

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
04.11.	Menübar hinzugefügt

Kapitel 2

Verzeichnis der Namensbereiche

2.1 Liste aller Namensbereiche

Liste aller Namensbereiche mit Kurzbeschreibung:

Ui ??

Kapitel 3

Hierarchie-Verzeichnis

3.1 Klassenhierarchie

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

City	??
GraphicsManager	??
MapTile	??
Player	??
QGraphicsScene	
Scene	??
QGraphicsView	
View	??
QMainWindow	
MainWindow	??
QMenuBar	
MenuBar	??
QObject	
DataModel	??
QThread	
Client	??
QWidget	
SidePanel	??
ToolTipMenu	??
River	??

Kapitel 4

Klassen-Verzeichnis

4.1 Auflistung der Klassen

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

City	??
Client	??
DataModel	??
GraphicsManager	??
MainWindow	??
MapTile	??
MenuBar	??
Player	??
River	??
Scene	??
SidePanel	??
ToolTipMenu	??
View	??

Kapitel 5

Datei-Verzeichnis

5.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

src/application_server/city.cpp	??
src/application_server/city.h	??
src/application_server/client.cpp	??
src/application_server/client.h	??
src/application_server/datamodel.cpp	??
src/application_server/datamodel.h	??
src/application_server/graphicsmanager.cpp	??
src/application_server/graphicsmanager.h	??
src/application_server/main.cpp	??
src/application_server/main.h	??
src/application_server/mainwindow.cpp	??
src/application_server/mainwindow.h	??
src/application_server/maptile.cpp	??
src/application_server/maptile.h	??
src/application_server/menubar.cpp	??
src/application_server/menubar.h	??
src/application_server/player.cpp	??
src/application_server/player.h	??
src/application_server/river.cpp	??
src/application_server/river.h	??
src/application_server/scene.cpp	??
src/application_server/scene.h	??
src/application_server/sidepanel.cpp	??
src/application_server/sidepanel.h	??
src/application_server/tooltipmenu.cpp	??
src/application_server/tooltipmenu.h	??
src/application_server/view.cpp	??
src/application_server/view.h	??

Kapitel 6

Dokumentation der Namensbereiche

6.1 Ui-Namensbereichsreferenz

Kapitel 7

Klassen-Dokumentation

7.1 City Klassenreferenz

```
#include <city.h>
```

Zusammengehörigkeiten von City:

City
- size - centerX - centerY - name
+ City() + City() + getSize() + getCenterX() + getCenterY() + getName() + setSize() + setCenter() + setName()

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::getCenterY` 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

<code>pX</code>	Der X-Index des Mittelpunktes.
<code>pY</code>	Der Y-Index des Mittelpunktes.
<code>pSize</code>	Die gröÙe 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()

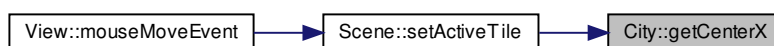
```
int City::getCenterX ( )
```

[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()

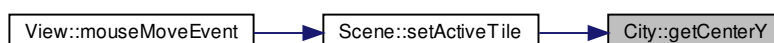
```
int City::getCenterY ( )
```

[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

<i>pX</i>	Der X-Index.
<i>pY</i>	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

<i>pName</i>	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

<i>pSize</i>	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]
```

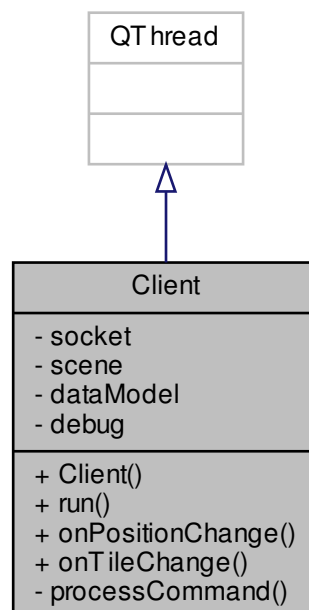
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/city.h](#)
- [src/application_server/city.cpp](#)

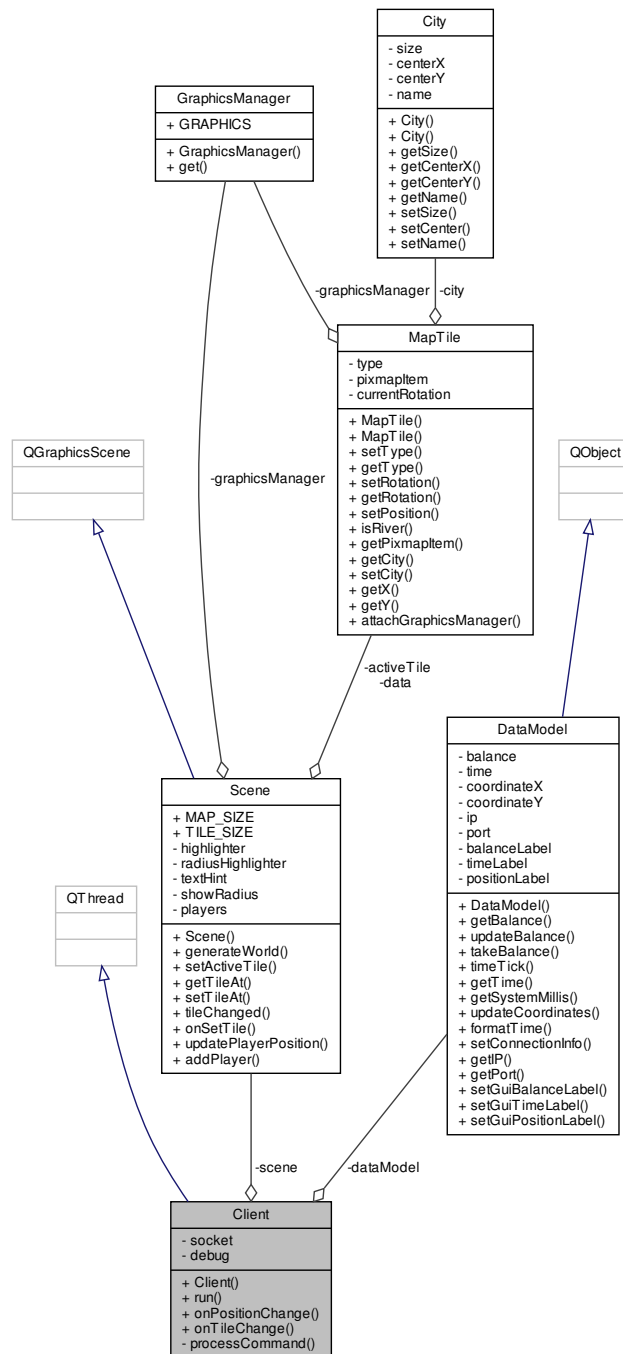
7.2 Client Klassenreferenz

```
#include <client.h>
```

Klassendiagramm für Client:



Zusammengehörigkeiten von Client:



Öffentliche Slots

- void `onPositionChange` (int, int)
Client::onPositionChange Slot für Ändern der Position.
- void `onTileChange` (int, int, int, int)
Client::onTileChange Slot für Ändern eines Tiles.

Signale

- void `mapLoaded` ()
- void `tileChanged` (int, int, int, int)
- void `playerConnect` (int)
- void `playerPositionChange` (int, int, int)

Öffentliche Methoden

- `Client` (QString *connectionInfo, `Scene` *pScene, `DataModel` *pDataModel)
`Client::Client` Erzeugt einen neuen `Client`.
- void `run` () override
`Client::run` Startet den `Client-Thread`.

Private Methoden

- void `processCommand` (QString command)
`Client::processCommand` Führt einen empfangenen Befehl aus dem Serverprotokoll aus.

Private Attribute

- QTcpSocket * `socket`
- `Scene` * `scene`
- `DataModel` * `dataModel`
- bool `debug`

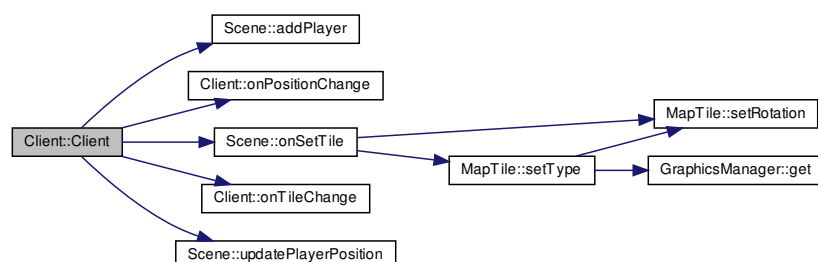
7.2.1 Beschreibung der Konstruktoren und Destruktoren

7.2.1.1 Client()

```
Client::Client (
    QString * connectionInfo,
    Scene * pScene,
    DataModel * pDataModel )
```

`Client::Client` Erzeugt einen neuen `Client`.

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

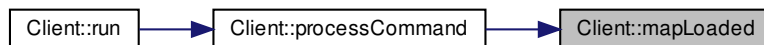


7.2.2 Dokumentation der Elementfunktionen

7.2.2.1 mapLoaded

```
void Client::mapLoaded ( ) [signal]
```

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



7.2.2.2 onPositionChange

```
void Client::onPositionChange (
    int pX,
    int pY ) [slot]
```

[Client::onPositionChange](#) Slot für Ändern der Position.

Parameter

<i>pX</i>	Der X-Index.
<i>pY</i>	Der Y-Index.

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



7.2.2.3 onTileChange

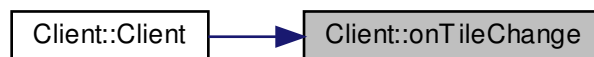
```
void Client::onTileChange (
    int pX,
    int pY,
    int pType,
    int pRotation ) [slot]
```

[Client::onTileChange](#) Slot für Ändern eines Tiles.

Parameter

<i>pX</i>	Der X-Index.
<i>pY</i>	Der Y-Index.
<i>pType</i>	Der Typ.
<i>pRotation</i>	Die Rotation.

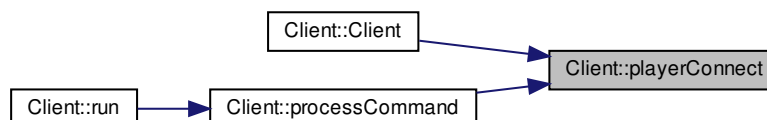
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.2.2.4 playerConnect

```
void Client::playerConnect (
    int ) [signal]
```

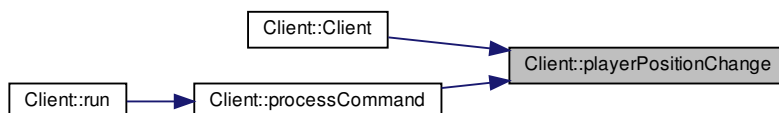
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.2.2.5 playerPositionChange

```
void Client::playerPositionChange (
    int ,
    int ,
    int ) [signal]
```

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



7.2.2.6 processCommand()

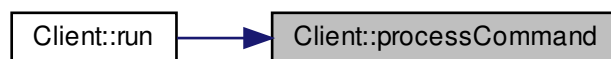
```
void Client::processCommand (
    QString cmd ) [private]
```

[Client::processCommand](#) Führt einen empfangenen Befehl aus dem Serverprotokoll aus.

Parameter

<i>cmd</i>	Der Befehl als String.
------------	------------------------

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

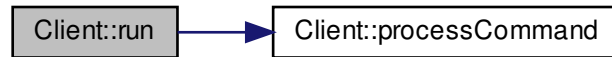


7.2.2.7 run()

```
void Client::run ( ) [override]
```

`Client::run` Startet den Client-Thread.

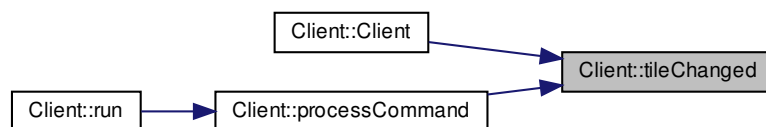
Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



7.2.2.8 tileChanged

```
void Client::tileChanged (
    int ,
    int ,
    int ,
    int ) [signal]
```

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



7.2.3 Dokumentation der Datenelemente

7.2.3.1 dataModel

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

7.2.3.2 debug

```
bool Client::debug [private]
```

7.2.3.3 scene

```
Scene* Client::scene [private]
```

7.2.3.4 socket

```
QTcpSocket* Client::socket [private]
```

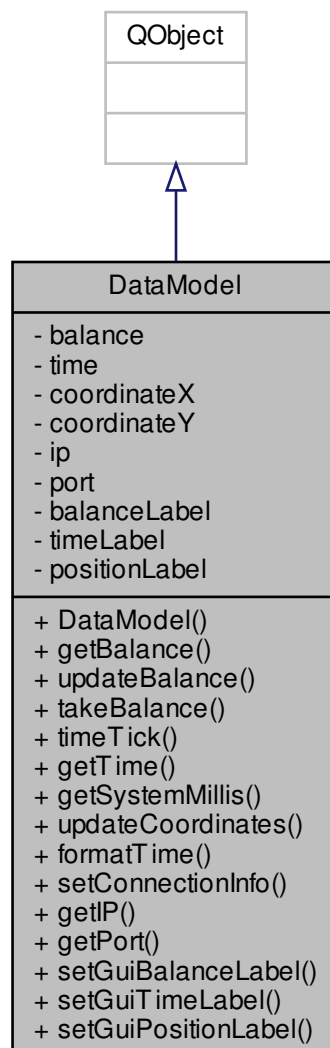
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application_server/[client.h](#)
- src/application_server/[client.cpp](#)

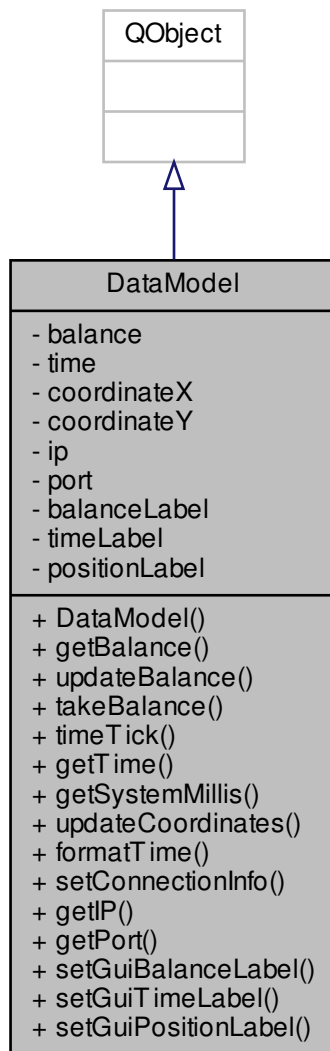
7.3 DataModel Klassenreferenz

