CompleteServer

0.9

Erzeugt von Doxygen 1.8.3

Mit Jan 9 2013 16:09:35

Inhaltsverzeichnis

Kapitel 1

Beispielimplementierung eines Server-Programms

1.1 Einführung

Dieses Projekt beinhaltet eine Beispielimplementierung eines Server-Programms, um zu demonstrieren, wie die `Interprozesskommunikation` zu verwenden ist. Alle, bisher im `Projekt Computerkicker (Pro-CK)` verwendete Daten, können mit diesem Server ausgetauscht werden. Wenn im Laufe der Zeit andere auszutauschende Daten im Projekt verwendet werden, sollten diese auch hier eingefügt werden.

1.2 Benutzung

Der vorliegende Quellcode kann als Basis, für ein neues Server-Programm verwendet werden.

Folgende Dateien sollten dazu in das neue Projekt kopiert werden:

- server.h
- server.cpp
- serverinterface.h
- serveruserinterface.h
- defines.h
- observer.h
- publish.h
- udpsocket.h
- udpsocket.cpp

Im neuen Projekt, muss es eine Klasse geben, die von ServerUserInterface abgeleitet ist. Diese Klasse muss alle virtuelle Funktionen von ServerUserInterface implementieren. Die Klasse Server ist, mit den entsprechenden Parametern im Konstruktor, anzulegen. Dies passiert am besten im Konstruktor, der von ServerUserInterface abgeleiteten Klasse. Danach muss die Funktion Server::start() aufgerufen werden, als Parameter muss hier ein Zeiger auf die von ServerUserInterface abgeleitete Klasse übergeben werden.

Nachdem die Klasse Server angelegt und mit Server::start() funktionsbereit gemacht wurde, können alle in ServerInterface definierten und in Server implementierten, öffentlichen Funktionen verwendet werden, um mit dem Client zu kommunizieren.

Im vorliegenden Projekt, wurde eine, von von ServerUserInterface abgeleitete Klasse (ServerUser) erzeugt. Diese Klasse verwaltet die gesamten übertragbaren Daten und stellt eine Konsolen-Benutzerschnittstelle dar, mit der die Daten manipuliert und mit dem Client kommuniziert werden kann. In einem realen Programm, sind dies die Daten der realen Umgebung.

1.3 Funktionsweise

Um die Zusammenhänge der einzelnen Klassen, deren Funktionsweise und das, auf UDP aufgesetzte Protokoll, zu erläutern, wird hier auf die entsprechenden Themen eingegangen.

1.3.1 Anlegen des Server

Wie bereits unter Benutzung beschrieben, sollte die Klasse Server, im Konstruktor der, von ServerUserInterface abgeleiteten Klasse angelegt werden.

Dies führt zu folgender Konfiguration:

Konstruktor der von ServerUserInterface abgeleiteten Klasse (ServerUser::ServerUser()) ruft den Konstruktor von Server auf (Server::Server())

Server::Server() ruft den Konstruktor von UDPsocket auf (UDPsocket::UDPsocket())

UDPsocket::UDPsocket() Startet `Winsock`

Beim Aufruf der Funktion Server::start():

Die von ServerUserInterface abgeleiteten Klasse (ServerUser) startet Server (Server::start())

Server::start() fügt sich selbst als Observer von UDPsocket hinzu (UDPsocket::attach()), und

startet UDPsocket (UDPsocket::start())

UDPsocket::start() legt einen Socket an, und

legt den Empfangsspeicher an, und

erzeugt einen neuen Thread zum Empfangen von UDP-Paketen (UDPsocket::recvData())

1.3.2 Betreiben des Server

Nachdem alle Klassen angelegt und gestartet worden sind, kann mit dem Client kommuniziert werden. Hierzu stehen aus Server-Sicht, die von ServerInterface definierten Funktionen zur Verfügung, um Datenpakete zu versenden. Dazu wird in den Funktionen, die entsprechende Struktur angelegt und mit Hilfe von UDPsocket::sendData() an den Client übertragen.

Wenn vom Client Datenpakete eintreffen, werden diese in Server::update() entgegengenommen und ausgewertet. Je nach Inhalt der Pakete, wird die entsprechende, in ServerUserInterface definierte Funktion aufgerufen.

Alive-Pakete, werden ähnlich wie beim Client behandelt, mit dem Unterschied, dass es nicht nur einen Kommunikationspartner gibt, sondern, im allgemeinen Fall, mehrere. Trifft also eine Alive-Antwort ein, so werden die Listen mit den Clients nach der entsprechenden Alive-Anfrage durchsucht, um die Toggle-Variable (clientValues::aliveToggle) auf true zu setzen.

Request-Palete, holen sich bei der von ServerUserInterface abgeleiteten Klasse die benötigten Daten ab und senden diese an den Client.

Supbscribe-Pakete, fügen den abonnierenden Client in die entsprechende Abonennten-Liste ein

1.3.3 Auf UDP aufgesetztes Protokoll

Das auf UDP aufgesetzte Protokoll ist bereits im `Wiki` grob beschrieben. Hier wird allerdings detaillierter auf das Protokoll und dessen Implementierung eingegangen.

Damit der Datenaustausch Prozess- und Plattform-übergreifend stattfinden kann, muss vereinbart werden, wie die Daten im Datenbereich eines UDP-Paketes abzulegen sind. Jedes Paket hat mindestens ein 16-Bit großes, unsigned int-Feld mit einer ID, die angibt, welche Daten folgen. Danach, folgt dann die, für die entsprechende ID festgelegt Datenstruktur. Die ID's sind in der Datei defines.h in PacketID definiert.

Die Strukturen, die als Kontainer für die Daten, inklusive der ID dienen, sind ebenfalls in defines.h definiert:

- requestPacket
- subscribePacket
- long_1
- bool_1
- bool_2
- unsignedShort_2
- float_1
- float_4
- float_2_4
- float_2_2_4
- ballPacket

Wichtig ist, darauf zu achten, dass beim Server und beim Client, genau die gleichen Strukturen verwendet werden.

Noch zu erledigen Namen für das auf UDP aufgesetzte Protokoll suchen.

Kapitel 2

Ausstehende Aufgaben

page Beispielimplementierung eines Server-Programms

Namen für das auf UDP aufgesetzte Protokoll suchen.

Element ServerUser::run ()

Konsolenausgabe in Funktionen auslagern.

Element subscribePacket::cycle

Überlegen ob dies umgesetzt werden soll und wenn ja, dann umsetzen.

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

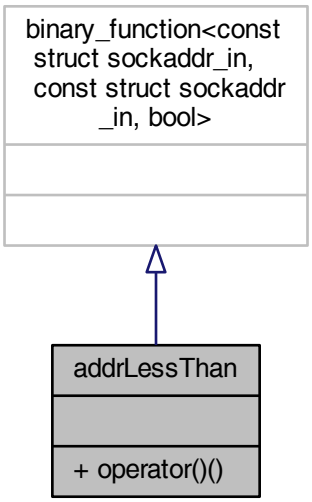
Klassen-Dokumentation

6.1 addrLessThan Strukturreferenz

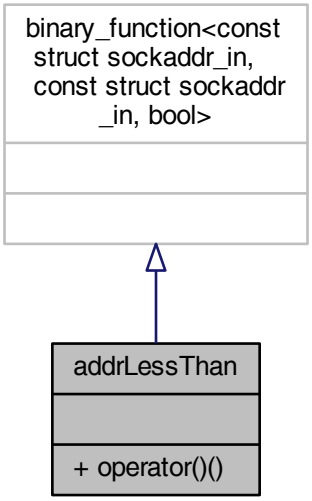
Zum vergleich zweier sockaddr_in Strukturen.

