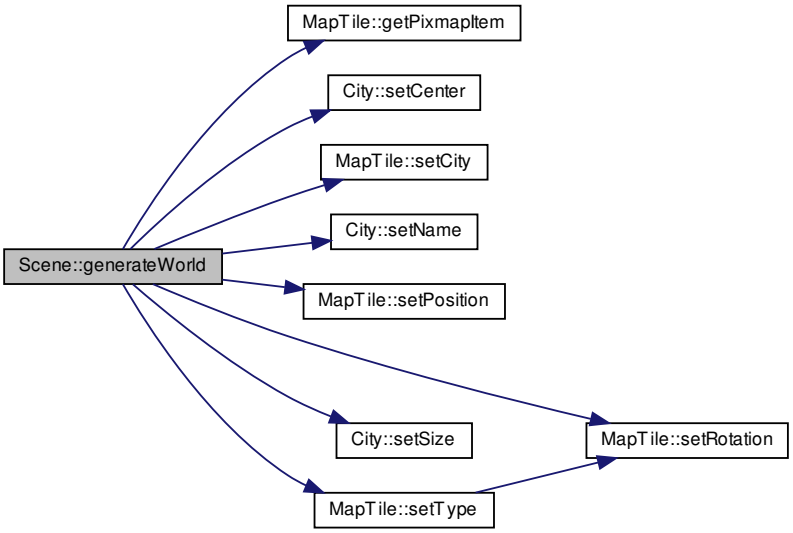
### 7.7.2 Dokumentation der Elementfunktionen

#### 7.7.2.1 generateWorld()

```
void Scene::generateWorld ( )
```

Scene::generateWorld Diese Methode generiert eine neue Welt.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:




Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:


#### 7.7.2.2 getTileAt()

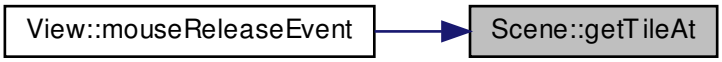
```
MapTile * Scene::getTileAt (
            int posX,
            int posY,
            bool isPixelCoordinate )
```

Scene::getTileAt Liefert ein MapTile anhand der Pixel-Koordinaten.

**Parameter**

| | |
|---|---|
| *posX* | Die X-Koordinate |
| *posY* | Die Y-Koordinate |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.7.2.3 setActiveTile()

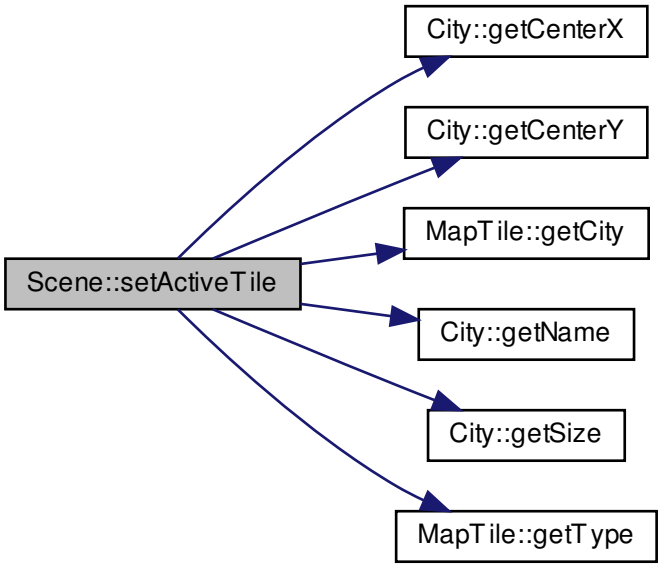
```
void Scene::setActiveTile (
            QGraphicsItem * pItem )
```

Scene::setActiveTile Setzt den MapTile über dem die Maus gerade ist. Wird von view aufgerufen.

**Parameter**

| | |
|---|---|
| *pItem* | Ein Grafikitem zu dem die Methode den zugehörigen Maptile bestimmt. |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:


### 7.7.3 Dokumentation der Datenelemente

#### 7.7.3.1 activeTile

```
MapTile* Scene::activeTile  [private]
```

#### 7.7.3.2 data

```
MapTile Scene::data[Scene::MAP_SIZE][Scene::MAP_SIZE]  [private]
```

#### 7.7.3.3 highlighter

```
QGraphicsRectItem* Scene::highlighter  [private]
```

#### 7.7.3.4 MAP_SIZE

```
const static int Scene::MAP_SIZE {300}  [static]
```

#### 7.7.3.5 radiusHighlighter

```
QGraphicsEllipseItem* Scene::radiusHighlighter  [private]
```

#### 7.7.3.6 showRadius

```
bool Scene::showRadius  [private]
```

#### 7.7.3.7 textHint

```
QGraphicsTextItem* Scene::textHint  [private]
```

#### 7.7.3.8 TILE_SIZE

```
const static int Scene::TILE_SIZE {64}  [static]
```

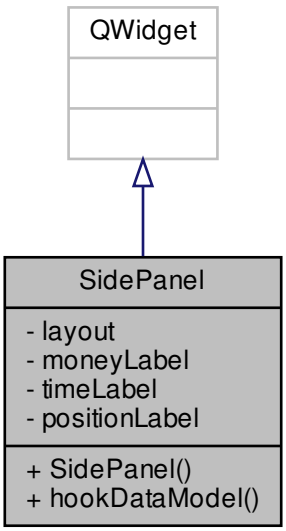
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/scene.h
- src/application/scene.cpp

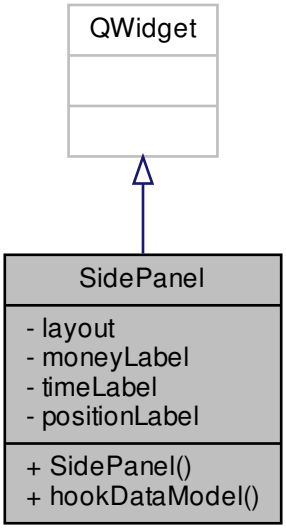
## 7.8 SidePanel Klassenreferenz

```
#include <sidepanel.h>
```

Klassendiagramm für SidePanel:



Zusammengehörigkeiten von SidePanel:




## Öffentliche Methoden

- SidePanel ()

  *SidePanel::SidePanel Erzeugt ein neues Side-Panel (Menü)*
- void hookDataModel (DataModel ∗pModel)

  *SidePanel::hookDataModel Verknüpft ein Datenmodell mit der Anzeige. Dadurch können dann Textfelder etc. aktualisiert werden.*