```
#include <datamodel.h>
```

Klassendiagramm für DataModel:



Zusammengehörigkeiten von DataModel:



Signale

- void `positionChange` (int, int)

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)
- long *getSystemMillis* ()
 - DataModel::getSystemMillis* Gibt die Zahl der Millisekunden seit 1970 zurück.
- void *updateCoordinates* (int pX, int pY)
 - DataModel::updateCoordinates* Aktualisiert die Koordinaten des fokussierten Quadranten.
- std::string *formatTime* (long pTime)
 - DataModel::formatTime* Formattiert einen Timecode als String.
- void *setConnectionInfo* (QString pString)
 - DataModel::setConnectionInfo* Setzt die Verbindungsinformation als String.
- QString * *getIP* ()
 - DataModel::getIP* Gibt die IP Adresse zur Verbindung zurück.
- quint16 *getPort* ()
 - DataModel::getPort* Gibt den Port zur Verbindung zurück.
- 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*
- QString *ip*
- quint16 *port*
- QLabel * *balanceLabel*
- QLabel * *timeLabel*
- QLabel * *positionLabel*

7.3.1 Beschreibung der Konstruktoren und Destruktoren

7.3.1.1 DataModel()

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

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

7.3.2 Dokumentation der Elementfunktionen

7.3.2.1 formatTime()

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

[DataModel::formatTime](#) Formattiert einen Timecode als String.

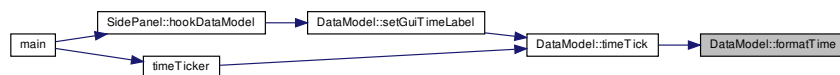
Parameter

<i>pTime</i>	Der Timecode.
--------------	---------------

Rückgabe

Der Text.

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



7.3.2.2 getBalance()

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

[DataModel::getBalance](#) Liefert den aktuellen Kontostand zurück.

Rückgabe

Der aktuelle Kontostand.

7.3.2.3 getIP()

```
QString * DataModel::getIP ( )
```

[DataModel::getIP](#) Gibt die IP Adresse zur Verbindung zurück.

Rückgabe

Die IP Adresse als QString

7.3.2.4 getPort()

```
quint16 DataModel::getPort ( )
```

[DataModel::getPort](#) Gibt den Port zur Verbindung zurück.

Rückgabe

Der Port als int.

7.3.2.5 getSystemMillis()

```
long DataModel::getSystemMillis ( )
```

[DataModel::getSystemMillis](#) Gibt die Zahl der Millisekunden seit 1970 zurück.

Rückgabe

Die Zahl der Millisekunden.

7.3.2.6 getTime()

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

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

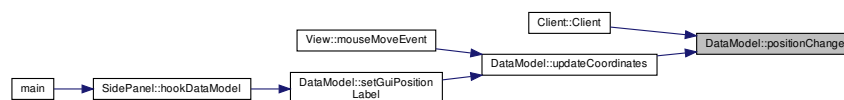
Rückgabe

Der Timecode.

7.3.2.7 positionChange

```
void DataModel::positionChange (
    int ,
    int ) [signal]
```

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



7.3.2.8 setConnectionInfo()

```
void DataModel::setConnectionInfo (
    QString pString )
```

[DataModel::setConnectionInfo](#) Setzt die Verbindungsinformation als String.

Parameter

<i>pString</i>	Die IP und der Port im Format IP:PORT
----------------	---------------------------------------

7.3.2.9 setGuiBalanceLabel()

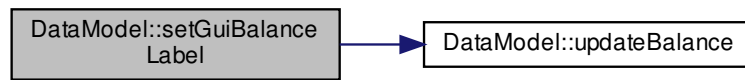
```
void DataModel::setGuiBalanceLabel (
    QLabel * label )
```

[DataModel::setGuiBalanceLabel](#) Setzt das Label in welchem der Kontostand dargestellt wird.

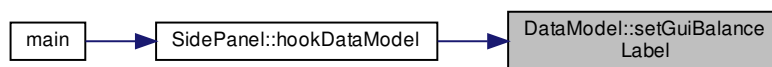
Parameter

<i>label</i>	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:



7.3.2.10 setGuiPositionLabel()

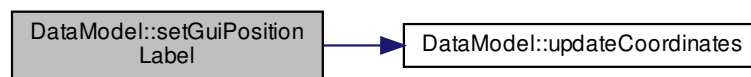
```
void DataModel::setGuiPositionLabel (
    QLabel * label )
```

[DataModel::setGuiTimeLabel](#) Setzt das Label in welchem die Koordinate dargestellt wird.

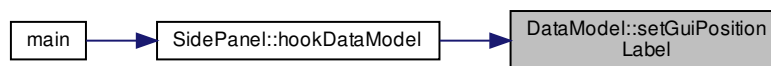
Parameter

<i>label</i>	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:



7.3.2.11 setGuiTimeLabel()

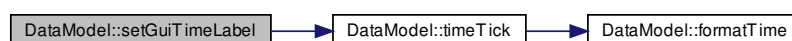
```
void DataModel::setGuiTimeLabel (
    QLabel * label )
```

[DataModel::setGuiTimeLabel](#) Setzt das Label in welchem die Zeit dargestellt wird.

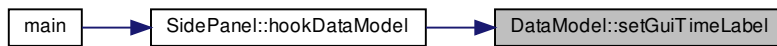
Parameter

<i>label</i>	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:



7.3.2.12 takeBalance()

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

[DataModel::takeBalance](#) Zieht Geld ab falls noch genug da ist.

Parameter

<i>pAmount</i>	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:



7.3.2.13 timeTick()

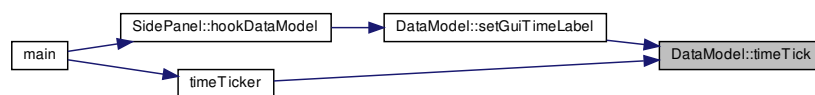
```
void DataModel::timeTick ( )
```

[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:



7.3.2.14 updateBalance()

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

[DataModel::updateBalance](#) Aktualisiert den Kontostand. Auch in Anzeigen etc.

Parameter

<i>pBalance</i>	Der neue Kontostand.
-----------------	----------------------

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



7.3.2.15 updateCoordinates()

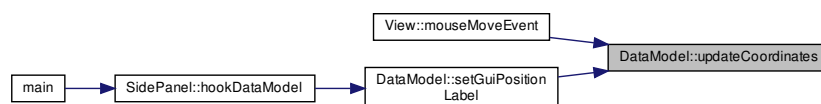
```
void DataModel::updateCoordinates (
    int pX,
    int pY )
```

[DataModel::updateCoordinates](#) Aktualisiert die Koordinaten des fokussierten Quadranten.

Parameter

<i>pX</i>	Die X Koordinate.
<i>pY</i>	Die Y Koordinate.

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



7.3.3 Dokumentation der Datenelemente

7.3.3.1 balance

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

7.3.3.2 balanceLabel

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

7.3.3.3 coordinateX

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


7.3.3.4 coordinateY

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

7.3.3.5 ip

```
QString DataModel::ip [private]
```

7.3.3.6 port

```
quint16 DataModel::port [private]
```

7.3.3.7 positionLabel

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

7.3.3.8 time

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

7.3.3.9 timeLabel

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

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/datamodel.h](#)
- [src/application_server/datamodel.cpp](#)

7.4 GraphicsManager Klassenreferenz

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

Zusammengehörigkeiten von GraphicsManager:

GraphicsManager
+ GRAPHICS
+ GraphicsManager() + get()

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](#)

7.4.1 Beschreibung der Konstruktoren und Destruktoren

7.4.1.1 GraphicsManager()

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

[GraphicsManager::GraphicsManager](#) Lädt alle Grafiken. Neue bitte im selben Stil ergänzen.

7.4.2 Dokumentation der Elementfunktionen

7.4.2.1 get()

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

[GraphicsManager::get](#) Liefert eine Grafik mit einem bestimmten Namen.

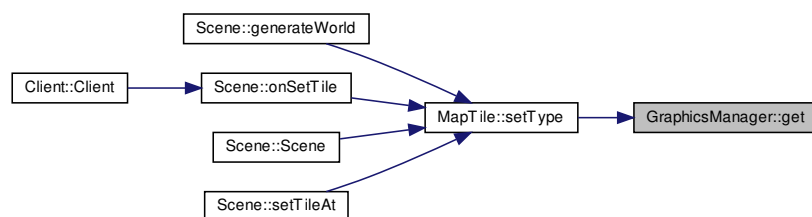
Parameter

<i>key</i>	Name der Grafik.
------------	------------------

Rückgabe

Die Grafik.

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



7.4.3 Dokumentation der Datenelemente

7.4.3.1 GRAPHICS

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

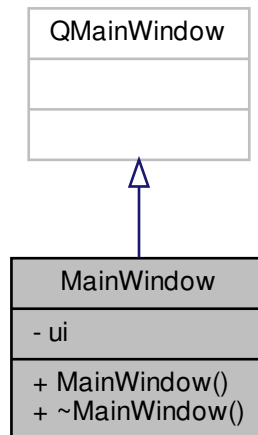
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/graphicsmanager.h](#)
- [src/application_server/graphicsmanager.cpp](#)

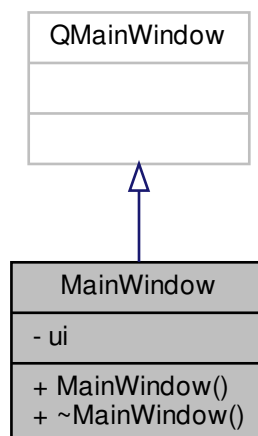
7.5 MainWindow Klassenreferenz

```
#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`

7.5.1 Beschreibung der Konstruktoren und Destruktoren

7.5.1.1 MainWindow()

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

[MainWindow::MainWindow.](#)

Parameter

<i>parent</i>	
---------------	--

7.5.1.2 ~MainWindow()

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

[MainWindow::~~MainWindow.](#)

7.5.2 Dokumentation der Datenelemente

7.5.2.1 ui

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

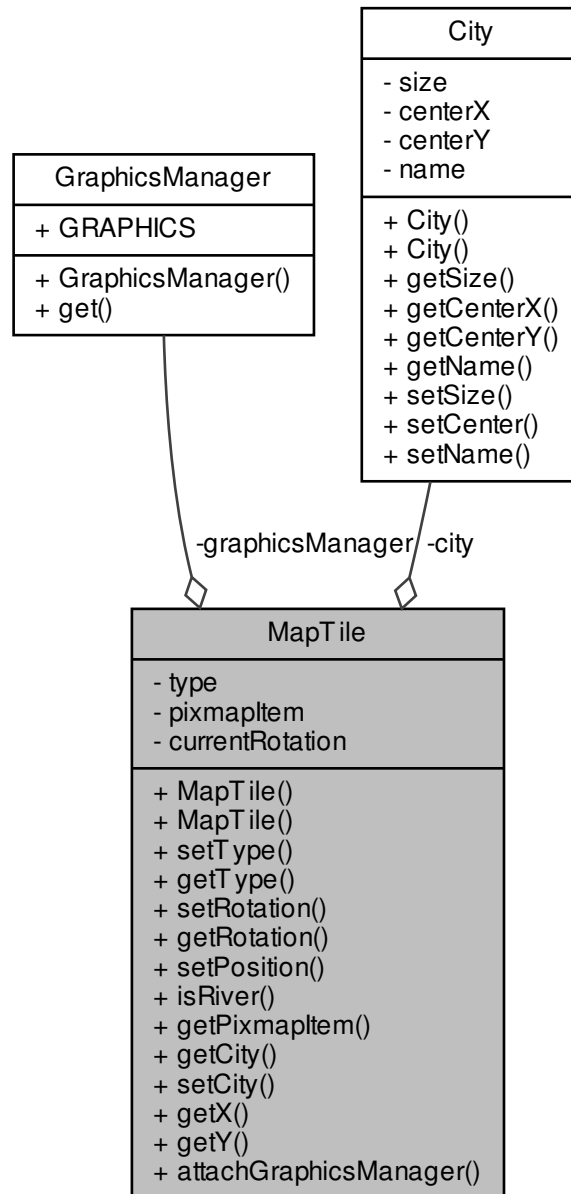
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/mainwindow.h](#)
- [src/application_server/mainwindow.cpp](#)

7.6 MapTile Klassenreferenz

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

Zusammengehörigkeiten von MapTile:



Öffentliche Typen

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

Öffentliche Methoden

- [MapTile](#) ([GraphicsManager](#) *pGraphicsManager)
MapTile::MapTile Konstruktor.
- [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.
- void [attachGraphicsManager](#) ([GraphicsManager](#) *pGraphicsManager)
MapTile::attachGraphicsManager Setzte den *GraphicsManager*.