`#include <server.h>`

Klassendiagramm für addrLessThan:



Zusammengehörigkeiten von addrLessThan:



Öffentliche Methoden

- bool operator() (const struct sockaddr_in addr1, const struct sockaddr_in addr2) const

6.1.1 Ausführliche Beschreibung

Zum vergleich zweier sockaddr_in Strukturen.

Wird von dem map's für die Client-Adressen verwendet.

Definiert in Zeile 47 der Datei server.h.

6.1.2 Dokumentation der Elementfunktionen

6.1.2.1 bool addrLessThan::operator() (const struct sockaddr_in *addr1*, const struct sockaddr_in *addr2*) const [inline]

Definiert in Zeile 49 der Datei server.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

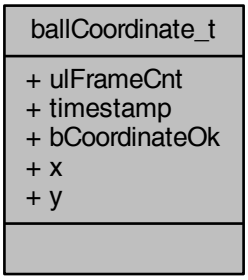
- server.h

6.2 ballCoordinate_t Strukturreferenz

Struktur der Ballposition.

`#include <defines.h>`

Zusammengehörigkeiten von ballCoordinate_t:



Öffentliche Attribute

- unsigned long ulFrameCnt
- double timestamp
- bool bCoordinateOk
- float x
- float y

6.2.1 Ausführliche Beschreibung

Struktur der Ballposition.

Definiert in Zeile 337 der Datei defines.h.

6.2.2 Dokumentation der Datenelemente

6.2.2.1 bool ballCoordinate_t::bCoordinateOk

True wenn Ballposition erfolgreich bestimmt weden konnte

Definiert in Zeile 344 der Datei defines.h.

Wird benutzt von ServerUser::run().

6.2.2.2 double ballCoordinate_t::timestamp

Zeitstempel der linken Kamera

Definiert in Zeile 342 der Datei defines.h.

Wird benutzt von ServerUser::run().

6.2.2.3 unsigned long ballCoordinate_t::ulFrameCnt

Bildpaare seit Start

Definiert in Zeile 340 der Datei defines.h.

Wird benutzt von ServerUser::run().

6.2.2.4 float ballCoordinate_t::x

Ballposition in cm (0,0 - 120,7)

Definiert in Zeile 346 der Datei defines.h.

Wird benutzt von ServerUser::run().

6.2.2.5 float ballCoordinate_t::y

Ballposition in cm (0,0 - 68,5)

Definiert in Zeile 348 der Datei defines.h.

Wird benutzt von ServerUser::run().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

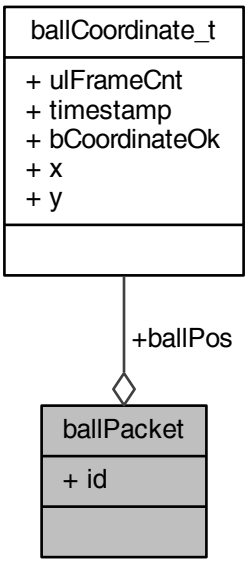
- defines.h

6.3 ballPacket Strukturreferenz

Paket mit der Ballposition.

`#include <defines.h>`

Zusammengehörigkeiten von ballPacket:



Öffentliche Attribute

- unsigned int id: 16
- ballCoordinate_t ballPos

6.3.1 Ausführliche Beschreibung

Paket mit der Ballposition.

Beinhaltet die Paket-ID und eine Struktur mit der Ballposition

Definiert in Zeile 356 der Datei defines.h.

6.3.2 Dokumentation der Datenelemente

6.3.2.1 ballCoordinate_t ballPacket::ballPos

Definiert in Zeile 359 der Datei defines.h.

Wird benutzt von Server::sendBallPos() und Server::update().

6.3.2.2 unsigned int ballPacket::id

Definiert in Zeile 358 der Datei defines.h.

Wird benutzt von Server::sendBallPos() und Server::update().

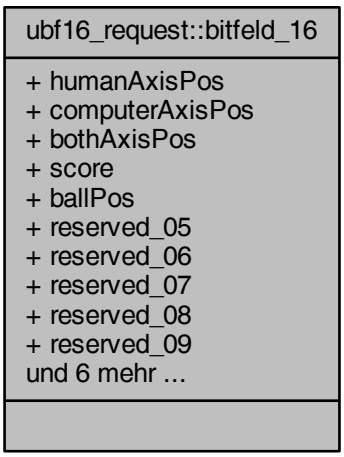
Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- defines.h

6.4 ubf16_request::bitfeld_16 Strukturreferenz

```
#include <defines.h>
```

Zusammengehörigkeiten von ubf16_request::bitfeld_16:



Öffentliche Attribute

- unsigned int humanAxisPos: 1

- unsigned int computerAxisPos: 1
- unsigned int bothAxisPos: 1
- unsigned int score: 1
- unsigned int ballPos: 1
- unsigned int reserved_05: 1
- unsigned int reserved_06: 1
- unsigned int reserved_07: 1
- unsigned int reserved_08: 1
- unsigned int reserved_09: 1
- unsigned int reserved_10: 1
- unsigned int reserved_11: 1
- unsigned int reserved_12: 1
- unsigned int reserved_13: 1
- unsigned int reserved_14: 1
- unsigned int reserved_15: 1

6.4.1 Ausführliche Beschreibung

Definiert in Zeile 191 der Datei defines.h.

6.4.2 Dokumentation der Datenelemente

6.4.2.1 unsigned int ubf16_request::bitfeld_16::ballPos

Definiert in Zeile 197 der Datei defines.h.

Wird benutzt von Server::update().

6.4.2.2 unsigned int ubf16_request::bitfeld_16::bothAxisPos

Definiert in Zeile 195 der Datei defines.h.

Wird benutzt von Server::update().

6.4.2.3 unsigned int ubf16_request::bitfeld_16::computerAxisPos

Definiert in Zeile 194 der Datei defines.h.

Wird benutzt von Server::update().

6.4.2.4 unsigned int ubf16_request::bitfeld_16::humanAxisPos

Definiert in Zeile 193 der Datei defines.h.

Wird benutzt von Server::update().

6.4.2.5 unsigned int ubf16_request::bitfeld_16::reserved_05

Definiert in Zeile 198 der Datei defines.h.

6.4.2.6 unsigned int ubf16_request::bitfeld_16::reserved_06

Definiert in Zeile 199 der Datei defines.h.

6.4.2.7 unsigned int ubf16_request::bitfeld_16::reserved_07

Definiert in Zeile 200 der Datei defines.h.

6.4.2.8 unsigned int ubf16_request::bitfeld_16::reserved_08

Definiert in Zeile 201 der Datei defines.h.

6.4.2.9 unsigned int ubf16_request::bitfeld_16::reserved_09

Definiert in Zeile 202 der Datei defines.h.

6.4.2.10 unsigned int ubf16_request::bitfeld_16::reserved_10

Definiert in Zeile 203 der Datei defines.h.

6.4.2.11 unsigned int ubf16_request::bitfeld_16::reserved_11

Definiert in Zeile 204 der Datei defines.h.