## Private Attribute

- QGridLayout ∗ layout
- QLabel ∗ moneyLabel
- QLabel ∗ timeLabel
- QLabel ∗ positionLabel

### 7.8.1 Beschreibung der Konstruktoren und Destruktoren

#### 7.8.1.1 SidePanel()

```
SidePanel::SidePanel ( )
```

SidePanel::SidePanel Erzeugt ein neues Side-Panel (Menü)

**Parameter**

| | |
|---|---|
| *pParent* | Das Parent-Element |

## 7.8.2 Dokumentation der Elementfunktionen

### 7.8.2.1 hookDataModel()

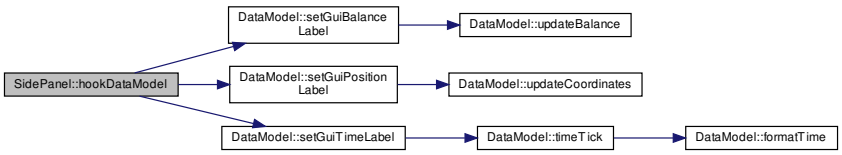
```
void SidePanel::hookDataModel (
            DataModel * pModel )
```

SidePanel::hookDataModel Verknüpft ein Datenmodell mit der Anzeige. Dadurch können dann Textfelder etc. aktualisiert werden.

**Parameter**

| | |
|---|---|
| *pModel* | Ein Datenmodell. |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:




Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:


## 7.8.3 Dokumentation der Datenelemente

#### 7.8.3.1 layout

```
QGridLayout* SidePanel::layout  [private]
```

#### 7.8.3.2 moneyLabel

```
QLabel* SidePanel::moneyLabel  [private]
```

#### 7.8.3.3 positionLabel

```
QLabel* SidePanel::positionLabel  [private]
```

#### 7.8.3.4 timeLabel

```
QLabel* SidePanel::timeLabel  [private]
```

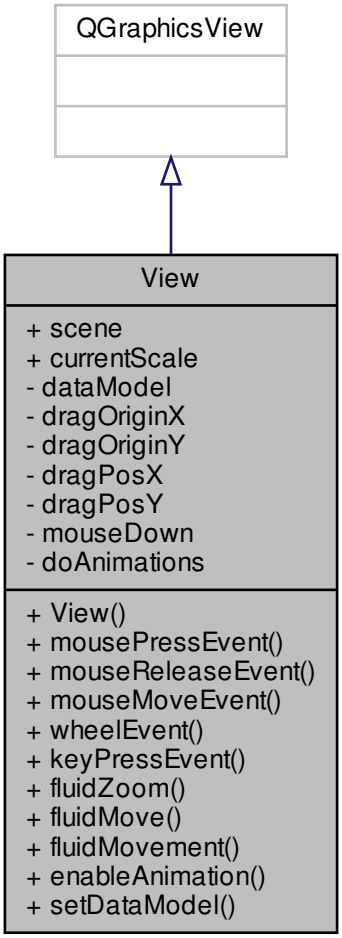
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/sidepanel.h
- src/application/sidepanel.cpp

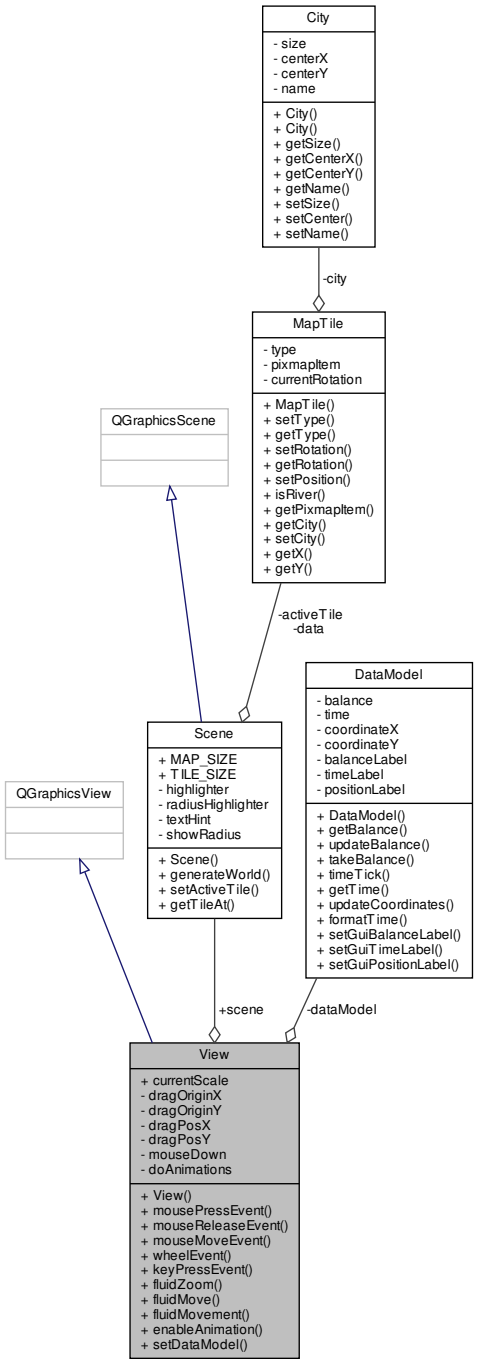
## 7.9 View Klassenreferenz

```
#include <view.h>
```

Klassendiagramm für View:



Zusammengehörigkeiten von View:




## Öffentliche Methoden

- View (Scene ∗pScene)

  *View::View Konstruktor. Versteckt u.a. die Scrollbars und aktiviert Mousetracking.*
- void mousePressEvent (QMouseEvent ∗event) override

  *View::mousePressEvent QT Methode. Wird aufgerufen wenn die Maus gedrückt wurde.*
- void mouseReleaseEvent (QMouseEvent ∗event) override

*View::mouseReleaseEvent QT Methode. Wird aufgerufen wenn die Maus losgelassen wird.*

- void mouseMoveEvent (QMouseEvent ∗event) override

  *View::mouseMoveEvent QT Methode. Wird aufgerufen wenn die Maus bewegt wird.*
- void wheelEvent (QWheelEvent ∗event) override

  *View::wheelEvent QT Methode. Wird aufgerufen wenn das Mausrad gedreht wird.*
- void keyPressEvent (QKeyEvent ∗event) override

  *View::keyPressEvent QT Methode. Wird Aufgerufen wenn eine Taste gedrückt wird.*
- void fluidZoom (double target, bool in)