Private Attribute

- [TYPE](#) type
- [QGraphicsPixmapItem](#) * pixmapItem
- int currentRotation
- [City](#) * city
- [GraphicsManager](#) * graphicsManager

7.6.1 Dokumentation der Aufzählungstypen

7.6.1.1 TYPE

```
enum MapTile::TYPE
```

Aufzählungswerte

GRASS	
FORREST	
CITY	
RIVER_H	
RIVER_V	
RIVER_LB	
RIVER_LT	
RIVER_RT	
RIVER_RB	
RAIL_H	
RAIL_CURVE	

7.6.2 Beschreibung der Konstruktoren und Destruktoren

7.6.2.1 MapTile() [1/2]

```
MapTile::MapTile (
    GraphicsManager * pGraphicsManager )
```

[MapTile::MapTile](#) Konstruktor.

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



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



7.6.2.2 MapTile() [2/2]

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

[MapTile::MapTile](#) Konstruktor.

7.6.3 Dokumentation der Elementfunktionen

7.6.3.1 attachGraphicsManager()

```
void MapTile::attachGraphicsManager (
    GraphicsManager * pGraphicsManager )
```

[MapTile::attachGraphicsManager](#) Setzte den [GraphicsManager](#).

Parameter

<i>pGraphicsManager</i>	Ein GraphicsManager .
-------------------------	---------------------------------------

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



7.6.3.2 getCity()

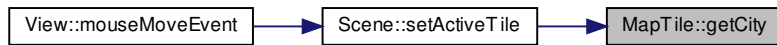
```
City * MapTile::getCity ( )
```

[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:



7.6.3.3 `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:



7.6.3.4 `getRotation()`

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

[MapTile::getRotation](#) Liefert die aktuelle Rotation. (Himmelsrichtung)

Rückgabe

Die aktuelle Rotation (0-3)

7.6.3.5 getType()

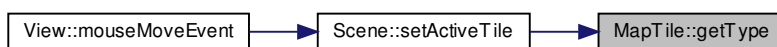
```
MapTile::TYPE MapTile::getType ( )
```

[MapTile::getType](#) Liefert den Typ des Quadranten.

Rückgabe

Den Typ.

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



7.6.3.6 getX()

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

[MapTile::getX](#).

Rückgabe

Der X Index des Quadranten.

7.6.3.7 getY()

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

[MapTile::getY](#).

Rückgabe

Der Y Index des Quadranten.

7.6.3.8 isRiver()

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

[MapTile::isRiver](#) Checkt ob die Kachel ein Fluss ist.

Rückgabe

Ob die Kachel ein Fluss ist.

7.6.3.9 setCity()

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

[MapTile::setCity](#).

Parameter

<i>pCity</i>	Fügt dem Quadranten Daten über eine Stadt hinzu.
--------------	--

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



7.6.3.10 setPosition()

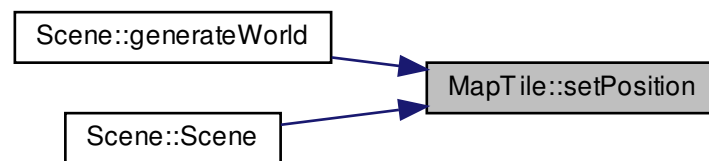
```
void MapTile::setPosition (
    int posX,
    int posY )
```

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

Parameter

<i>posX</i>	Die X Koordinate.
<i>posY</i>	Die Y Koordinate.

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



7.6.3.11 setRotation()

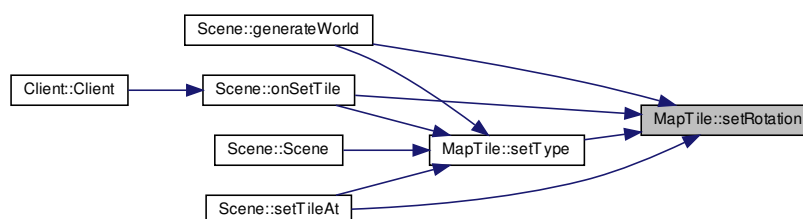
```
void MapTile::setRotation (
    int pRotation )
```

[MapTile::setRotation](#) Hilfsfunktion zur Rotation im Quadrat.

Parameter

<i>pRotation</i>	0=Ursprung 1=90° Grad 2=180° Grad 3=270°
------------------	--

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



7.6.3.12 setType()

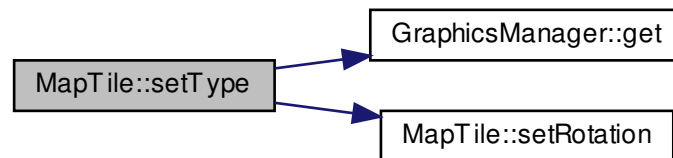
```
void MapTile::setType (
    MapTile::TYPE pType )
```

[MapTile::setType](#) Setzt den Typ der Kachel.

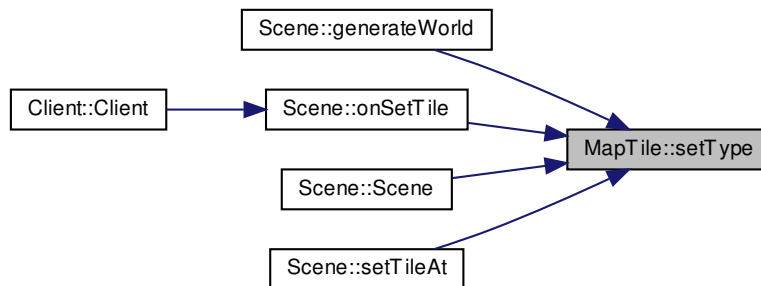
Parameter

<i>pType</i>	Der Typ.
--------------	----------

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



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



7.6.4 Dokumentation der Datenelemente

7.6.4.1 city

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

7.6.4.2 currentRotation

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

7.6.4.3 graphicsManager

```
GraphicsManager* MapTile::graphicsManager [private]
```

7.6.4.4 pixmapItem

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

7.6.4.5 type

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

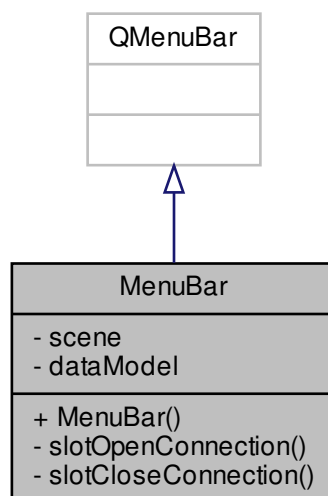
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/maptile.h](#)
- [src/application_server/maptile.cpp](#)

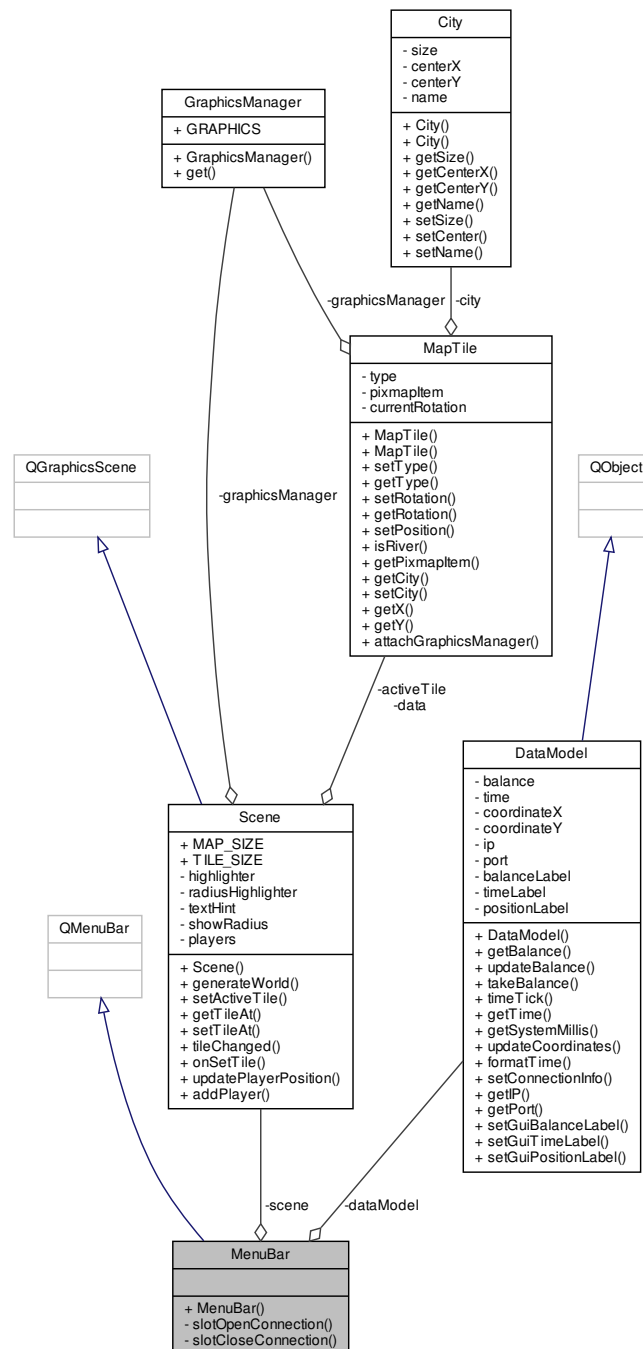
7.7 MenuBar Klassenreferenz

```
#include <menubar.h>
```

Klassendiagramm für MenuBar:



Zusammengehörigkeiten von MenuBar:



Öffentliche Methoden

- **MenuBar** (**Scene** *pScene, **DataModel** *pDataModel)

MenuBar::MenuBar Erzeugt Menüstruktur.

Private Slots

- void `slotOpenConnection` ()
MenuBar::openConnection Öffnet Input-Dialog für IP-Adresse und initiiert `Client`.
- void `slotCloseConnection` ()
MenuBar::closeConnection Schließt die aktuelle Verbindung mit dem Server.

Private Attribute

- `Scene` * `scene`
- `DataModel` * `dataModel`

7.7.1 Beschreibung der Konstruktoren und Destruktoren

7.7.1.1 MenuBar()

```
MenuBar::MenuBar (
    Scene * pScene,
    DataModel * pDataModel )
```

`MenuBar::MenuBar` Erzeugt Menüstruktur.

7.7.2 Dokumentation der Elementfunktionen

7.7.2.1 slotCloseConnection

```
void MenuBar::slotCloseConnection ( ) [private], [slot]
```

`MenuBar::closeConnection` Schließt die aktuelle Verbindung mit dem Server.

7.7.2.2 slotOpenConnection

```
void MenuBar::slotOpenConnection ( ) [private], [slot]
```

`MenuBar::openConnection` Öffnet Input-Dialog für IP-Adresse und initiiert `Client`.

7.7.3 Dokumentation der Datenelemente

7.7.3.1 dataModel

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

7.7.3.2 scene

```
Scene* MenuBar::scene [private]
```

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/menubar.h](#)
- [src/application_server/menubar.cpp](#)

7.8 Player Klassenreferenz

```
#include <player.h>
```

Zusammengehörigkeiten von Player:



Öffentliche Methoden

- [Player](#) (int pld)
- void [setPosition](#) (int pX, int pY)
[Player::setPosition](#) Updated die Position des Spielers.

Öffentliche Attribute

- QGraphicsRectItem * [highlighter](#)
- int [posX](#)
- int [posY](#)

7.8.1 Beschreibung der Konstruktoren und Destruktoren

7.8.1.1 Player()

```
Player::Player (
    int pId )
```

7.8.2 Dokumentation der Elementfunktionen

7.8.2.1 setPosition()

```
void Player::setPosition (
    int pX,
    int pY )
```

[Player::setPosition](#) Updated die Position des Spielers.

Parameter

<i>pX</i>	Der X-Index
<i>pY</i>	Der Y-Index

7.8.3 Dokumentation der Datenelemente

7.8.3.1 highlighter

```
QGraphicsRectItem* Player::highlighter
```

7.8.3.2 posX

```
int Player::posX
```

7.8.3.3 posY

```
int Player::posY
```

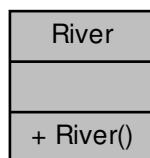
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/player.h](#)
- [src/application_server/player.cpp](#)

7.9 River Klassenreferenz

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

Zusammengehörigkeiten von River:



Öffentliche Methoden

- [River\(\)](#)

7.9.1 Beschreibung der Konstruktoren und Destruktoren

7.9.1.1 River()

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

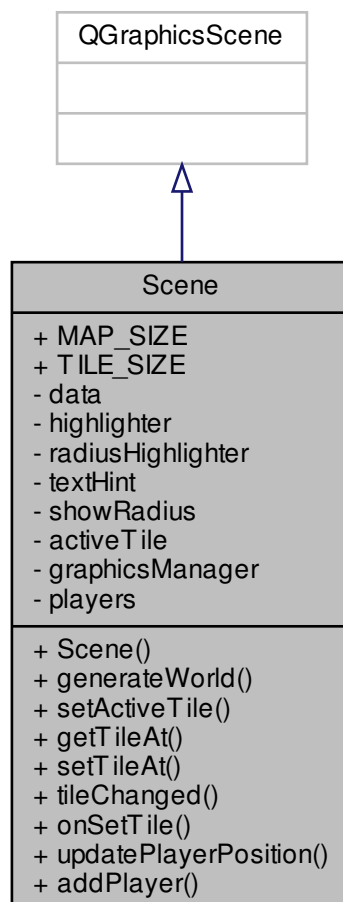
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/river.h](#)
- [src/application_server/river.cpp](#)