6.4.2.12 unsigned int ubf16_request::bitfeld_16::reserved_12

Definiert in Zeile 205 der Datei defines.h.

6.4.2.13 unsigned int ubf16_request::bitfeld_16::reserved_13

Definiert in Zeile 206 der Datei defines.h.

6.4.2.14 unsigned int ubf16_request::bitfeld_16::reserved_14

Definiert in Zeile 207 der Datei defines.h.

6.4.2.15 unsigned int ubf16_request::bitfeld_16::reserved_15

Definiert in Zeile 208 der Datei defines.h.

6.4.2.16 unsigned int ubf16_request::bitfeld_16::score

Definiert in Zeile 196 der Datei defines.h.

Wird benutzt von Server::update().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- defines.h

6.5 bool_1 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von bool_1:

Öffentliche Attribute

- unsigned int id: 16
- bool bl

6.5.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein bool-Wert.

Definiert in Zeile 262 der Datei defines.h.

6.5.2 Dokumentation der Datenelemente

6.5.2.1 bool bool_1::bl

Definiert in Zeile 265 der Datei defines.h.

6.5.2.2 unsigned int bool_1::id

Definiert in Zeile 264 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

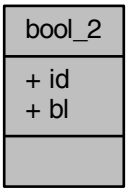
- defines.h

6.6 bool_2 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

```
#include <defines.h>
```

Zusammengehörigkeiten von bool_2:



Öffentliche Attribute

- unsigned int id: 16
- bool bl [2]

6.6.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und zwei bool-Werte.

Definiert in Zeile 273 der Datei defines.h.

6.6.2 Dokumentation der Datenelemente

6.6.2.1 bool bool_2::bl[2]

Definiert in Zeile 276 der Datei defines.h.

6.6.2.2 unsigned int bool_2::id

Definiert in Zeile 275 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

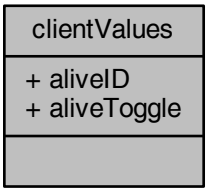
- defines.h

6.7 clientValues Strukturreferenz

Struktur für die Daten, die mit den Clientadressen in die map eingefügt werden.

`#include <server.h>`

Zusammengehörigkeiten von clientValues:



Öffentliche Attribute

- long aliveID

  ID des aktuellen Alive-Pakets.
- bool aliveToggle

  Gibt an ob der Server auf Alive-Anfragen antwortet.

6.7.1 Ausführliche Beschreibung

Struktur für die Daten, die mit den Clientadressen in die map eingefügt werden.

Struktur für die Daten, die mit einer Clientadresse als mapped value in die map eingefügt werden, welche alle Client-Adressen enthält, die Daten abonniert haben.

Definiert in Zeile 33 der Datei server.h.

6.7.2 Dokumentation der Datenelemente

6.7.2.1 long clientValues::aliveID

ID des aktuellen Alive-Pakets.

Definiert in Zeile 36 der Datei server.h.

6.7.2.2 bool clientValues::aliveToggle

Gibt an ob der Server auf Alive-Anfragen antwortet.

Definiert in Zeile 39 der Datei server.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- server.h

6.8 float_1 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von float_1:

Öffentliche Attribute

- unsigned int id: 16
- float flt

6.8.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein float-Wert.

Definiert in Zeile 295 der Datei defines.h.

6.8.2 Dokumentation der Datenelemente

6.8.2.1 float float_1::flt

Definiert in Zeile 298 der Datei defines.h.

6.8.2.2 unsigned int float_1::id

Definiert in Zeile 297 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

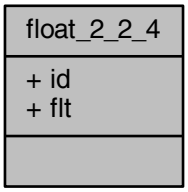
- defines.h

6.9 float_2_2_4 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

```
#include <defines.h>
```

Zusammengehörigkeiten von float_2_2_4:



Öffentliche Attribute

- unsigned int id: 16
- float flt [2][2][4]

6.9.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein dreidimensionales Array aus zwei mal zwei mal vier float-Werten.

Definiert in Zeile 328 der Datei defines.h.

6.9.2 Dokumentation der Datenelemente

6.9.2.1 float float_2_2_4::flt[2][2][4]

Definiert in Zeile 331 der Datei defines.h.

Wird benutzt von Server::sendAxisPos() und Server::update().

6.9.2.2 unsigned int float_2_2_4::id

Definiert in Zeile 330 der Datei defines.h.

Wird benutzt von Server::sendAxisPos() und Server::update().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

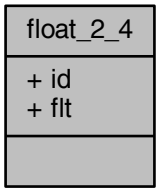
- defines.h

6.10 float_2_4 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

```
#include <defines.h>
```

Zusammengehörigkeiten von float_2_4:



Öffentliche Attribute

- unsigned int id: 16
- float flt [2][4]

6.10.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein zweidimensionales Array aus zwei mal vier float-Werten.

Definiert in Zeile 317 der Datei defines.h.

6.10.2 Dokumentation der Datenelemente

6.10.2.1 float float_2_4::flt[2][4]

Definiert in Zeile 320 der Datei defines.h.

Wird benutzt von Server::sendAxisPos() und Server::update().

6.10.2.2 unsigned int float_2_4::id

Definiert in Zeile 319 der Datei defines.h.

Wird benutzt von Server::sendAxisPos() und Server::update().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- defines.h

6.11 float_4 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von float_4:

Öffentliche Attribute

- unsigned int id: 16
- float flt [4]

6.11.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein Array aus vier float-Werten.

Definiert in Zeile 306 der Datei defines.h.

6.11.2 Dokumentation der Datenelemente

6.11.2.1 float float_4::flt[4]

Definiert in Zeile 309 der Datei defines.h.

6.11.2.2 unsigned int float_4::id

Definiert in Zeile 308 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

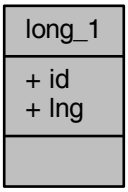
- defines.h

6.12 long_1 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von long_1:



Öffentliche Attribute

- unsigned int id: 16
- long lng

6.12.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein long-Wert.

Definiert in Zeile 251 der Datei defines.h.

6.12.2 Dokumentation der Datenelemente

6.12.2.1 unsigned int long_1::id

Definiert in Zeile 253 der Datei defines.h.

Wird benutzt von Server::answerAlive() und Server::requestAlive().

6.12.2.2 long long_1::lng

Definiert in Zeile 254 der Datei defines.h.

Wird benutzt von Server::answerAlive(), Server::requestAlive() und Server::update().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

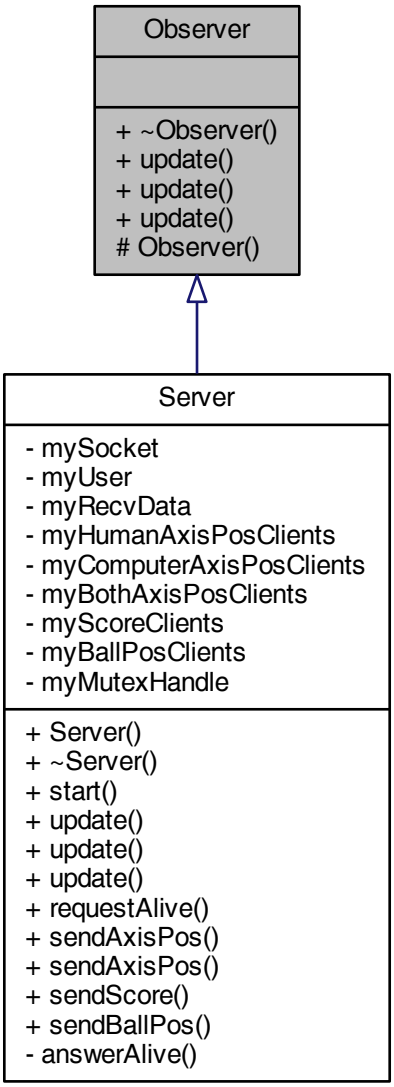
- defines.h

6.13 Observer Klassenreferenz

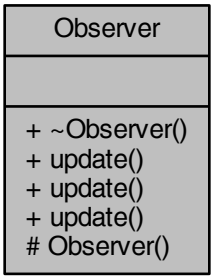
Observer Klasse für das Observer Entwurfsmodell.