  *View::fluidZoom Startet eine Zoom-Animation. Zuvor muss doAnimations=true gesetzt sein. Bsp: fluidZoom(3, true) zoomt 3x in die Karte hinein.*
- void fluidMove (int vX, int vY)

  *View::fluidMove Verschiebt die Karte animiert und relativ zur aktuellen Position.*
- void fluidMovement (int pX, int pY)

  *View::fluidMovement Verschiebt die Karte animiert an zu einer absoluten Koordinate.*
- void enableAnimation ()

  *View::enableAnimation Aktiviert animationen bis zum nächsten Event.*
- void setDataModel (DataModel ∗pModel)

  *View::setDataModel Setzt das Datenmodell. An dieses wird dann kontinuierlich die aktuelle Position weitergegeben.*

## Öffentliche Attribute

- Scene ∗ scene
- double currentScale {1.0}

## Private Attribute

- DataModel ∗ dataModel
- int dragOriginX
- int dragOriginY
- int dragPosX
- int dragPosY
- bool mouseDown
- bool doAnimations

### 7.9.1 Beschreibung der Konstruktoren und Destruktoren

#### 7.9.1.1 View()

```
View::View (
            Scene * pScene )
```

View::View Konstruktor. Versteckt u.a. die Scrollbars und aktiviert Mousetracking.

**Parameter**

| | |
|---|---|
| *pScene* | Das Zugehörige Szenenobjekt. |

### 7.9.2 Dokumentation der Elementfunktionen

#### 7.9.2.1 enableAnimation()

```
void View::enableAnimation ( )
```

View::enableAnimation Aktiviert animationen bis zum nächsten Event.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:


#### 7.9.2.2 fluidMove()

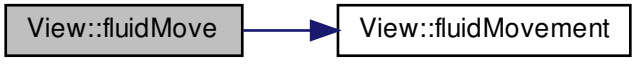
```
void View::fluidMove (
            int vX,
            int vY )
```

View::fluidMove Verschiebt die Karte animiert und relativ zur aktuellen Position.

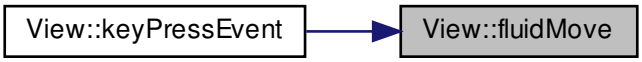
**Parameter**

| | |
|---|---|
| *vX* | Verschiebung in X-Richtung. |
| *vY* | Verschiebung in Y-Richtung. |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.9.2.3 fluidMovement()

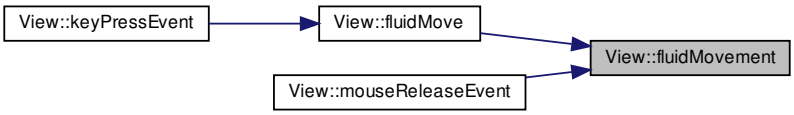
```
void View::fluidMovement (
            int pX,
            int pY )
```

View::fluidMovement Verschiebt die Karte animiert an zu einer absoluten Koordinate.

**Parameter**

| | |
|---|---|
| *pX* | Die X-Koordinate. |
| *pY* | Due Y-Koordinate. |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.9.2.4 fluidZoom()

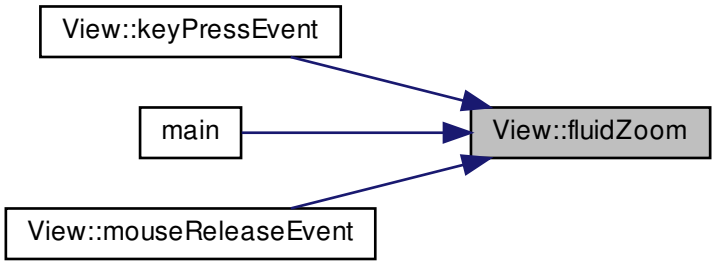
```
void View::fluidZoom (
            double target,
            bool in )
```

View::fluidZoom Startet eine Zoom-Animation. Zuvor muss doAnimations=true gesetzt sein. Bsp: fluidZoom(3, true) zoomt 3x in die Karte hinein.

**Parameter**

| | |
|---|---|
| *target* | Die angestrebte Skalierung. |
| *in* | Ob vergrößert oder verkleindert werden soll. (true = reinzoomen, false=rauszoomen). |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




#### 7.9.2.5 keyPressEvent()

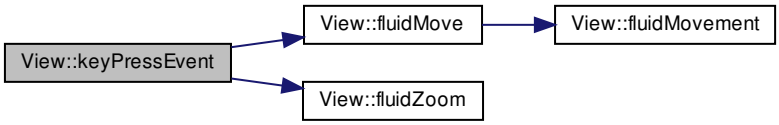
```
void View::keyPressEvent (
            QKeyEvent * event )  [override]
```

View::keyPressEvent QT Methode. Wird Aufgerufen wenn eine Taste gedrückt wird.

**Parameter**

| | |
|---|---|
| *event* | Event mit Informationen. Wichtig: event->text(): Text der Taste und event->key(): Id der Taste |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



#### 7.9.2.6 mouseMoveEvent()

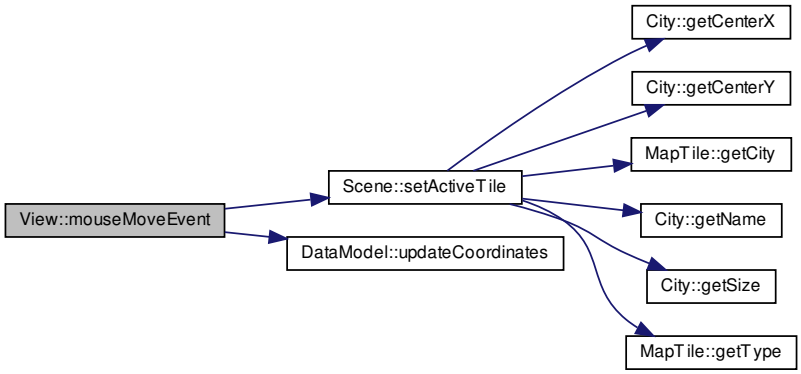
```
void View::mouseMoveEvent (
            QMouseEvent * event )  [override]
```

View::mouseMoveEvent QT Methode. Wird aufgerufen wenn die Maus bewegt wird.

**Parameter**

| | |
|---|---|
| *event* | Informationen über Position der Maus |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:




#### 7.9.2.7 mousePressEvent()

```
void View::mousePressEvent (
            QMouseEvent * event )  [override]
```

View::mousePressEvent QT Methode. Wird aufgerufen wenn die Maus gedrückt wurde.