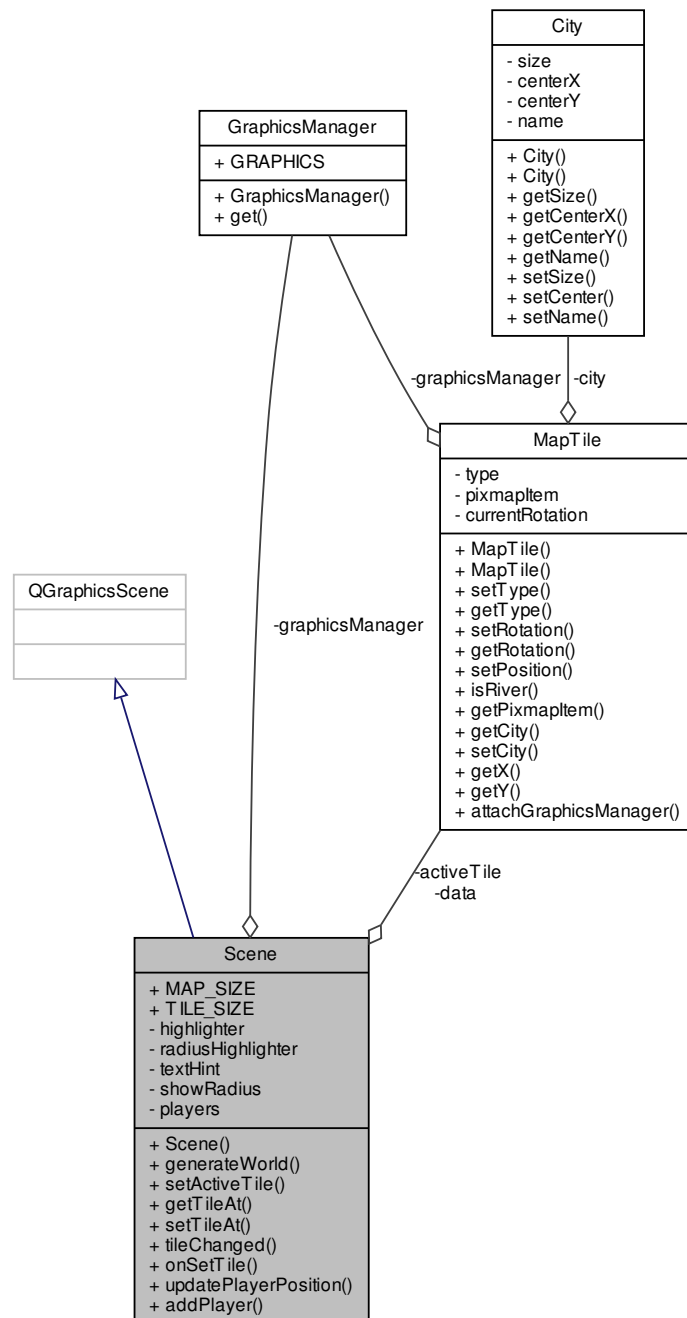
7.10 Scene Klassenreferenz

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

Klassendiagramm für Scene:



Zusammengehörigkeiten von Scene:



Öffentliche Slots

- void [onSetTile](#) (int, int, int, int)
Scene::onSetTile Ändert ein [MapTile](#) ohne ein Signal an den Server. Notwendig zum Empfangen von Änderungen.
- void [updatePlayerPosition](#) (int, int, int)
Scene::updatePlayerPosition Slot zum Updaten eines Spielers.
- void [addPlayer](#) (int)
Scene::addPlayer Fügt einen Netzwerkspieler hinzu.

Signale

- void `tileUpdate` (int, int, int, int)

Öffentliche Methoden

- `Scene` (`GraphicsManager *pGraphicsManager`)
`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=false)
`Scene::getTileAt` Liefert ein `MapTile` anhand der Pixel-Koordinaten oder der Indizes.
- void `setTileAt` (int, int, int, int)
`Scene::setTileAt` Setzt ein `MapTile` anhand der Pixel-Koordinaten oder der Indizes.
- void `tileChanged` (int, int)
`Scene::tileChanged` Meldet das sich ein `MapTile` geändert hat.

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`
- `GraphicsManager * graphicsManager`
- `std::map< int, Player * > players`

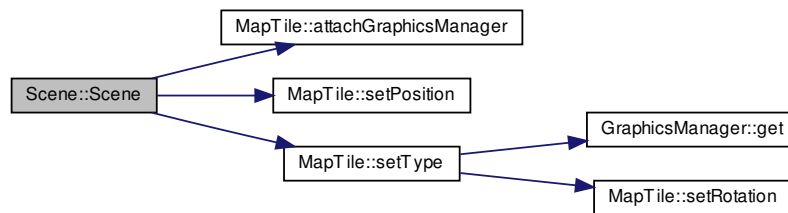
7.10.1 Beschreibung der Konstruktoren und Destruktoren

7.10.1.1 Scene()

```
Scene::Scene (
    GraphicsManager * pGraphicsManager )
```

[Scene::Scene](#) Konstruktor.

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



7.10.2 Dokumentation der Elementfunktionen

7.10.2.1 addPlayer

```
void Scene::addPlayer (
    int pId ) [slot]
```

[Scene::addPlayer](#) Fügt einen Netzwerkspieler hinzu.

Parameter

$p \leftrightarrow$ <i>Id</i>	Die ID des Spielers.
----------------------------------	----------------------

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

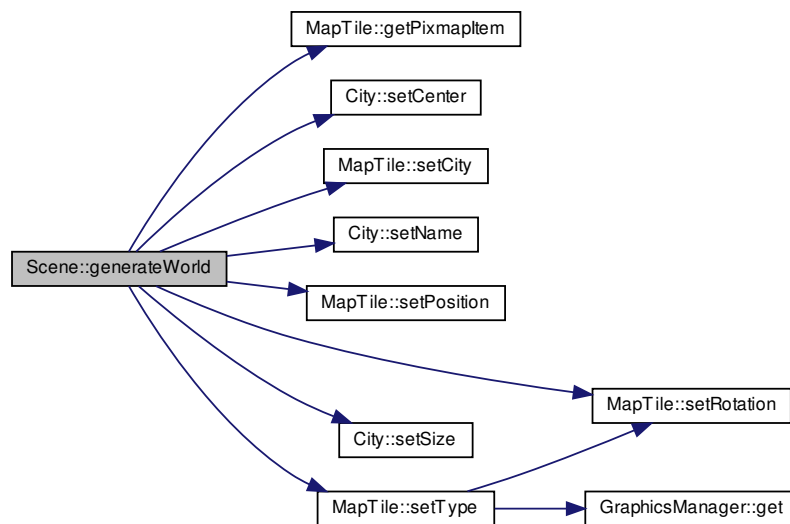


7.10.2.2 generateWorld()

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

[Scene::generateWorld](#) Diese Methode generiert eine neue Welt.

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



7.10.2.3 getTileAt()

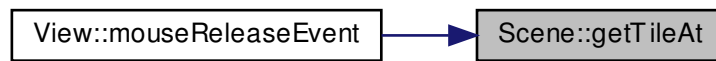
```
MapTile * Scene::getTileAt (
    int posX,
    int posY,
    bool isPixelCoordinate = false )
```

[Scene::getTileAt](#) Liefert ein [MapTile](#) anhand der Pixel-Koordinaten oder der Indizes.

Parameter

<i>posX</i>	Die X-Koordinate
<i>posY</i>	Die Y-Koordinate

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



7.10.2.4 onSetTile

```

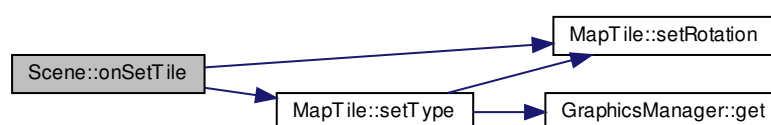
void Scene::onSetTile (
    int pX,
    int pY,
    int pType,
    int pRotation ) [slot]
  
```

[Scene::onSetTile](#) Ändert ein [MapTile](#) ohne ein Signal an den Server. Notwendig zum Empfangen von Änderungen.

Parameter

<i>pX</i>	Die X-Koordinate.
<i>pY</i>	Die Y-Koordinate.
<i>pType</i>	Der Typ.
<i>pRotation</i>	Die Rotation.

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



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



7.10.2.5 setActiveTile()

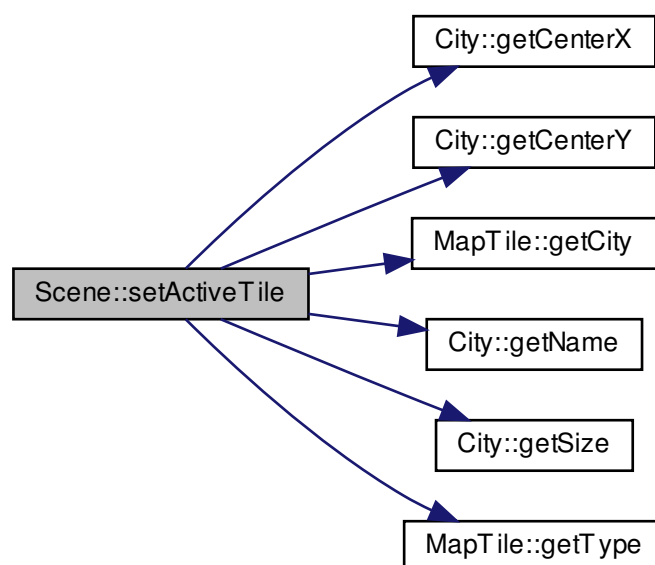
```
void Scene::setActiveTile (
    QGraphicsItem * pItem )
```

[Scene::setActiveTile](#) Setzt den [MapTile](#) über dem die Maus gerade ist. Wird von view aufgerufen.

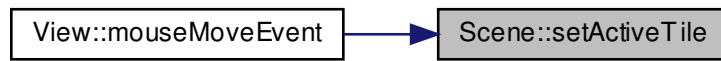
Parameter

<i>pItem</i>	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:



7.10.2.6 setTileAt()

```

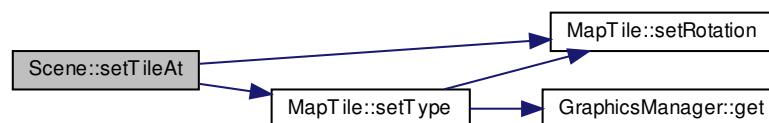
void Scene::setTileAt (
    int pX,
    int pY,
    int pType,
    int pRotation )
  
```

[Scene::setTileAt](#) Setzt ein [MapTile](#) anhand der Pixel-Koordinaten oder der Indizes.

Parameter

<i>posX</i>	
<i>posY</i>	
<i>isPixelCoordinate</i>	

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



7.10.2.7 tileChanged()

```

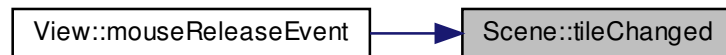
void Scene::tileChanged (
    int pX,
    int pY )
  
```

[Scene::tileChanged](#) Meldet das sich ein [MapTile](#) geändert hat.

Parameter

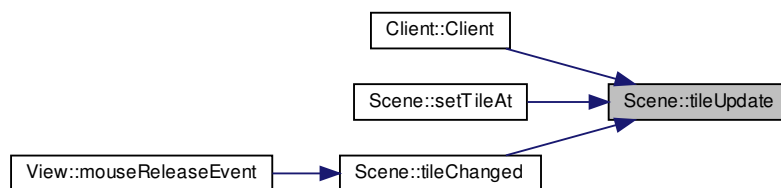
<i>pX</i>	Die X-Koordinate.
<i>pY</i>	Die Y-Koordinate.

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

**7.10.2.8 tileUpdate**

```
void Scene::tileUpdate (  
    int ,  
    int ,  
    int ,  
    int ) [signal]
```

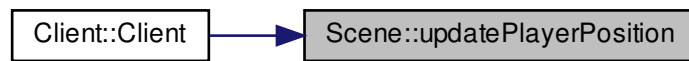
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

**7.10.2.9 updatePlayerPosition**

```
void Scene::updatePlayerPosition (  
    int pId,  
    int pX,  
    int pY ) [slot]
```

[Scene::updatePlayerPosition](#) Slot zum Updaten eines Spielers.