`#include <observer.h>`

Klassendiagramm für Observer:



Zusammengehörigkeiten von Observer:



Öffentliche Methoden

- virtual ~Observer ()

  Destruktor.
- virtual void update (Publish *const pub)=0

  Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.
- virtual void update (Publish *const pub, const int *const iValues, const unsigned int iValuesSize)=0

  Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.
- virtual void update (Publish *const pub, const float *const fValues, const unsigned int fValuesSize, const int *const iValues=NULL, const unsigned int iValuesSize=0)=0

  Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.

Geschützte Methoden

- Observer ()

  Konstruktor.

6.13.1 Ausführliche Beschreibung

Observer Klasse für das Observer Entwurfsmodell.

Siehe auch

Publish

Definiert in Zeile 20 der Datei observer.h.

6.13.2 Beschreibung der Konstruktoren und Destruktoren

6.13.2.1 virtual Observer::~Observer ( ) [inline], [virtual]

Destruktor.

Definiert in Zeile 26 der Datei observer.h.

6.13.2.2 Observer::Observer ( ) [inline], [protected]

Konstruktor.

Definiert in Zeile 56 der Datei observer.h.

6.13.3 Dokumentation der Elementfunktionen

6.13.3.1 virtual void Observer::update (Publish *const *pub) [pure virtual]

Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.

Parameter

in	*pub*	Publisher Objekt, welches über ein Ereignis benachichtigt

Implementiert in Server.

Wird benutzt von Publish::notify().

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

6.13.3.2 virtual void Observer::update (Publish *const *pub*, const int *const *iValues*, const unsigned int *iValuesSize*) [pure virtual]

Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.

Parameter

in	*pub*	Publisher Objekt, welches über ein Ereignis benachichtigt
in	*iValues*	int-Array zum übergeben von Variabeln
in	*iValuesSize*	Größe des Array values

Implementiert in Server.

6.13.3.3 virtual void Observer::update (Publish *const *pub*, const float *const *fValues*, const unsigned int *fValuesSize*, const int *const *iValues* = NULL, const unsigned int *iValuesSize* = 0) [pure virtual]

Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.

Parameter

in	*pub*	Publisher Objekt, welches über ein Ereignis benachichtigt
in	*fValues*	float Array zum übergeben von Variabeln
in	*fValuesSize*	Größe des Array values
in	*iValues*	int-Array zum übergeben von Variabeln
in	*iValuesSize*	Größe des Array values

Implementiert in Server.

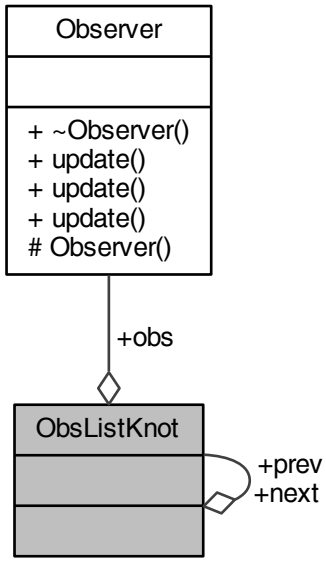
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- observer.h

6.14 ObsListKnot Strukturreferenz

`#include <publish.h>`

Zusammengehörigkeiten von ObsListKnot:



Öffentliche Attribute

- Observer * obs

  Pointer auf das Observer Objekt.
- ObsListKnot * next

  Pointer auf das nächste Listenelement.
- ObsListKnot * prev

  Pointer auf das vorherige Listenelement.

6.14.1 Ausführliche Beschreibung

Definiert in Zeile 24 der Datei publish.h.

6.14.2 Dokumentation der Datenelemente

6.14.2.1 ObsListKnot* ObsListKnot::next

Pointer auf das nächste Listenelement.

Definiert in Zeile 29 der Datei publish.h.

Wird benutzt von Publish::attach(), Publish::detach() und Publish::notify().

6.14.2.2 Observer* ObsListKnot::obs

Pointer auf das Observer Objekt.

Definiert in Zeile 27 der Datei publish.h.

Wird benutzt von Publish::attach(), Publish::detach() und Publish::notify().

6.14.2.3 ObsListKnot* ObsListKnot::prev

Pointer auf das vorherige Listenelement.

Definiert in Zeile 31 der Datei publish.h.

Wird benutzt von Publish::attach() und Publish::detach().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

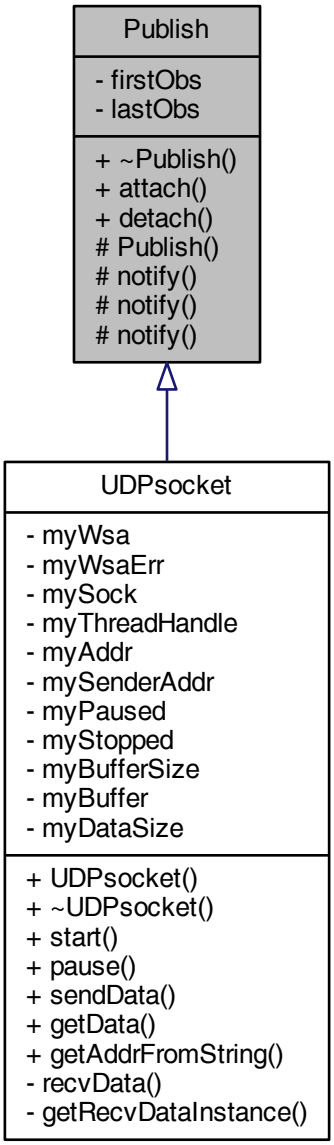
- publish.h

6.15 Publish Klassenreferenz

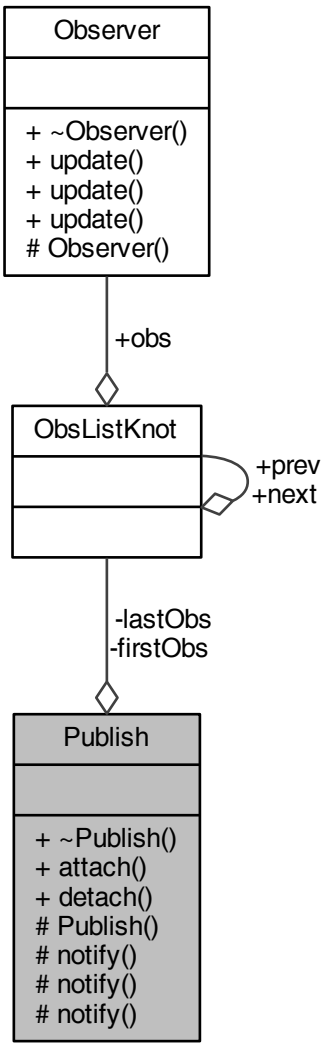
Publisher Klasse für das Observer Entwurfsmodell.