**Parameter**

| | |
|---|---|
| *event* | Enthält Informationen über die Taste und Position. |

#### 7.9.2.8 mouseReleaseEvent()

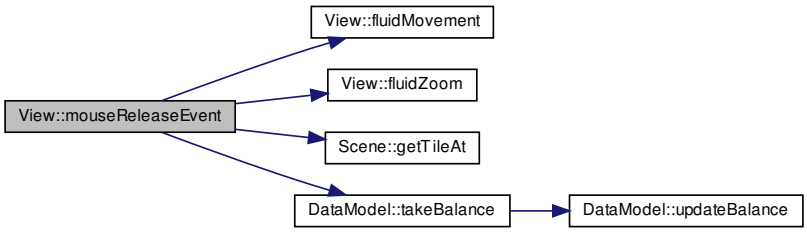
```
void View::mouseReleaseEvent (
            QMouseEvent * event )  [override]
```

View::mouseReleaseEvent QT Methode. Wird aufgerufen wenn die Maus losgelassen wird.

**Parameter**

| | |
|---|---|
| *event* | Informationen über Position und Taste |

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:




#### 7.9.2.9 setDataModel()

```
void View::setDataModel (
            DataModel * pModel )
```

View::setDataModel Setzt das Datenmodell. An dieses wird dann kontinuierlich die aktuelle Position weitergegeben.

**Parameter**

| | |
|---|---|
| *pModel* | Ein Datenmodell. |

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:


#### 7.9.2.10 wheelEvent()

```
void View::wheelEvent (
            QWheelEvent * event )  [override]
```

View::wheelEvent QT Methode. Wird aufgerufen wenn das Mausrad gedreht wird.

**Parameter**

| | |
|---|---|
| *event* | Eventobjekt mit Infos. Wichtig: event->delta(): Positiv oder negativ jenachdem in welche Richtung gedreht wurde. |

## 7.9.3 Dokumentation der Datenelemente

### 7.9.3.1 currentScale

```
double View::currentScale {1.0}
```

### 7.9.3.2 dataModel

```
DataModel* View::dataModel  [private]
```

### 7.9.3.3 doAnimations

```
bool View::doAnimations  [private]
```

### 7.9.3.4 dragOriginX

```
int View::dragOriginX  [private]
```

### 7.9.3.5 dragOriginY

```
int View::dragOriginY  [private]
```

### 7.9.3.6 dragPosX

```
int View::dragPosX  [private]
```

#### 7.9.3.7 dragPosY

```
int View::dragPosY  [private]
```

#### 7.9.3.8 mouseDown

```
bool View::mouseDown  [private]
```

#### 7.9.3.9 scene

```
Scene* View::scene
```

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/view.h
- src/application/view.cpp

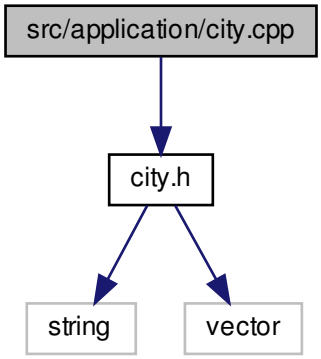
# Kapitel 8

# Datei-Dokumentation

## 8.1 src/application/city.cpp-Dateireferenz

```
#include "city.h"
```

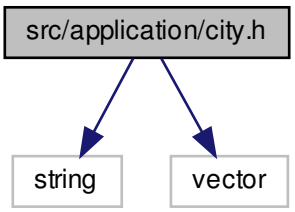
Include-Abhängigkeitsdiagramm für city.cpp:



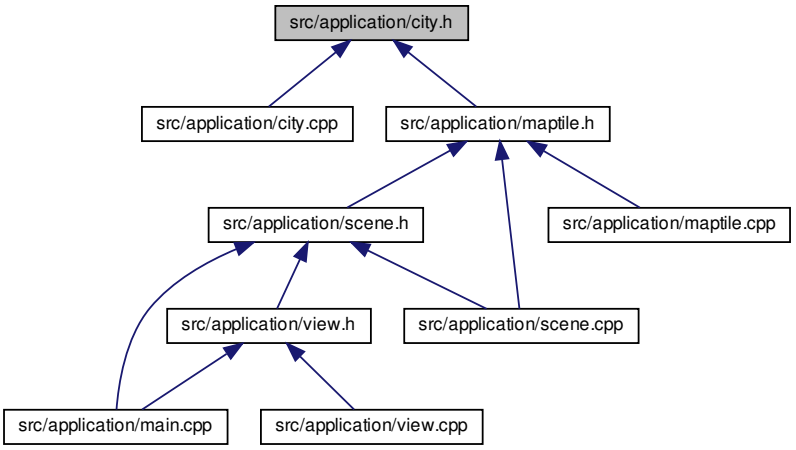

## 8.2 src/application/city.h-Dateireferenz

```
#include <string>
#include <vector>
```

Include-Abhängigkeitsdiagramm für city.h:




Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



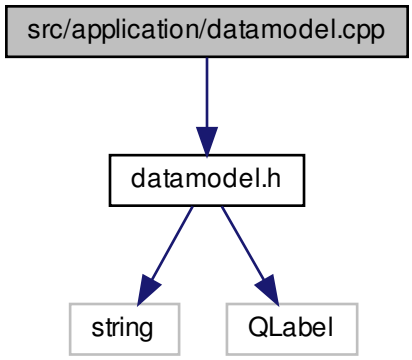

### Klassen

- class City

## 8.3 src/application/datamodel.cpp-Dateireferenz

```
#include "datamodel.h"
```

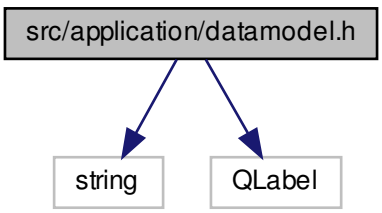
Include-Abhängigkeitsdiagramm für datamodel.cpp:



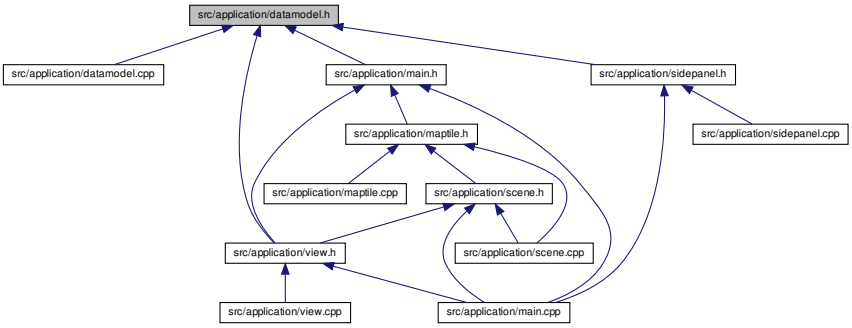

## 8.4 src/application/datamodel.h-Dateireferenz

```
#include <string>
#include <QLabel>
```

Include-Abhängigkeitsdiagramm für datamodel.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