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



7.10.3 Dokumentation der Datenelemente

7.10.3.1 activeTile

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

7.10.3.2 data

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

7.10.3.3 graphicsManager

```
GraphicsManager* Scene::graphicsManager [private]
```

7.10.3.4 highlighter

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

7.10.3.5 MAP_SIZE

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

7.10.3.6 players

```
std::map<int, Player *> Scene::players [private]
```

7.10.3.7 radiusHighlighter

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

7.10.3.8 showRadius

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

7.10.3.9 textHint

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

7.10.3.10 TILE_SIZE

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

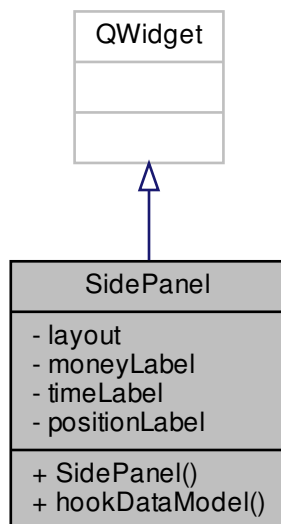
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/scene.h](#)
- [src/application_server/scene.cpp](#)

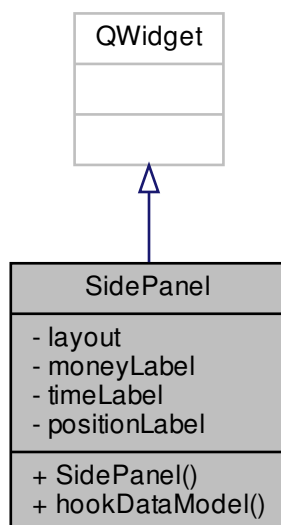
7.11 SidePanel Klassenreferenz

```
#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](#)

7.11.1 Beschreibung der Konstruktoren und Destruktoren

7.11.1.1 SidePanel()

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

[SidePanel::SidePanel](#) Erzeugt ein neues Side-Panel (Menü)

Parameter

pParent	Das Parent-Element
-------------------------	--------------------

7.11.2 Dokumentation der Elementfunktionen

7.11.2.1 hookDataModel()

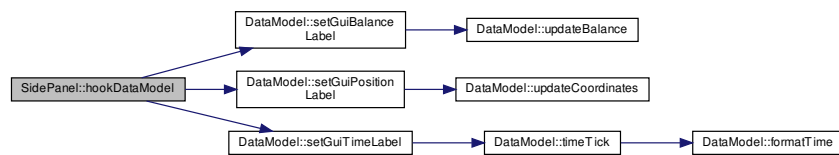
```
void SidePanel::hookDataModel (
    DataModel * pModel )
```

[SidePanel::hookDataModel](#) Verknüpft ein Datenmodell mit der Anzeige. Dadurch können dann Textfelder etc. aktualisiert werden.

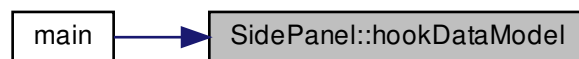
Parameter

pModel	Ein Datenmodell.
------------------------	------------------

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



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



7.11.3 Dokumentation der Datenelemente

7.11.3.1 layout

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

7.11.3.2 moneyLabel

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

7.11.3.3 positionLabel

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

7.11.3.4 timeLabel

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

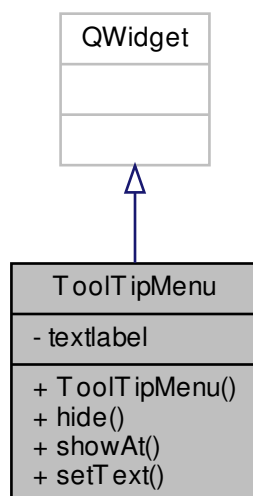
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/sidepanel.h](#)
- [src/application_server/sidepanel.cpp](#)

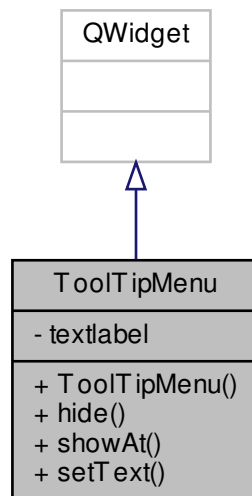
7.12 ToolTipMenu Klassenreferenz

```
#include <tooltipmenu.h>
```

Klassendiagramm für ToolTipMenu:



Zusammengehörigkeiten von ToolTipMenu:



Öffentliche Methoden

- [ToolTipMenu\(\)](#)
- void [hide\(\)](#)
- void [showAt\(\)](#) (int x, int y)
- void [setText\(\)](#) (QString pText)

Private Attribute

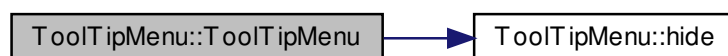
- QLabel * [textlabel](#)

7.12.1 Beschreibung der Konstruktoren und Destruktoren

7.12.1.1 ToolTipMenu()

```
ToolTipMenu::ToolTipMenu ( )
```

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

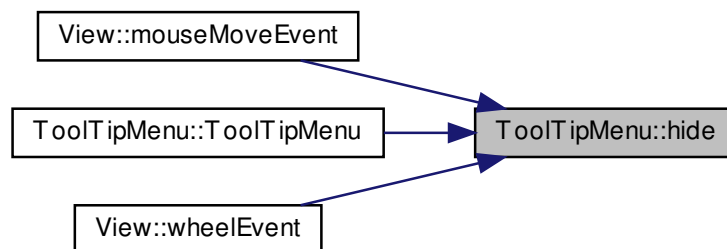


7.12.2 Dokumentation der Elementfunktionen

7.12.2.1 hide()

```
void ToolTipMenu::hide ( )
```

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



7.12.2.2 setText()

```
void ToolTipMenu::setText (
    QString pText )
```

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



7.12.2.3 showAt()

```
void ToolTipMenu::showAt (  
    int x,  
    int y )
```

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



7.12.3 Dokumentation der Datenelemente

7.12.3.1 textlabel

```
QLabel* ToolTipMenu::textlabel [private]
```

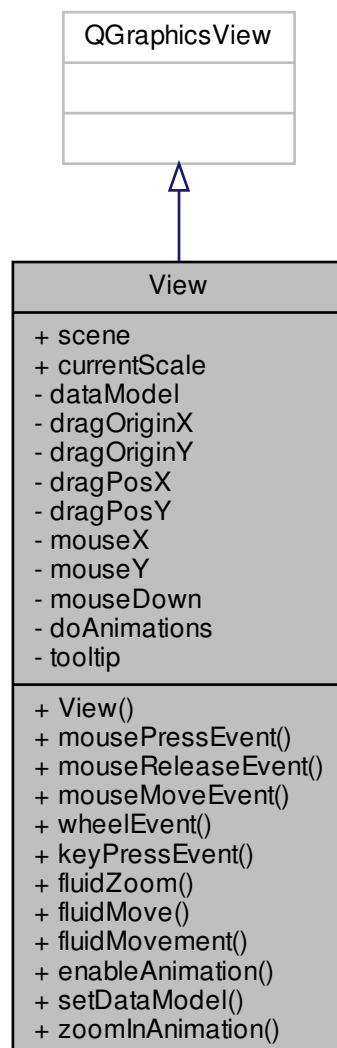
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/tooltipmenu.h](#)
- [src/application_server/tooltipmenu.cpp](#)

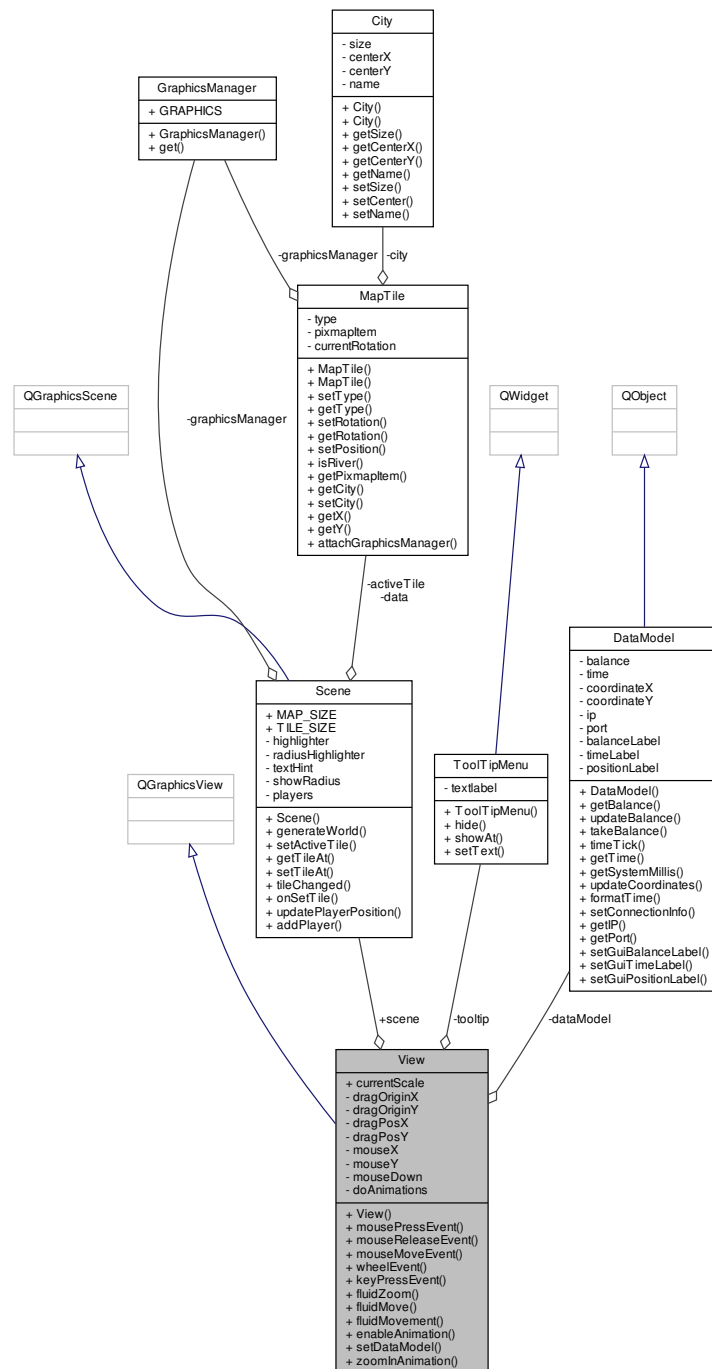
7.13 View Klassenreferenz

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

Klassendiagramm für View:



Zusammengehörigkeiten von View:



Öffentliche Slots

- void [zoomInAnimation\(\)](#)

View::zoomInAnimation Slot der nach dem Laden der Karte aufgerufen wird.

Öffentliche Methoden

- [View](#) ([Scene](#) *pScene, [ToolTipMenu](#) *pToolTip)
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 Mousrad 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](#)
- int [mouseX](#)
- int [mouseY](#)
- bool [mouseDown](#)
- bool [doAnimations](#)
- [ToolTipMenu](#) * tooltip

7.13.1 Beschreibung der Konstruktoren und Destruktoren

7.13.1.1 View()

```
View::View (
    Scene * pScene,
    ToolTipMenu * pToolTip )
```

[View::View](#) Konstruktor. Versteckt u.a. die Scrollbars und aktiviert Mousetracking.

Parameter

<code>pScene</code>	Das Zugehörige Szenenobjekt.
---------------------	------------------------------

7.13.2 Dokumentation der Elementfunktionen

7.13.2.1 enableAnimation()

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

[View::enableAnimation](#) Aktiviert animationen bis zum nächsten Event.

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



7.13.2.2 fluidMove()

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

[View::fluidMove](#) Verschiebt die Karte animiert und relativ zur aktuellen Position.

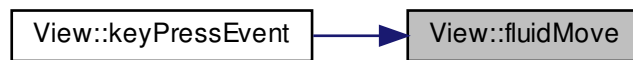
Parameter

<code>vX</code>	Verschiebung in X-Richtung.
<code>vY</code>	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:



7.13.2.3 fluidMovement()

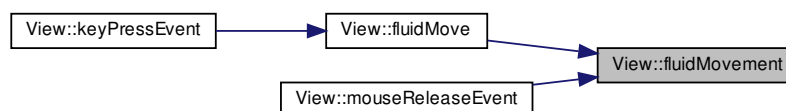
```
void View::fluidMovement (
    int pX,
    int pY )
```

[View::fluidMovement](#) Verschiebt die Karte animiert an zu einer absoluten Koordinate.

Parameter

<i>pX</i>	Die X-Koordinate.
<i>pY</i>	Die Y-Koordinate.

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



7.13.2.4 fluidZoom()

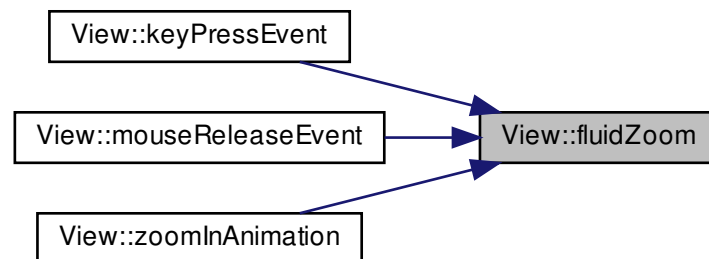
```
void View::fluidZoom (
    double target,
    bool in )
```

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

Parameter

<i>target</i>	Die angestrebte Skalierung.
<i>in</i>	Ob vergrößert oder verkleinert werden soll. (true = reinzoomen, false=rauszoomen).

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



7.13.2.5 keyPressEvent()

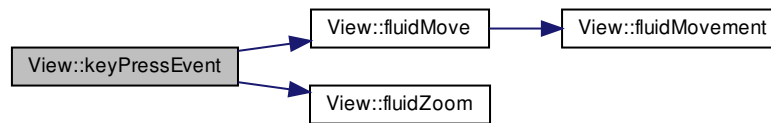
```
void View::keyPressEvent (
    QKeyEvent * event ) [override]
```

[View::keyPressEvent](#) QT Methode. Wird Aufgerufen wenn eine Taste gedrückt wird.

Parameter

<i>event</i>	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:



7.13.2.6 mouseMoveEvent()

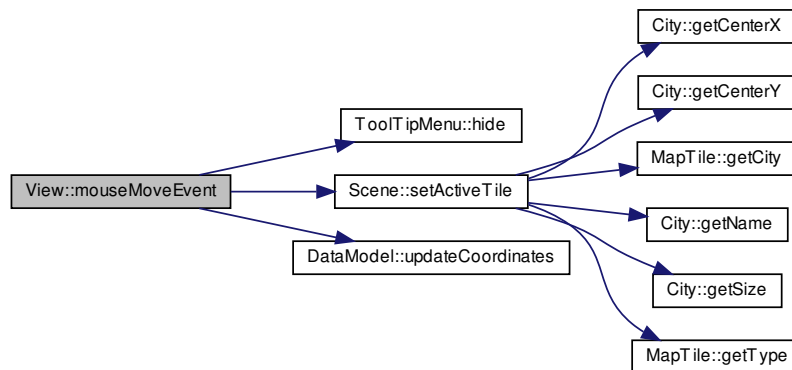
```
void View::mouseMoveEvent (
    QMouseEvent * event ) [override]
```

[View::mouseMoveEvent](#) QT Methode. Wird aufgerufen wenn die Maus bewegt wird.

Parameter

<code>event</code>	Informationen über Position der Maus
--------------------	--------------------------------------

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



7.13.2.7 mousePressEvent()

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

[View::mousePressEvent](#) QT Methode. Wird aufgerufen wenn die Maus gedrückt wurde.

Parameter

<i>event</i>	Enthält Informationen über die Taste und Position.
--------------	--

7.13.2.8 mouseReleaseEvent()

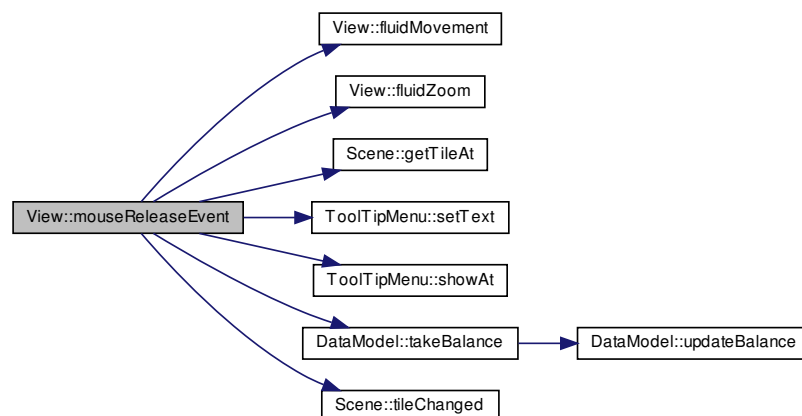
```
void View::mouseReleaseEvent (  
    QMouseEvent * event ) [override]
```

[View::mouseReleaseEvent](#) QT Methode. Wird aufgerufen wenn die Maus losgelassen wird.

Parameter

<i>event</i>	Informationen über Position und Taste
--------------	---------------------------------------

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



7.13.2.9 setDataModel()

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

[View::setDataModel](#) Setzt das Datenmodell. An dieses wird dann kontinuierlich die aktuelle Position weitergegeben.

Parameter

<i>pModel</i>	Ein Datenmodell.
---------------	------------------

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



7.13.2.10 wheelEvent()

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

[View::wheelEvent](#) QT Methode. Wird aufgerufen wenn das Mausevent gedreht wird.

Parameter

<i>event</i>	Eventobjekt mit Infos. Wichtig: <code>event->delta()</code> : Positiv oder negativ je nachdem in welche Richtung gedreht wurde.
--------------	--

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

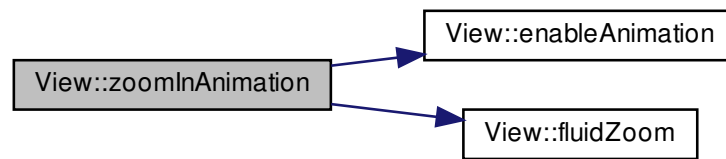


7.13.2.11 zoomInAnimation