`#include <publish.h>`

Klassendiagramm für Publish:



Zusammengehörigkeiten von Publish:



Öffentliche Methoden

- virtual ~Publish ()

  Destruktor.
- virtual void attach (Observer *const obs)

  Fügt ein Observer Objekt in die Liste der zu benachrichtigenden Objekte hinzu.
- virtual bool detach (Observer *const obs)

  Entfernt das Observer Objekt aus der Liste der zu benachichtigenden Objekte.

Geschützte Methoden

- Publish ()

  Konstruktor.

- virtual void notify (void)

  Benachichtigt alle eingetragenen Observer Objekte.
- virtual void notify (const int ∗const iValues, const unsigned int iValuesSize)

  Benachichtigt alle eingetragenen Observer Objekte.
- virtual void notify (const float ∗const fValues, const unsigned int fValuesSize, const int ∗const iValues=NULL, const unsigned int iValuesSize=0)

  Benachichtigt alle eingetragenen Observer Objekte.

Private Attribute

- ObsListKnot_t ∗ firstObs

  Pointer auf das erste Listenelement.
- ObsListKnot_t ∗ lastObs

  Pointer auf das letzte Listenelement.

6.15.1 Ausführliche Beschreibung

Publisher Klasse für das Observer Entwurfsmodell.

Siehe auch

Observer

Definiert in Zeile 39 der Datei publish.h.

6.15.2 Beschreibung der Konstruktoren und Destruktoren

6.15.2.1 virtual Publish::∼Publish ( ) [inline], [virtual]

Destruktor.

Definiert in Zeile 45 der Datei publish.h.

6.15.2.2 Publish::Publish ( ) [inline], [protected]

Konstruktor.

Da die Liste noch leer ist wird das erste und letzte auf NULL gesetzt.

Definiert in Zeile 133 der Datei publish.h.

Benutzt firstObs, lastObs und NULL.

6.15.3 Dokumentation der Elementfunktionen

6.15.3.1 virtual void Publish::attach (Observer ∗const *obs*) [inline], [virtual]

Fügt ein Observer Objekt in die Liste der zu benachrichtigenden Objekte hinzu.

Parameter

in	*obs*	Pointer auf das in die Liste einzutragende Observer Element

Definiert in Zeile 53 der Datei publish.h.

Benutzt firstObs, lastObs, ObsListKnot::next, NULL, ObsListKnot::obs und ObsListKnot::prev.

6.15.3.2 virtual bool Publish::detach (Observer *const *obs*) [inline], [virtual]

Entfernt das Observer Objekt aus der Liste der zu benachichtigenden Objekte.

Parameter

in	*obs*	Pointer auf das aus der Liste zu entfernende Observer Element

Definiert in Zeile 80 der Datei publish.h.

Benutzt firstObs, lastObs, ObsListKnot::next, NULL, ObsListKnot::obs und ObsListKnot::prev.

6.15.3.3 virtual void Publish::notify (void) [inline], [protected], [virtual]

Benachichtigt alle eingetragenen Observer Objekte.

Siehe auch

Observer::update()

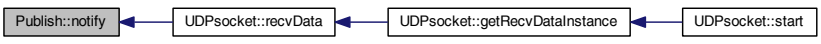
Definiert in Zeile 139 der Datei publish.h.

Benutzt firstObs, ObsListKnot::next, ObsListKnot::obs und Observer::update().

Wird benutzt von UDPsocket::recvData().

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

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



6.15.3.4 virtual void Publish::notify (const int *const *iValues*, const unsigned int *iValuesSize*) [inline], [protected], [virtual]

Benachichtigt alle eingetragenen Observer Objekte.

Siehe auch

Observer::update()

Definiert in Zeile 156 der Datei publish.h.

Benutzt firstObs, ObsListKnot::next, ObsListKnot::obs und Observer::update().

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

6.15.3.5 virtual void Publish::notify (const float *const *fValues*, const unsigned int *fValuesSize*, const int *const *iValues* = NULL, const unsigned int *iValuesSize* = 0) [inline], [protected], [virtual]

Benachichtigt alle eingetragenen Observer Objekte.

Siehe auch

Observer::update()

Definiert in Zeile 173 der Datei publish.h.

Benutzt firstObs, ObsListKnot::next, ObsListKnot::obs und Observer::update().

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

6.15.4 Dokumentation der Datenelemente

6.15.4.1 ObsListKnot_t* Publish::firstObs [private]

Pointer auf das erste Listenelement.

Definiert in Zeile 193 der Datei publish.h.

Wird benutzt von attach(), detach(), notify() und Publish().

6.15.4.2 ObsListKnot_t* Publish::lastObs [private]

Pointer auf das letzte Listenelement.

Definiert in Zeile 198 der Datei publish.h.

Wird benutzt von attach(), detach() und Publish().

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

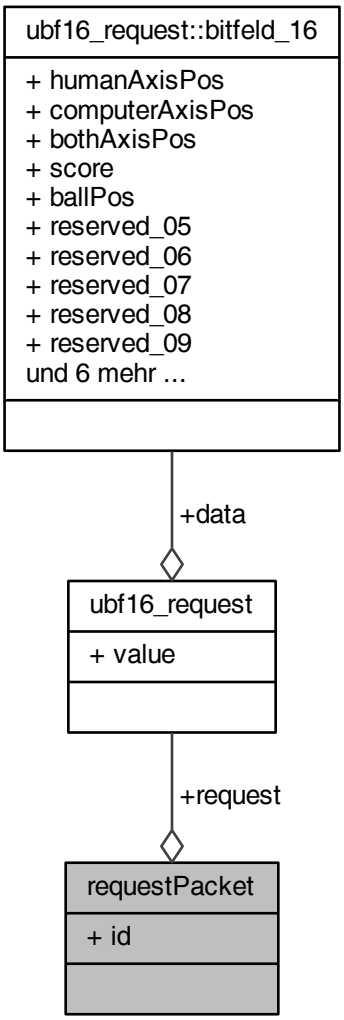
- publish.h

6.16 requestPacket Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von requestPacket:



Öffentliche Attribute

- unsigned int id: 16

  Paket-ID.

- ubf16_request request

  Bitfeld Bitfeld mit den Informationen, welche Daten angefragt werden sollen.

6.16.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Definiert in Zeile 215 der Datei defines.h.

6.16.2 Dokumentation der Datenelemente

6.16.2.1 unsigned int requestPacket::id

Paket-ID.

Definiert in Zeile 218 der Datei defines.h.

6.16.2.2 ubf16_request requestPacket::request

Bitfeld Bitfeld mit den Informationen, welche Daten angefragt werden sollen.

Definiert in Zeile 221 der Datei defines.h.