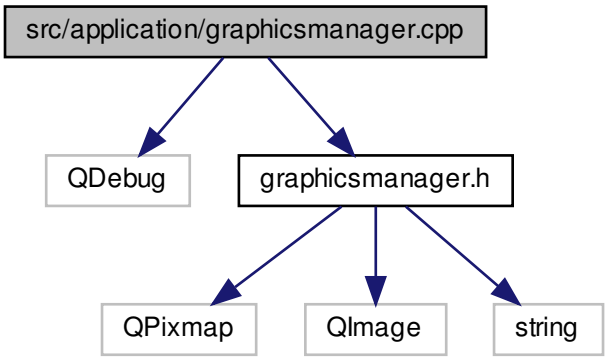

### Klassen

- class DataModel

## 8.5 src/application/graphicsmanager.cpp-Dateireferenz

```
#include <QDebug>
#include "graphicsmanager.h"
```

Include-Abhängigkeitsdiagramm für graphicsmanager.cpp:

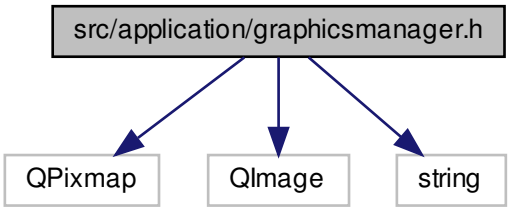



## 8.6 src/application/graphicsmanager.h-Dateireferenz

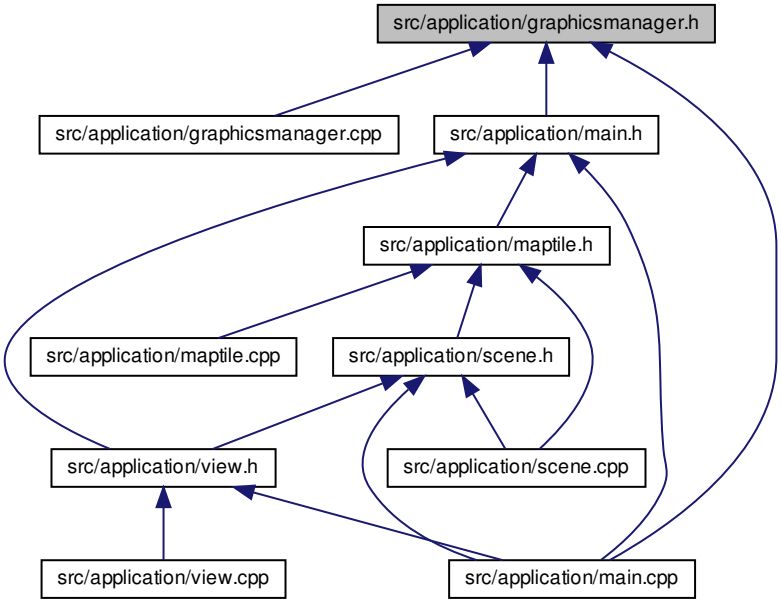
```
#include <QPixmap>
#include <QImage>
```

```
#include <string>
```

Include-Abhängigkeitsdiagramm für graphicsmanager.h:




Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:




### Klassen

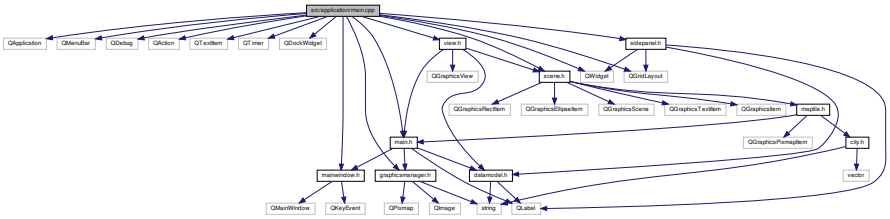
- class GraphicsManager

## 8.7 src/application/main.cpp-Dateireferenz

```
#include <QApplication>
#include <QMenuBar>
```

```
#include <QDebug>
#include <QAction>
#include <QTextItem>
#include <QTimer>
#include <QDockWidget>
#include <QWidget>
#include <QGridLayout>
#include "mainwindow.h"
#include "main.h"
#include "view.h"
#include "scene.h"
#include "graphicsmanager.h"
#include "sidepanel.h"
```

Include-Abhängigkeitsdiagramm für main.cpp:




## Funktionen

- void timeTicker ()
- int main (int argc, char ∗argv[ ])

  *main Startmethode.*

## Variablen

- GraphicsManager ∗ graphics
- MainWindow ∗ mainWindow
- DataModel ∗ dataModel
- bool gameRunning = true
- View ∗ view
- Scene ∗ scene
- SidePanel ∗ sidePanel

### 8.7.1 Dokumentation der Funktionen

#### 8.7.1.1 main()

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

main Startmethode.

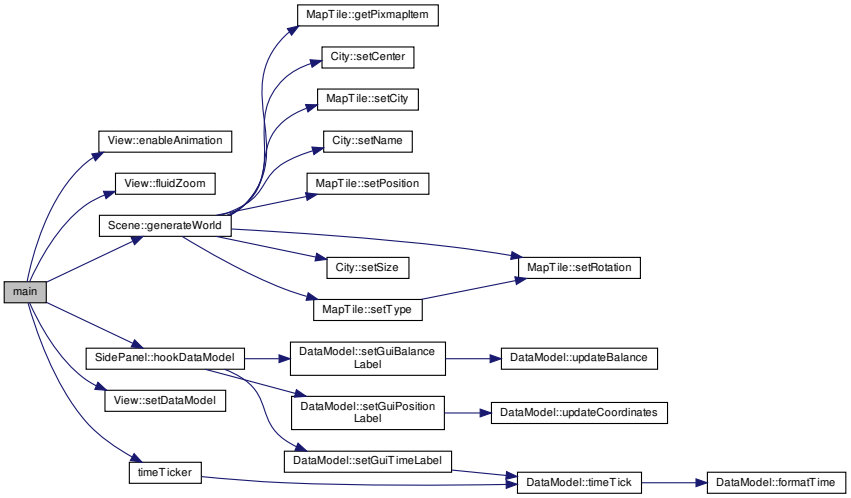
**Parameter**

| | |
|---|---|
| *argc* | Anzahl der Parameter |
| *argv* | Startparameter |

**Rückgabe**

Exit-Code (0=Alles gut)