```
void View::zoomInAnimation ( ) [slot]
```

[View::zoomInAnimation](#) Slot der nach dem Laden der Karte aufgerufen wird.

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



7.13.3 Dokumentation der Datenelemente

7.13.3.1 currentScale

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

7.13.3.2 dataModel

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

7.13.3.3 doAnimations

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

7.13.3.4 dragOriginX

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

7.13.3.5 dragOriginY

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


7.13.3.6 dragPosX

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

7.13.3.7 dragPosY

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

7.13.3.8 mouseDown

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

7.13.3.9 mouseX

```
int View::mouseX [private]
```

7.13.3.10 mouseY

```
int View::mouseY [private]
```

7.13.3.11 scene

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

7.13.3.12 tooltip

```
ToolTipMenu* View::tooltip [private]
```

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/application_server/view.h](#)
- [src/application_server/view.cpp](#)

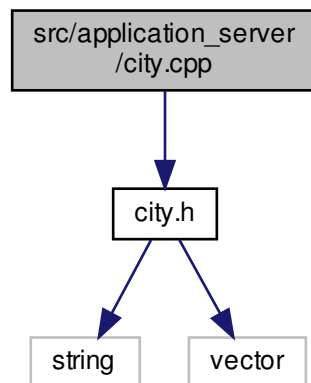
Kapitel 8

Datei-Dokumentation

8.1 src/application_server/city.cpp-Dateireferenz

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

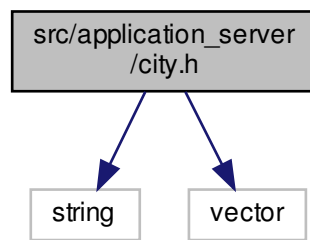
Include-Abhängigkeitsdiagramm für city.cpp:



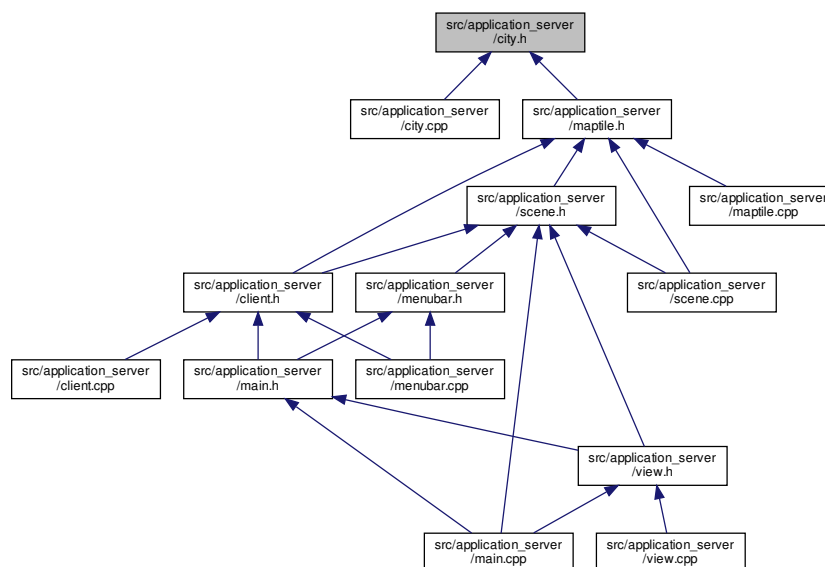
8.2 src/application_server/city.h-Dateireferenz

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

Include-Abhängigkeitsdiagramm für city.h:



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



Klassen

- class [City](#)

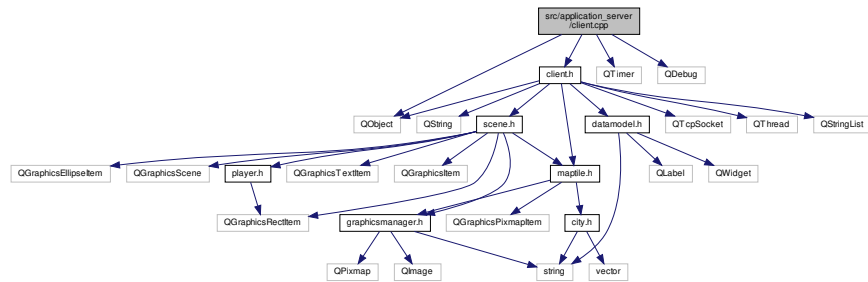
8.3 src/application_server/client.cpp-Dateireferenz

```

#include "client.h"
#include <QTimer>
#include <QDebug>
  
```

```
#include <QObject>
```

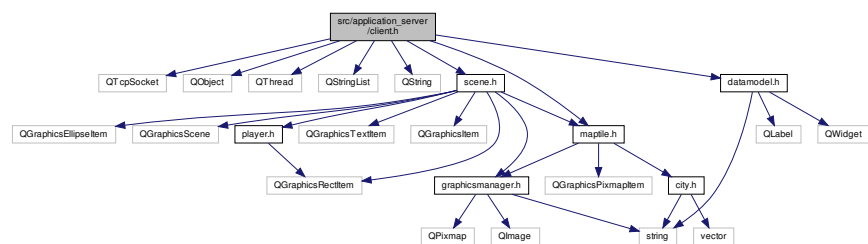
Include-Abhängigkeitsdiagramm für client.cpp:



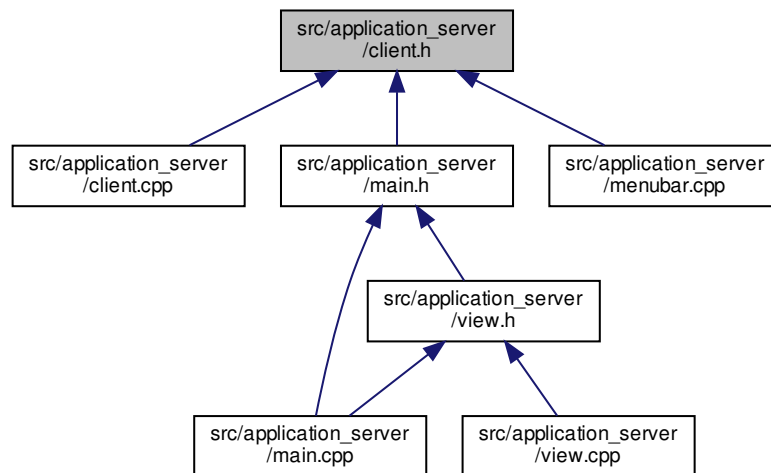
8.4 src/application_server/client.h-Dateireferenz

```
#include <QTcpSocket>
#include <QObject>
#include <QThread>
#include <QStringList>
#include <QString>
#include "scene.h"
#include "maptile.h"
#include "datamodel.h"
```

Include-Abhängigkeitsdiagramm für client.h:



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



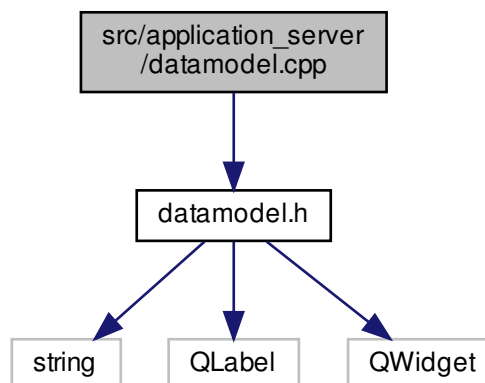
Klassen

- class [Client](#)

8.5 src/application_server/datamodel.cpp-Dateireferenz

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

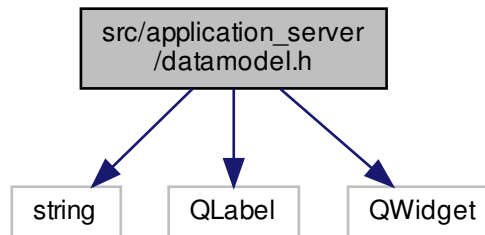
Include-Abhängigkeitsdiagramm für `datamodel.cpp`:



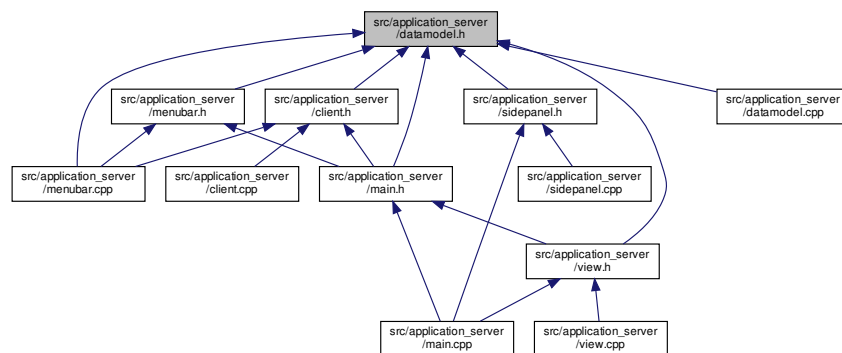
8.6 src/application_server/datamodel.h-Dateireferenz

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

Include-Abhängigkeitsdiagramm für datamodel.h:



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



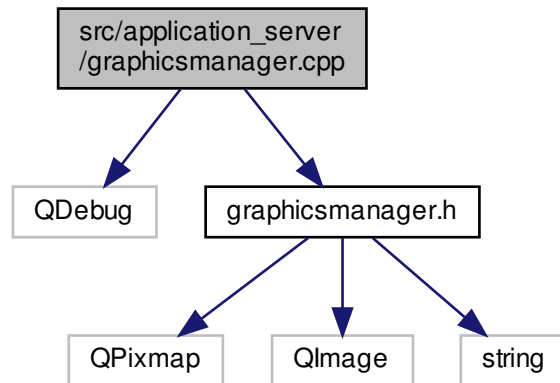
Klassen

- class [DataModel](#)

8.7 src/application_server/graphicsmanager.cpp-Dateireferenz

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

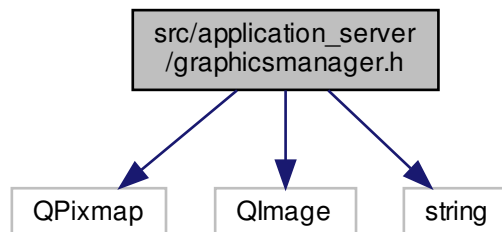
Include-Abhängigkeitsdiagramm für graphicsmanager.cpp:



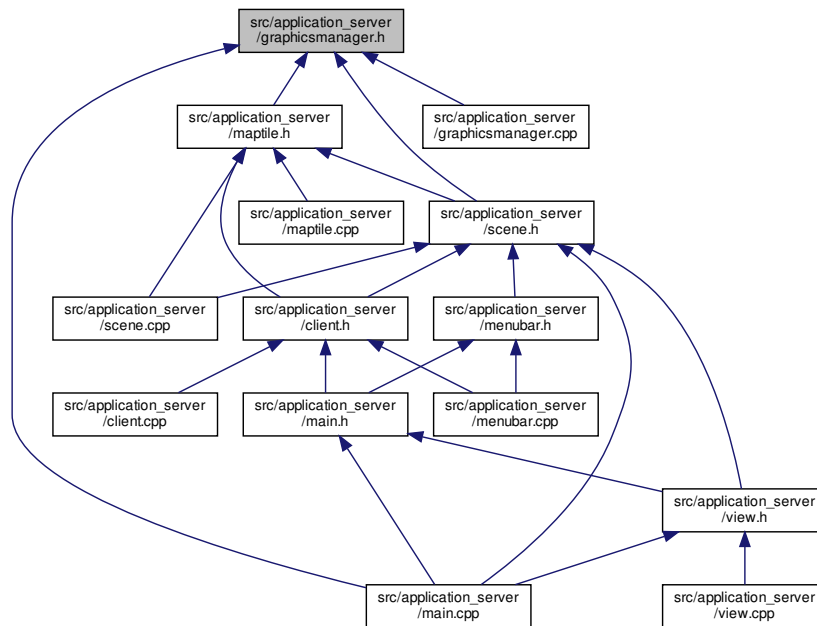
8.8 src/application_server/graphicsmanager.h-Dateireferenz