Wird benutzt von Server::update().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

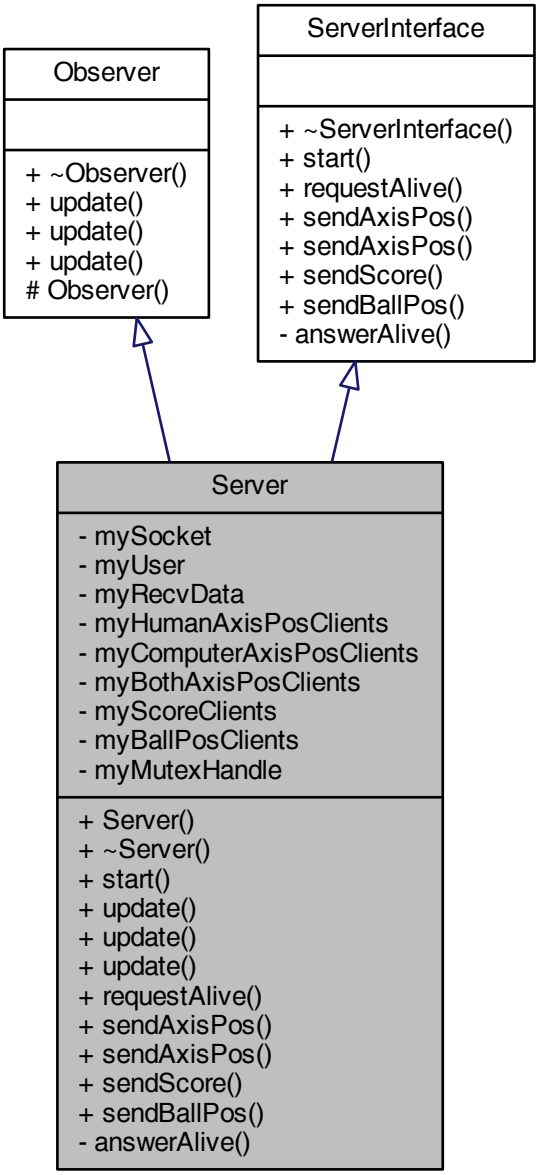
- defines.h

6.17 Server Klassenreferenz

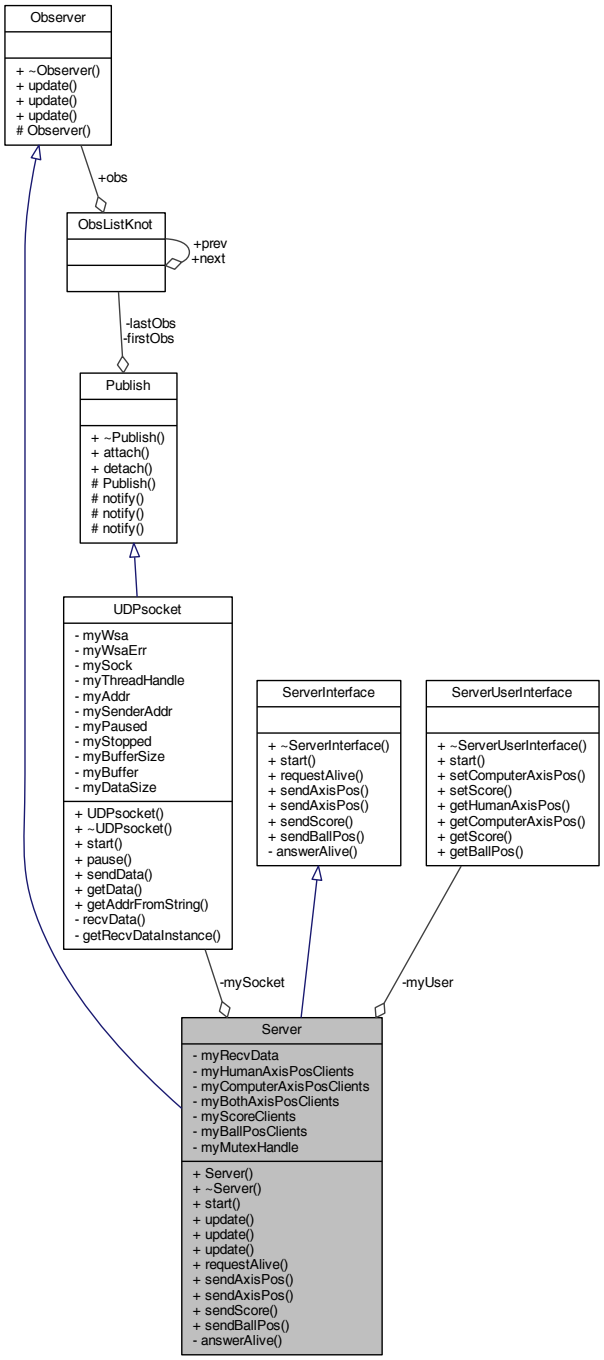
ein UDP-Server

```
#include <server.h>
```

Klassendiagramm für Server:



Zusammengehörigkeiten von Server:



Öffentliche Methoden

- Server (const unsigned short localPort)

  Konstruktor.
- ~Server ()

  Destruktor.
- bool start (ServerUserInterface ∗const user)

- void update (Publish ∗const pub)

  update-Funktion der Observer Klasse
- virtual void update (Publish ∗const pub, const int ∗const iValues, const unsigned int iValuesSize)

  Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.
- virtual void update (Publish ∗const pub, const float ∗const fValues, const unsigned int fValuesSize, const int ∗const iValues=NULL, const unsigned int iValuesSize=0)

  Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.
- list< sockaddr_in > requestAlive (const bool deleteClients=false)

  Sendet ein Alive-Anfragepaket und entfernt ggf. Clients.
- bool sendAxisPos (const unsigned int gamer, const float axisPos[4], const float axisAngle[4])

  sendet Spielstangenpositionen
- bool sendAxisPos (const unsigned int gamer, const float axisPos[2][4], const float axisAngle[2][4])

  sendet Spielstangenpositionen
- bool sendScore (const unsigned short computer, const unsigned short human)
- bool sendBallPos (const ballCoordinate_t ballPos)

Private Methoden

- bool answerAlive (const long aliveID, const sockaddr_in client)

  Beantwortet ein Alive-Paket.

Private Attribute

- UDPsocket ∗ mySocket

  die Klasse UDPsocket
- ServerUserInterface ∗ myUser

  Zeiger auf den Benutzer dieser Klasse.
- const char ∗ myRecvData

  Speicher für die empfangenen Daten.
- map< sockaddr_in, clientValues, addrLessThan > myHumanAxisPosClients

  Map mit allen Clients die die Spielstangenpositionen des Menschen abonniert haben.
- map< sockaddr_in, clientValues, addrLessThan > myComputerAxisPosClients

  Map mit allen Clients die die Spielstangenpositionen des Computer abonniert haben.
- map< sockaddr_in, clientValues, addrLessThan > myBothAxisPosClients

  Map mit allen Clients die die Spielstangenpositionen beider Spielter abonniert haben.
- map< sockaddr_in, clientValues, addrLessThan > myScoreClients

  Map mit allen Clients die den Spielstand abonniert haben.
- map< sockaddr_in, clientValues, addrLessThan > myBallPosClients

  Map mit allen Clients die die Ballposition abonniert haben.
- HANDLE myMutexHandle

  Mutex-Handle.

Weitere Geerbte Elemente

6.17.1 Ausführliche Beschreibung

ein UDP-Server

Beispielimplementierung eines UDP-Server unter Verwendung der Klasse UDPsocket.

Definiert in Zeile 71 der Datei server.h.

6.17.2 Beschreibung der Konstruktoren und Destruktoren

6.17.2.1 Server::Server (const unsigned short *localPort*)

Konstruktor.

Definiert in Zeile 18 der Datei server.cpp.

Benutzt NULL und RECV_BUFFER_SIZE.

6.17.2.2 Server::~Server ( )

Destruktor.

Definiert in Zeile 27 der Datei server.cpp.