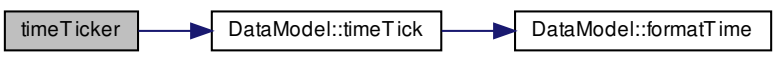
Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



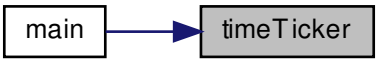

### 8.7.1.2 timeTicker()

```
void timeTicker ( )
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:




## 8.7.2 Variablen-Dokumentation

### 8.7.2.1 dataModel

```
DataModel* dataModel
```

### 8.7.2.2 gameRunning

```
bool gameRunning = true
```

### 8.7.2.3 graphics

```
GraphicsManager* graphics
```

### 8.7.2.4 mainWindow

```
MainWindow* mainWindow
```

### 8.7.2.5 scene

```
Scene* scene
```

#### 8.7.2.6 sidePanel

```
SidePanel* sidePanel
```

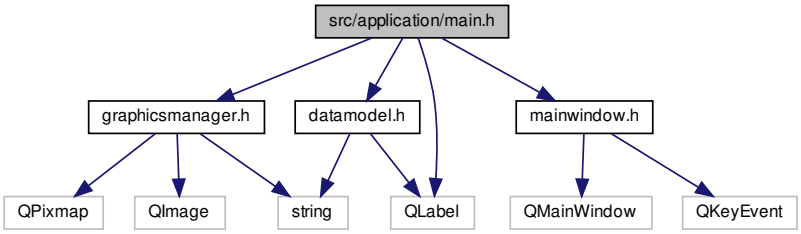
#### 8.7.2.7 view

```
View* view
```

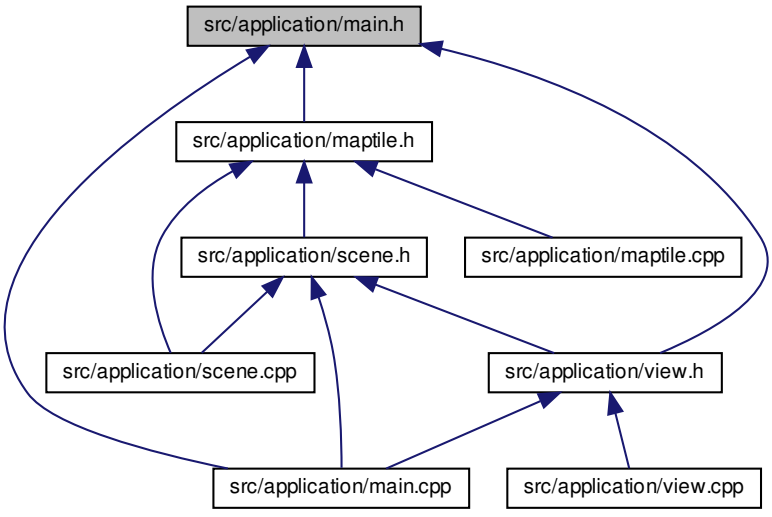
## 8.8 src/application/main.h-Dateireferenz

```
#include "graphicsmanager.h"
#include "mainwindow.h"
#include "datamodel.h"
#include <QLabel>
```

Include-Abhängigkeitsdiagramm für main.h:




Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



### Variablen

- GraphicsManager * graphics
- MainWindow * mainWindow
- bool gameRunning
- DataModel * dataModel

### 8.8.1 Variablen-Dokumentation

#### 8.8.1.1 dataModel

```
DataModel* dataModel
```

#### 8.8.1.2 gameRunning

```
bool gameRunning
```

#### 8.8.1.3 graphics

```
GraphicsManager* graphics
```

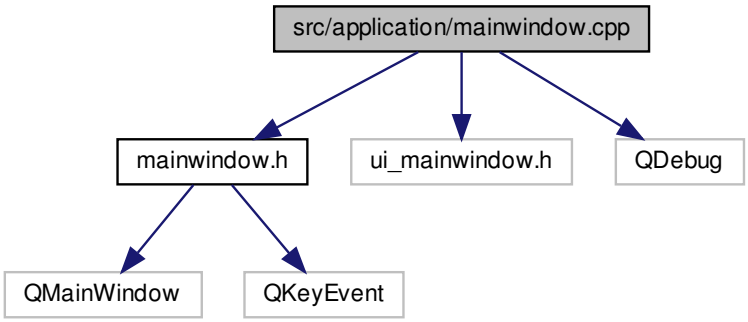
#### 8.8.1.4 mainWindow

```
MainWindow* mainWindow
```

## 8.9 src/application/mainwindow.cpp-Dateireferenz

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>
```

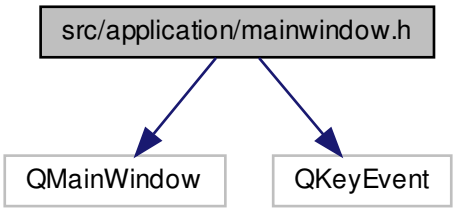
Include-Abhängigkeitsdiagramm für mainwindow.cpp:



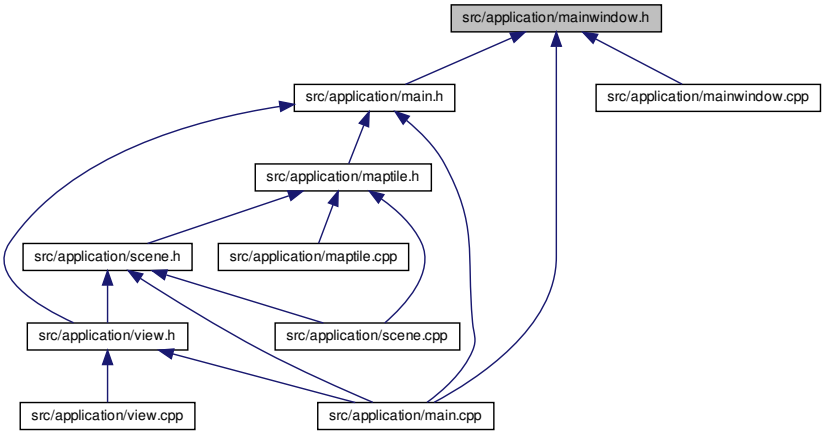
## 8.10 src/application/mainwindow.h-Dateireferenz

```
#include <QMainWindow>
#include <QKeyEvent>
```