```
#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](#)

8.9 src/application_server/main.cpp-Dateireferenz

```

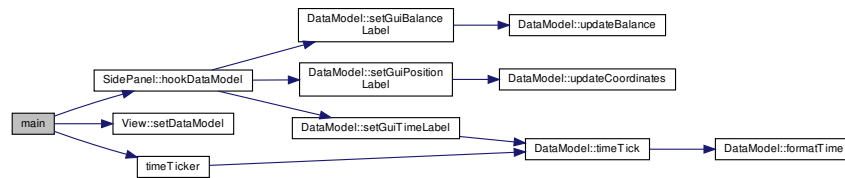
#include <QApplication>
#include <QMenuBar>
#include <QDebug>
#include <QAction>
#include <QTextItem>
#include <QTimer>
#include <QDockWidget>
#include <QWidget>
#include <QGridLayout>
#include <QInputDialog>
#include "mainwindow.h"
#include "main.h"
#include "view.h"
#include "scene.h"
#include "graphicsmanager.h"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

```


Rückgabe

Exit-Code (0=Alles gut)

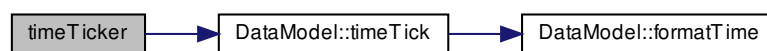
Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



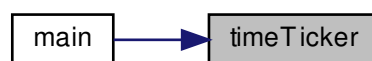
8.9.1.2 timeTicker()

```
void timeTicker ( )
```

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



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



8.9.2 Variablen-Dokumentation

8.9.2.1 client

```
Client* client
```

8.9.2.2 dataModel

```
DataModel* dataModel
```

8.9.2.3 gameRunning

```
bool gameRunning = true
```

8.9.2.4 graphics

```
GraphicsManager* graphics
```

8.9.2.5 mainWindow

```
MainWindow* mainWindow
```

8.9.2.6 scene

```
Scene* scene
```

8.9.2.7 sidePanel

```
SidePanel* sidePanel
```

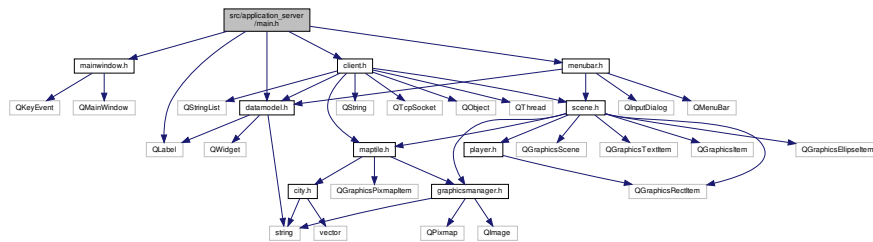
8.9.2.8 view

```
View* view
```

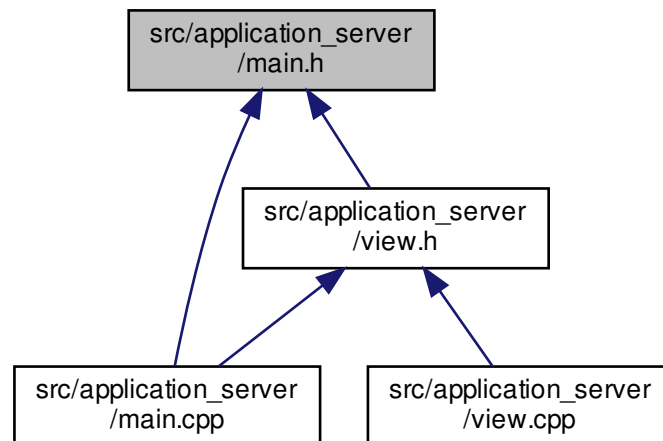
8.10 src/application_server/main.h-Dateireferenz

```
#include "mainwindow.h"
#include "datamodel.h"
#include <QLabel>
#include <client.h>
#include "menubar.h"
```

Include-Abhängigkeitsdiagramm für main.h:



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



Variablen

- `MainWindow` * `mainWindow`
- `bool gameRunning`
- `DataModel` * `dataModel`
- `Client` * `client`

8.10.1 Variablen-Dokumentation

8.10.1.1 client

```
Client* client
```

8.10.1.2 dataModel

```
DataModel* dataModel
```

8.10.1.3 gameRunning

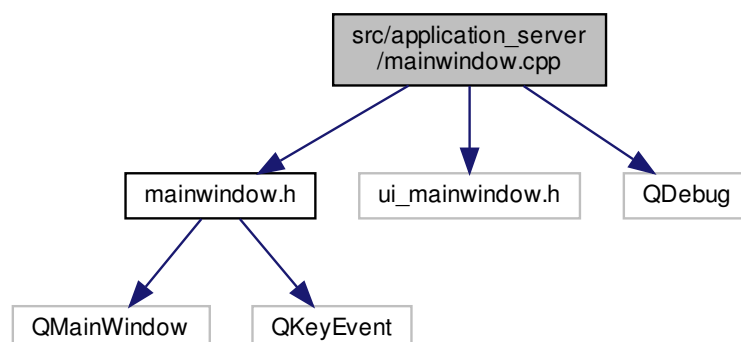
```
bool gameRunning
```

8.10.1.4 mainWindow

```
MainWindow* mainWindow
```

8.11 src/application_server/mainwindow.cpp-Dateireferenz

```
#include "mainwindow.h"  
#include "ui_mainwindow.h"  
#include <QDebug>  
Include-Abhängigkeitsdiagramm für mainwindow.cpp:
```

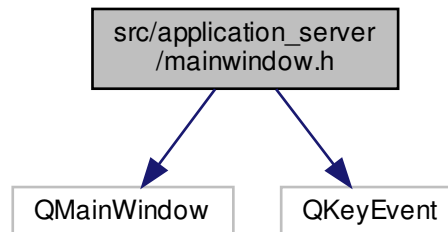


8.12 src/application_server/mainwindow.h-Dateireferenz

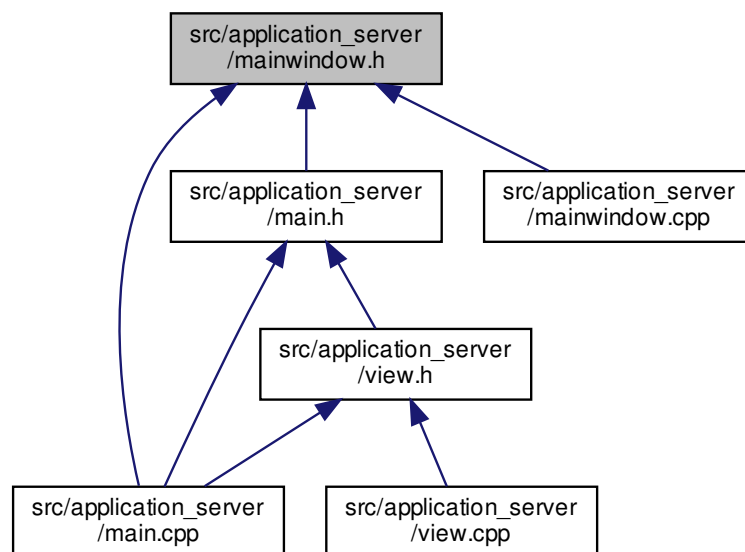
```
#include <QMainWindow>
```

```
#include <QKeyEvent>
```

Include-Abhängigkeitsdiagramm für mainwindow.h:



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



Klassen

- class [MainWindow](#)

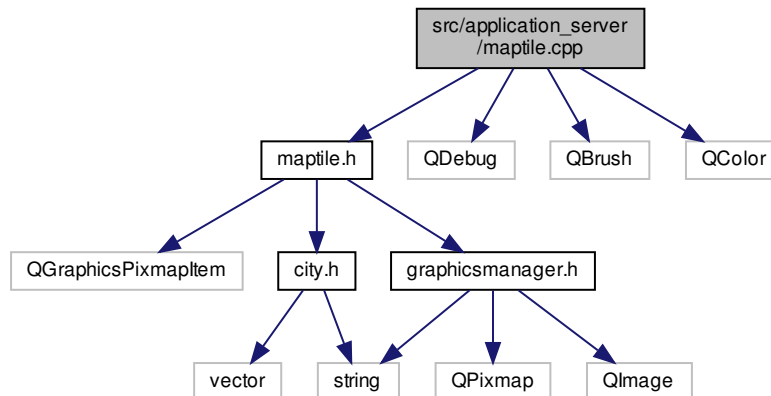
Namensbereiche

- [Ui](#)

8.13 src/application_server/maptile.cpp-Dateireferenz

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

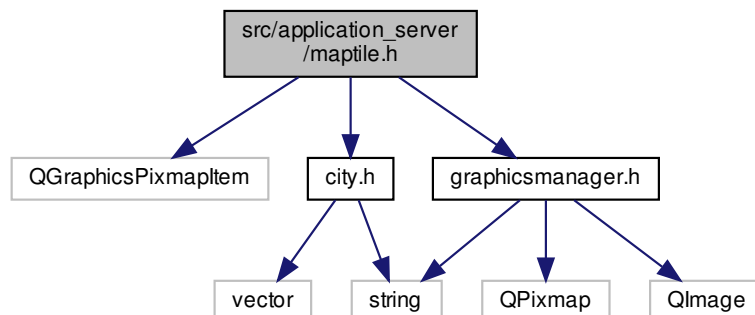
Include-Abhängigkeitsdiagramm für maptile.cpp:



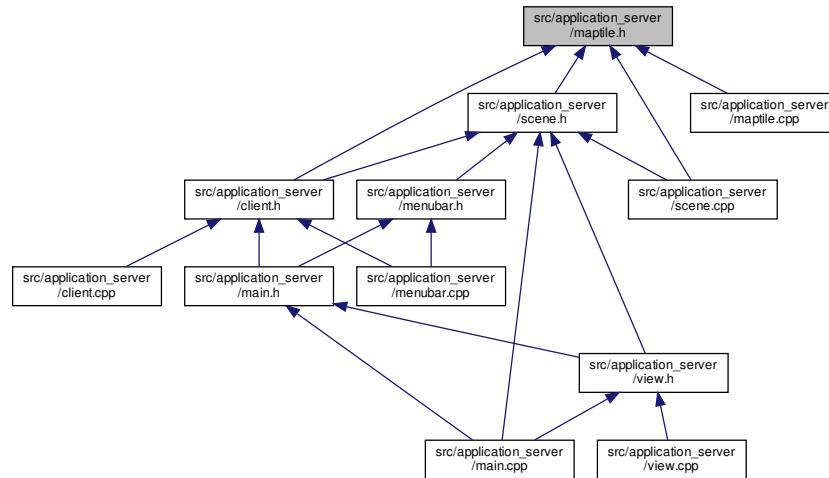
8.14 src/application_server/maptile.h-Dateireferenz

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

Include-Abhängigkeitsdiagramm für maptile.h:



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



Klassen

- class [MapTile](#)

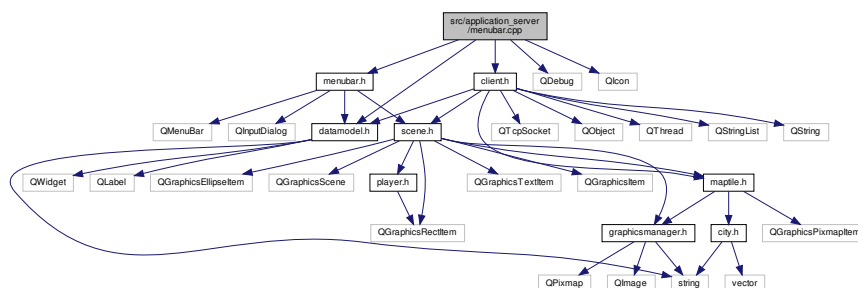
8.15 src/application_server/menubar.cpp-Dateireferenz

```

#include "menubar.h"
#include <QDebug>
#include "client.h"
#include "datamodel.h"
#include <QIcon>

```

Include-Abhängigkeitsdiagramm für menubar.cpp:



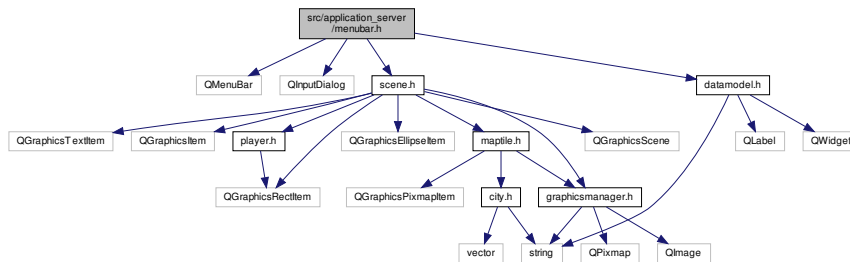
8.16 src/application_server/menubar.h-Dateireferenz

```

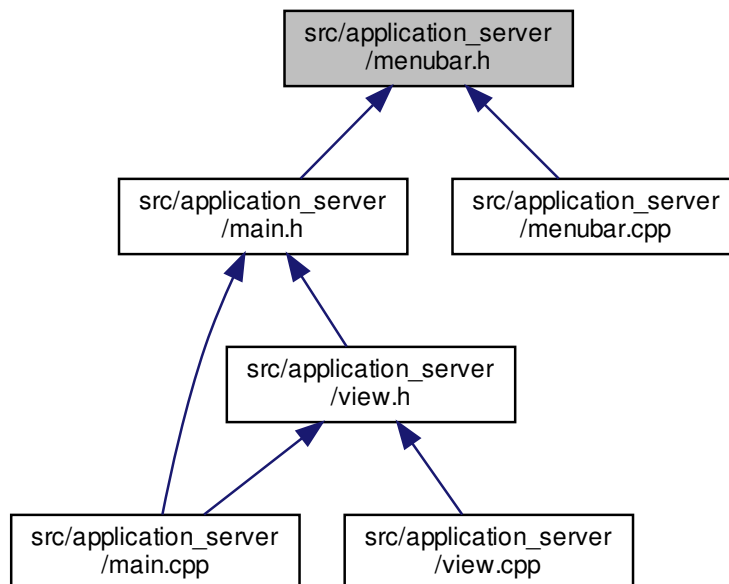
#include <QMenuBar>
#include <QInputDialog>

```

```
#include "scene.h"
#include "datamodel.h"
Include-Abhängigkeitsdiagramm für menubar.h:
```



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



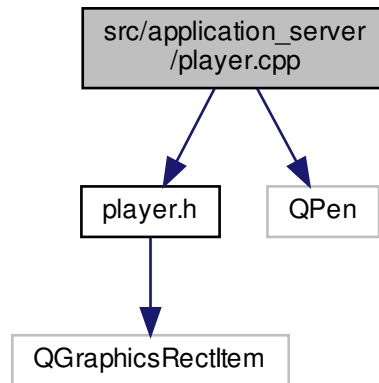
Klassen

- class [MenuBar](#)

8.17 src/application_server/player.cpp-Dateireferenz

```
#include "player.h"
#include <QPen>
```

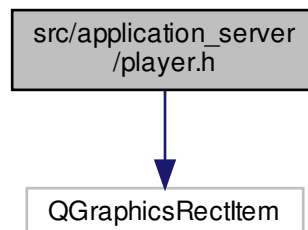
Include-Abhängigkeitsdiagramm für player.cpp:



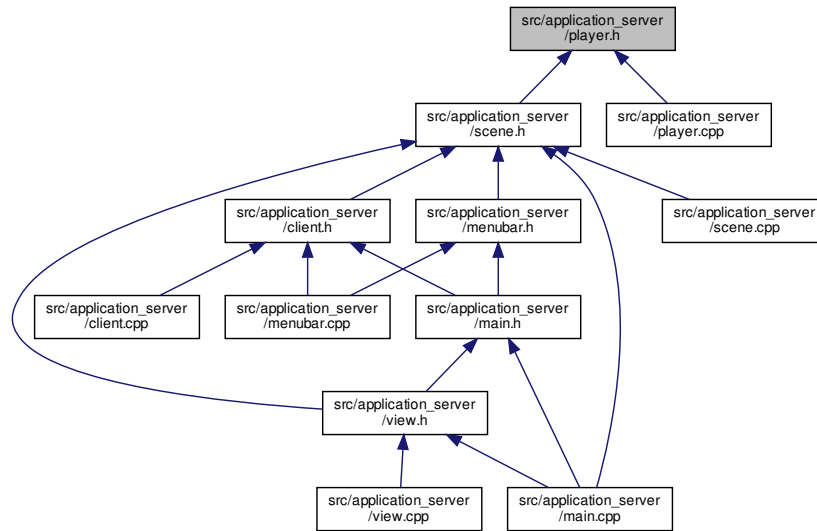
8.18 src/application_server/player.h-Dateireferenz

```
#include <QGraphicsRectItem>
```

Include-Abhängigkeitsdiagramm für player.h:



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



Klassen

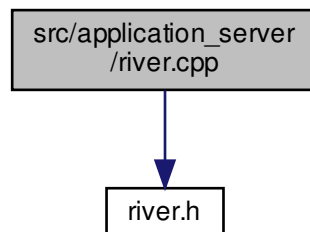
- class [Player](#)

8.19 src/application_server/README.md-Dateireferenz

8.20 src/application_server/river.cpp-Dateireferenz

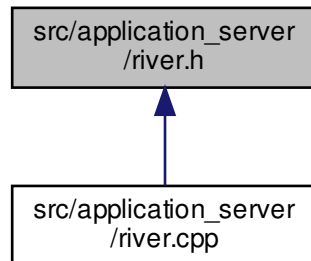
```
#include "river.h"
```

Include-Abhängigkeitsdiagramm für river.cpp:



8.21 src/application_server/river.h-Dateireferenz

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



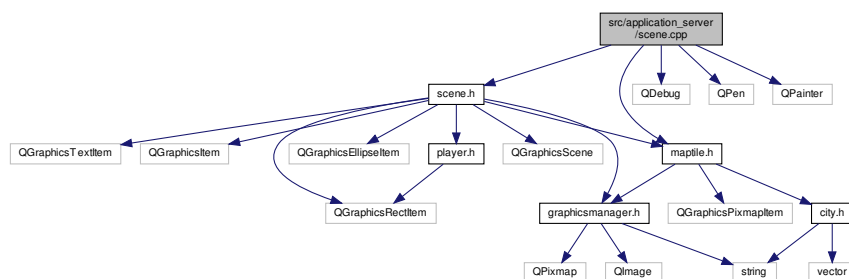
Klassen

- class [River](#)

8.22 src/application_server/scene.cpp-Dateireferenz

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

Include-Abhängigkeitsdiagramm für scene.cpp:

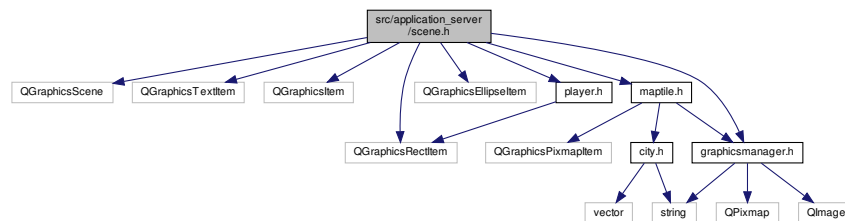


8.23 src/application_server/scene.h-Dateireferenz

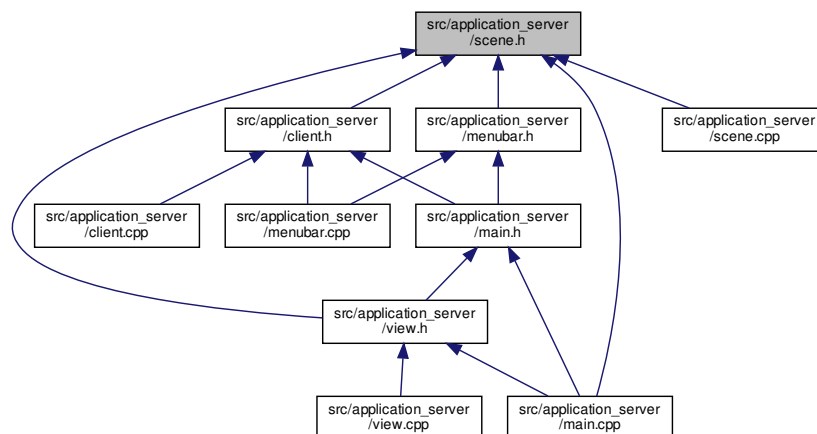
```
#include <QGraphicsScene>
#include <QGraphicsTextItem>
```

```
#include <QGraphicsItem>
#include <QGraphicsRectItem>
#include <QGraphicsEllipseItem>
#include "maptile.h"
#include "graphicsmanager.h"
#include "player.h"
```

Include-Abhängigkeitsdiagramm für scene.h:



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



Klassen

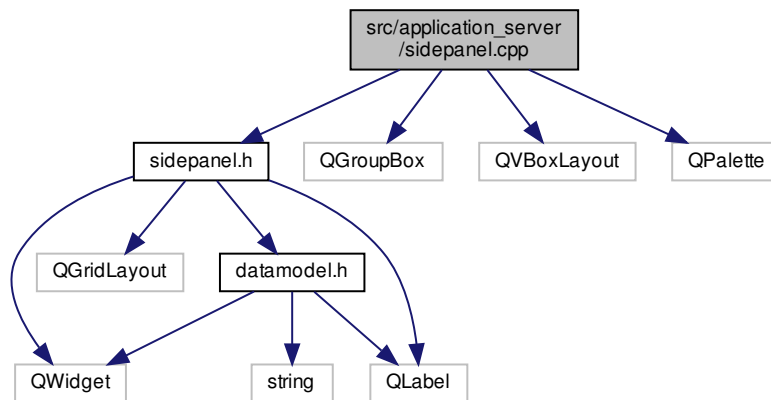
- class [Scene](#)

8.24 src/application_server/sidepanel.cpp-Dateireferenz

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

```
#include <QPalette>
```

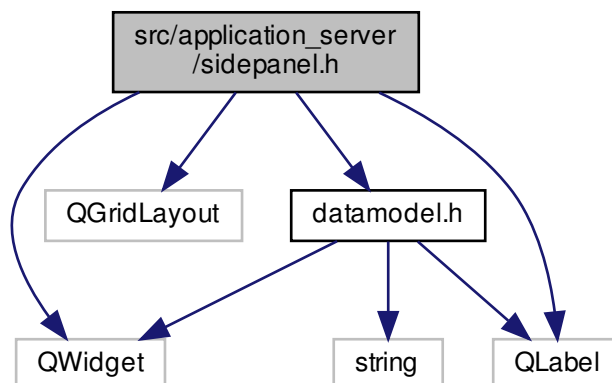
Include-Abhängigkeitsdiagramm für sidepanel.cpp:



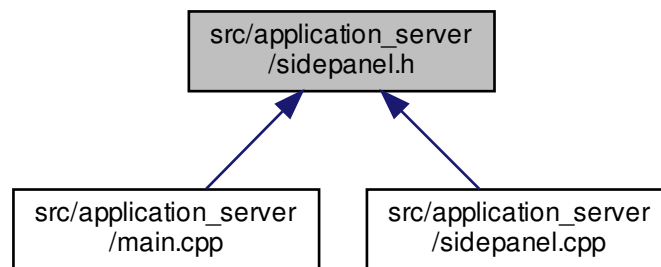
8.25 src/application_server/sidepanel.h-Dateireferenz

```
#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:



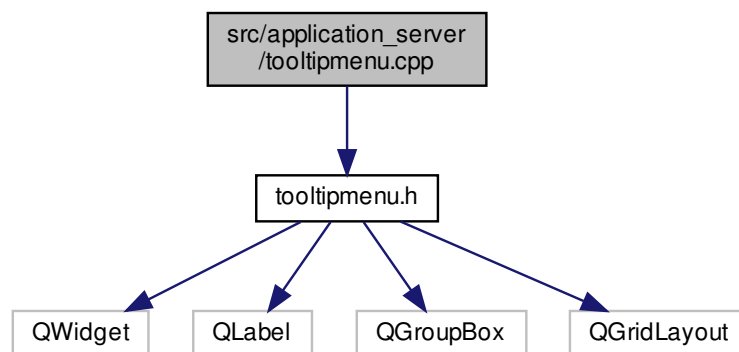
Klassen

- class [SidePanel](#)

8.26 src/application_server/tooltipmenu.cpp-Dateireferenz

```
#include "tooltipmenu.h"
```

Include-Abhängigkeitsdiagramm für tooltipmenu.cpp:



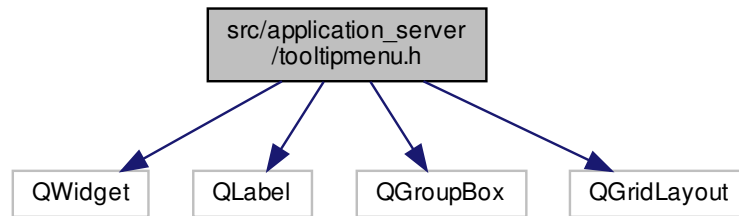
8.27 src/application_server/tooltipmenu.h-Dateireferenz

```
#include <QWidget>
#include <QLabel>
#include <QGroupBox>
```

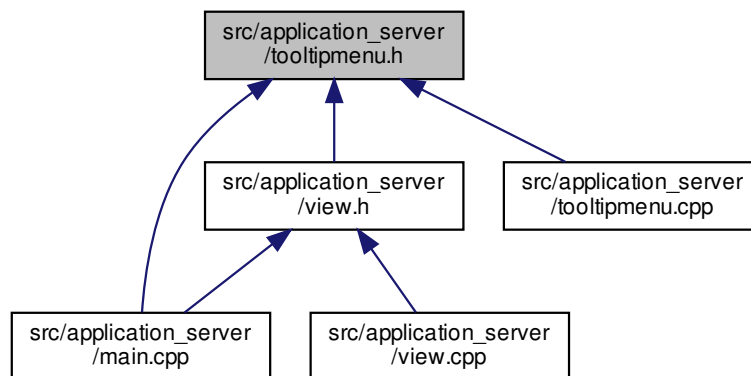


```
#include <QGridLayout>
```

Include-Abhängigkeitsdiagramm für tooltipmenu.h:



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



Klassen

- class `ToolTipMenu`

8.28 src/application_server/view.cpp-Dateireferenz

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


Klassen

- class [View](#)