Benutzt NULL.

6.17.3 Dokumentation der Elementfunktionen

6.17.3.1 bool Server::answerAlive (const long *aliveID,* const sockaddr_in *client*) [private], [virtual]

Beantwortet ein Alive-Paket.

Parameter

in	*aliveID*	eindeutige ID, die vom Client gesendet wurde um die Antwort der Anfrage zuzuordnen
in	*client*	Adresse des Clients, an den die Antwort gesendet werden soll

Rückgabe

true wenn die Antwort gesendet wurde

Implementiert ServerInterface.

Definiert in Zeile 379 der Datei server.cpp.

Benutzt ALIVE_ANSWER, long_1::id und long_1::lng.

6.17.3.2 list< sockaddr_in > Server::requestAlive (const bool *deleteClients* = false) [virtual]

Sendet ein Alive-Anfragepaket und entfernt ggf. Clients.

Parameter

in	*deleteClients*	Wenn true, werden Clients die nicht antworten gelöscht

Rückgabe

Eine Liste mit den Adressen der Clients, seit dem letzten Aufruf dieser Funktion kein ALIVE_ANSWER-Paket gesendet haben

Diese Funktion sendet ein ALIVE_REQUEST-Paket an alle Clients, die in den map's eingetragen sind. Wenn ein Client zwischen zwei Aufrufen dieser Funktion nicht mit einem ALIVE_ANSWER-Paket antwortet und deleteClients auf true gesetzt ist, so wird derjenige Client aus der entsprechenden map gelöscht.

Implementiert ServerInterface.

Definiert in Zeile 319 der Datei server.cpp.

Benutzt ALIVE_REQUEST, long_1::id, long_1::lng und NULL.

6.17.3.3 bool Server::sendAxisPos (const unsigned int *gamer*, const float *axisPos[4]*, const float *axisAngle[4]*) [virtual]

sendet Spielstangenpositionen

Parameter

in	*gamer*	HUMAN, COMPUTER oder BOTH, je nachdem für wen die Spielstangenpositionen gesendet werden sollen
in	*axisPos*	Array mit der Translation
in	*axisAngle*	Array mit der Rotation

Rückgabe

true wenn die Spielstangenpositionen gesendet wurden

Implementiert ServerInterface.

Definiert in Zeile 398 der Datei server.cpp.

Benutzt ANGLE, AXIS_FOUR, AXIS_ONE, COMPUTER, float_2_4::flt, GET_COMPUTER_AXISPOS, GET_HUMAN_AXISPOS, HUMAN, float_2_4::id und POSITION.

6.17.3.4 bool Server::sendAxisPos (const unsigned int *gamer*, const float *axisPos[2][4]*, const float *axisAngle[2][4]*) [virtual]

sendet Spielstangenpositionen

Parameter

in	*gamer*	HUMAN, COMPUTER oder BOTH, je nachdem für wen die Spielstangenpositionen gesendet werden sollen
in	*axisPos*	Array mit der Translation (Computer und Mensch)
in	*axisAngle*	Array mit der Rotation (Computer und Mensch)

Rückgabe

true wenn die Spielstangenpositionen gesendet wurden

Implementiert ServerInterface.

Definiert in Zeile 439 der Datei server.cpp.

Benutzt ANGLE, AXIS_FOUR, AXIS_ONE, COMPUTER, float_2_2_4::flt, GET_COMPUTER_AXISPOS, GET_HUMAN_AXISPOS, HUMAN, float_2_2_4::id und POSITION.

6.17.3.5 bool Server::sendBallPos (const ballCoordinate_t *ballPos*) [virtual]

Implementiert ServerInterface.

Definiert in Zeile 508 der Datei server.cpp.

Benutzt ballPacket::ballPos, GET_BALLPOS und ballPacket::id.

6.17.3.6 bool Server::sendScore (const unsigned short *computer,* const unsigned short *human*) [virtual]

Implementiert ServerInterface.

Definiert in Zeile 483 der Datei server.cpp.

Benutzt COMPUTER, GET_SCORE, HUMAN, unsignedShort_2::id und unsignedShort_2::us.

6.17.3.7 bool Server::start (ServerUserInterface ∗const *user*) [virtual]

Startet den Server

Implementiert ServerInterface.

Definiert in Zeile 38 der Datei server.cpp.

6.17.3.8 void Server::update (Publish ∗const *pub*) [virtual]

update-Funktion der Observer Klasse

Parameter

in	*pub*	Publisher Objekt, welches über ein Ereignis benachichtigt (hier UDPsocket)

Wird von UDPsocket aufgerufen wenn Daten eingetroffen sind.

Implementiert Observer.

Definiert in Zeile 55 der Datei server.cpp.

Benutzt ALIVE_ANSWER, ALIVE_REQUEST, ANGLE, ubf16_request::bitfeld_16::ballPos, ballPacket::ballPos, ubf16_request::bitfeld_16::bothAxisPos, COMPUTER, ubf16_request::bitfeld_16::computerAxisPos, ubf16_request::data, float_2_4::flt, float_2_2_4::flt, GET_BALLPOS, GET_BOTH_AXISPOS, GET_COMPUTER_AXISPOS, GET_HUMAN_AXISPOS, GET_SCORE, HUMAN, ubf16_request::bitfeld_16::humanAxisPos, unsignedShort_2::id, float_2_4::id, float_2_2_4::id, ballPacket::id, subscribePacket::initiate, long_1::lng, NULL, POSITION, REQUEST, requestPacket::request, ubf16_request::bitfeld_16::score, SET_AXISPOS, SET_SCORE, SUBSCRIBE, subscribePacket::subscribe und unsignedShort_2::us.

6.17.3.9 void Server::update (Publish ∗const *pub,* const int ∗const *iValues,* const unsigned int *iValuesSize*) [virtual]

Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.

Parameter

in	*pub*	Publisher Objekt, welches über ein Ereignis benachichtigt
in	*iValues*	int-Array zum übergeben von Variabeln
in	*iValuesSize*	Größe des Array values

Ruft hier nur update(Publish∗ const pub) auf, die anderen Parameter gehen verloren.

Implementiert Observer.

Definiert in Zeile 303 der Datei server.cpp.

6.17.3.10 void Server::update (Publish ∗const *pub*, const float ∗const *fValues*, const unsigned int *fValuesSize*, const int ∗const *iValues* = NULL, const unsigned int *iValuesSize* = 0) [virtual]

Wird von Publish aufgerufen wenn ein Ereignis eingetreten ist.

Parameter

in	*pub*	Publisher Objekt, welches über ein Ereignis benachichtigt
in	*fValues*	float Array zum übergeben von Variabeln
in	*fValuesSize*	Größe des Array values
in	*iValues*	int-Array zum übergeben von Variabeln
in	*iValuesSize*	Größe des Array values

Ruft hier nur update(Publish∗ const pub) auf, die anderen Parameter gehen verloren.

Implementiert Observer.

Definiert in Zeile 308 der Datei server.cpp.

6.17.4 Dokumentation der Datenelemente

6.17.4.1 map<sockaddr_in, clientValues, addrLessThan> Server::myBallPosClients [private]

Map mit allen Clients die die Ballposition abonniert haben.

Definiert in Zeile 213 der Datei server.h.

6.17.4.2 map<sockaddr_in, clientValues, addrLessThan> Server::myBothAxisPosClients [private]

Map mit allen Clients die die Spielstangenpositionen beider Spielter abonniert haben.