Include-Abhängigkeitsdiagramm für mainwindow.h:




Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:




### Klassen

- class MainWindow

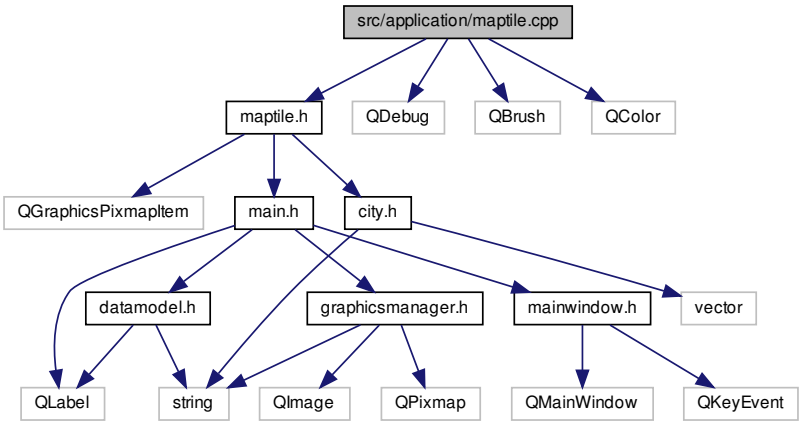
### Namensbereiche

- Ui

## 8.11 src/application/maptile.cpp-Dateireferenz

```
#include "maptile.h"
#include <QDebug>
#include <QBrush>
#include <QColor>
```

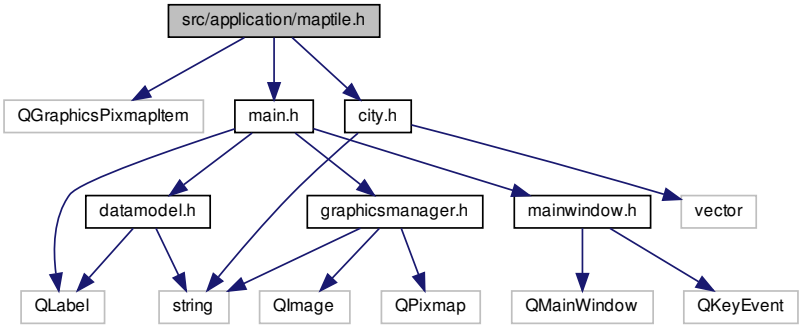
Include-Abhängigkeitsdiagramm für maptile.cpp:



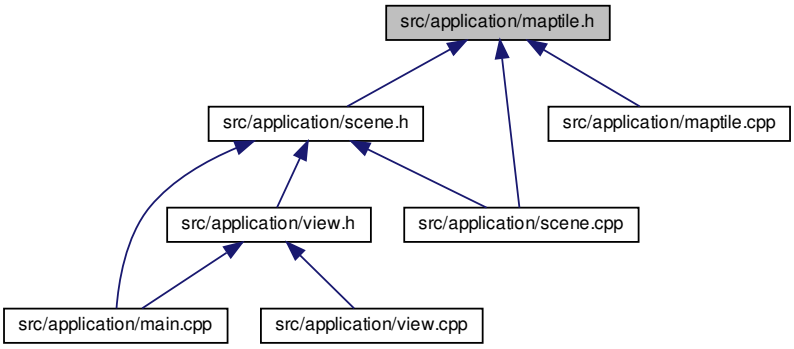

## 8.12 src/application/maptile.h-Dateireferenz

```
#include <QGraphicsPixmapItem>
#include "main.h"
#include "city.h"
```

Include-Abhängigkeitsdiagramm für maptile.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:




### Klassen

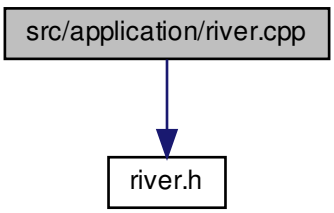
- class MapTile

## 8.13 src/application/README.md-Dateireferenz

## 8.14 src/application/river.cpp-Dateireferenz

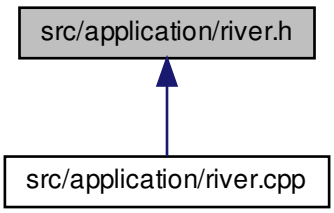
```
#include "river.h"
```

Include-Abhängigkeitsdiagramm für river.cpp:



## 8.15 src/application/river.h-Dateireferenz

Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



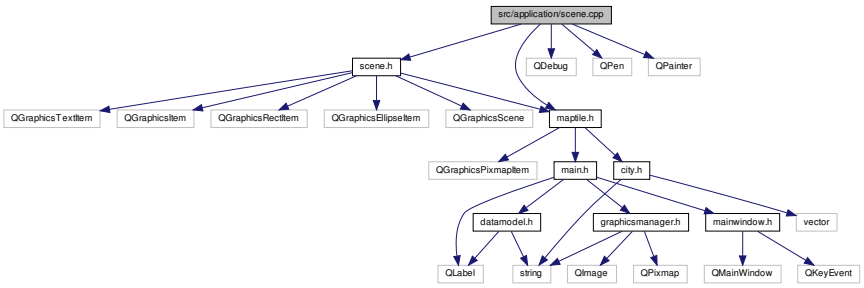

### Klassen

- class River

## 8.16 src/application/scene.cpp-Dateireferenz

```
#include "scene.h"
#include "maptile.h"
#include <QDebug>
#include <QPen>
#include <QPainter>
```

Include-Abhängigkeitsdiagramm für scene.cpp:

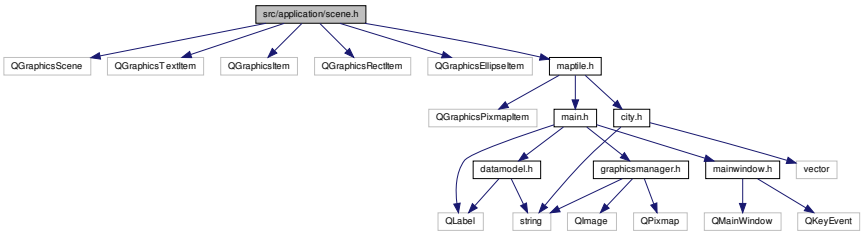



## 8.17 src/application/scene.h-Dateireferenz

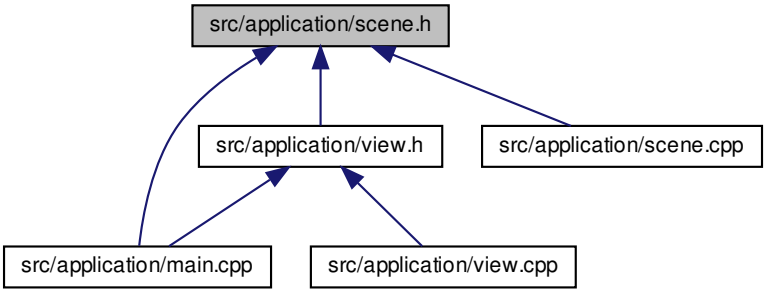
```
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include <QGraphicsItem>
#include <QGraphicsRectItem>
```

```
#include <QGraphicsEllipseItem>
#include "maptile.h"
```

Include-Abhängigkeitsdiagramm für scene.h:




Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



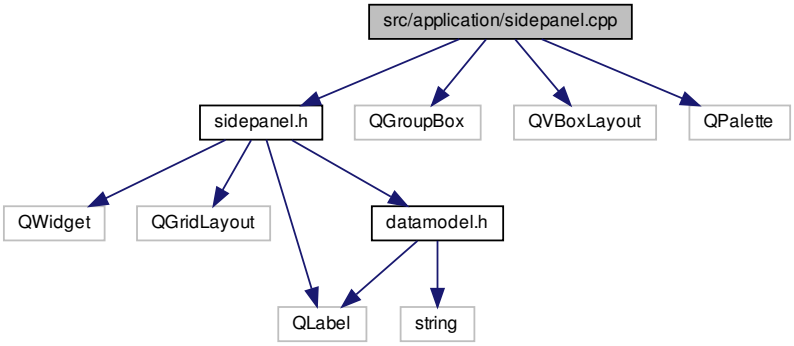

## Klassen

- class Scene

# 8.18 src/application/sidepanel.cpp-Dateireferenz

```
#include "sidepanel.h"
#include <QGroupBox>
#include <QVBoxLayout>
#include <QPalette>
```

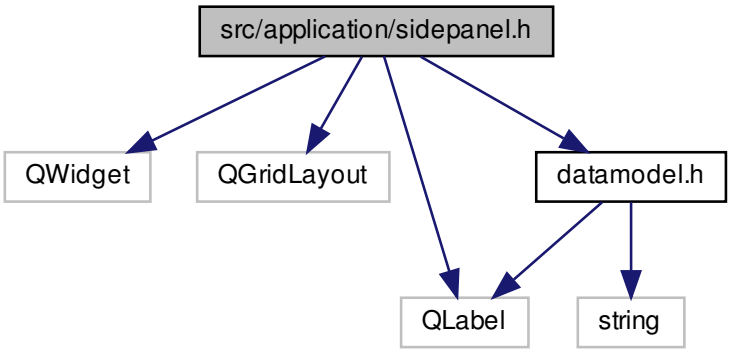
Include-Abhängigkeitsdiagramm für sidepanel.cpp:



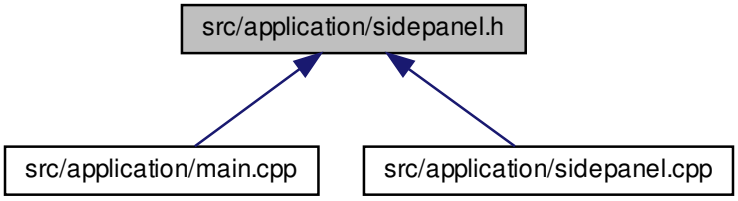

## 8.19 src/application/sidepanel.h-Dateireferenz

```
#include <QWidget>
#include <QGridLayout>
#include <QLabel>
#include "datamodel.h"
```

Include-Abhängigkeitsdiagramm für sidepanel.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



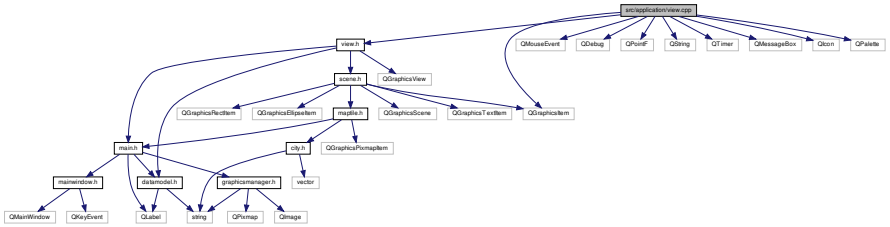

### Klassen

- class SidePanel

## 8.20 src/application/view.cpp-Dateireferenz

```
#include "view.h"
#include <QMouseEvent>
#include <QDebug>
#include <QPointF>
#include <QString>
#include <QGraphicsItem>
#include <QTimer>
#include <QMessageBox>
#include <QIcon>
#include <QPalette>
```

Include-Abhängigkeitsdiagramm für view.cpp:

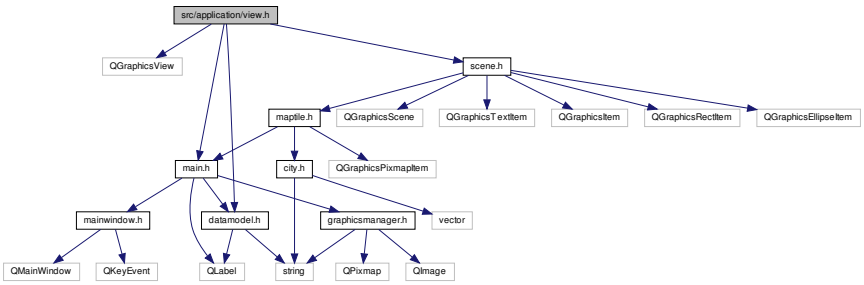



## 8.21 src/application/view.h-Dateireferenz

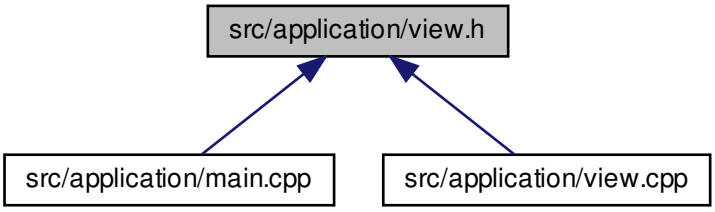
```
#include <QGraphicsView>
#include "main.h"
#include "scene.h"
```

```
#include "datamodel.h"
```

Include-Abhängigkeitsdiagramm für view.h:




Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:




## Klassen

- class View