Definiert in Zeile 203 der Datei server.h.

6.17.4.3 map<sockaddr_in, clientValues, addrLessThan> Server::myComputerAxisPosClients [private]

Map mit allen Clients die die Spielstangenpositionen des Computer abonniert haben.

Definiert in Zeile 198 der Datei server.h.

6.17.4.4 map<sockaddr_in, clientValues, addrLessThan> Server::myHumanAxisPosClients [private]

Map mit allen Clients die die Spielstangenpositionen des Menschen abonniert haben.

Definiert in Zeile 193 der Datei server.h.

6.17.4.5 HANDLE Server::myMutexHandle [private]

Mutex-Handle.

Definiert in Zeile 218 der Datei server.h.

6.17.4.6 const char∗ Server::myRecvData [private]

Speicher für die empfangenen Daten.

Definiert in Zeile 188 der Datei server.h.

6.17.4.7 map<sockaddr_in, clientValues, addrLessThan> Server::myScoreClients [private]

Map mit allen Clients die den Spielstand abonniert haben.

Definiert in Zeile 208 der Datei server.h.

6.17.4.8 UDPsocket* Server::mySocket [private]

die Klasse UDPsocket

Definiert in Zeile 178 der Datei server.h.

6.17.4.9 ServerUserInterface* Server::myUser [private]

Zeiger auf den Benutzer dieser Klasse.

Definiert in Zeile 183 der Datei server.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

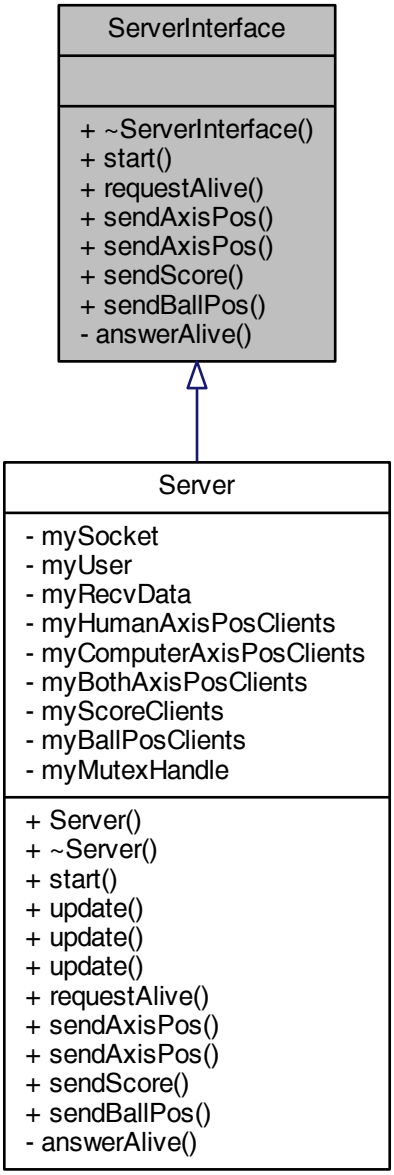
- server.h
- server.cpp

6.18 ServerInterface Schnittstellenreferenz

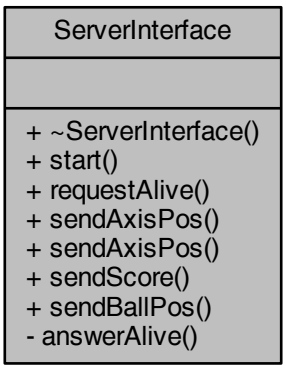
Interface zur Kommunikation mit dem UDP-Server.

`#include <serverinterface.h>`

Klassendiagramm für ServerInterface:



Zusammengehörigkeiten von ServerInterface:



Öffentliche Methoden

- virtual ~ServerInterface (void)
  Virtueller Destruktor.
- virtual bool start (ServerUserInterface *const userInterface)=0
  Startet den Server.
- virtual list< sockaddr_in > requestAlive (const bool deleteClients=false)=0
  Sendet ein Alive-Anfragepaket und entfernt ggf. Clients.
- virtual bool sendAxisPos (const unsigned int gamer, const float axisPos[4], const float axisAngle[4])=0
  sendet Spielstangenpositionen
- virtual bool sendAxisPos (const unsigned int gamer, const float axisPos[2][4], const float axisAngle[2][4])=0
  sendet Spielstangenpositionen
- virtual bool sendScore (const unsigned short computer, const unsigned short human)=0
- virtual bool sendBallPos (const ballCoordinate_t ballPos)=0

Private Methoden

- virtual bool answerAlive (const long aliveID, const sockaddr_in client)=0
  Beantwortet ein Alive-Paket.

6.18.1 Ausführliche Beschreibung

Interface zur Kommunikation mit dem UDP-Server.

Definiert in Zeile 23 der Datei serverinterface.h.

6.18.2 Beschreibung der Konstruktoren und Destruktoren

6.18.2.1 virtual ServerInterface::~ServerInterface (void) [inline], [virtual]

Virtueller Destruktor.

Definiert in Zeile 29 der Datei serverinterface.h.

6.18.3 Dokumentation der Elementfunktionen

6.18.3.1 virtual bool ServerInterface::answerAlive (const long *aliveID,* const sockaddr_in *client*) [private],[pure virtual]

Beantwortet ein Alive-Paket.

Parameter

in	*aliveID*	eindeutige ID, die vom Client gesendet wurde um die Antwort der Anfrage zuzuordnen
in	*client*	Adresse des Clients, an den die Antwort gesendet werden soll

Rückgabe

true wenn die Antwort gesendet wurde

Implementiert in Server.

6.18.3.2 virtual list<sockaddr_in> ServerInterface::requestAlive (const bool *deleteClients* = false) [pure virtual]

Sendet ein Alive-Anfragepaket und entfernt ggf. Clients.

Parameter

in	*deleteClients*	Wenn true, werden Clients die nicht antworten gelöscht

Rückgabe

Eine Liste mit den Adressen der Clients, seit dem letzten Aufruf dieser Funktion kein ALIVE_ANSWER-Paket gesendet haben

Diese Funktion sendet ein ALIVE_REQUEST-Paket an alle Clients, die in den map's eingetragen sind. Wenn ein Client zwischen zwei Aufrufen dieser Funktion nicht mit einem ALIVE_ANSWER-Paket antwortet und deleteClients auf true gesetzt ist, so wird derjenige Client aus der entsprechenden map gelöscht.

Implementiert in Server.

6.18.3.3 virtual bool ServerInterface::sendAxisPos (const unsigned int *gamer,* const float *axisPos[4],* const float *axisAngle[4]*) [pure virtual]

sendet Spielstangenpositionen

Parameter

in	*gamer*	HUMAN, COMPUTER oder BOTH, je nachdem für wen die Spielstangenpositionen gesendet werden sollen
in	*axisPos*	Array mit der Translation
in	*axisAngle*	Array mit der Rotation

Rückgabe

true wenn die Spielstangenpositionen gesendet wurden

Implementiert in Server.

6.18.3.4 virtual bool ServerInterface::sendAxisPos (const unsigned int *gamer,* const float *axisPos[2][4],* const float *axisAngle[2][4]*) [pure virtual]

sendet Spielstangenpositionen

Parameter

in	*gamer*	HUMAN, COMPUTER oder BOTH, je nachdem für wen die Spielstangenpositionen gesendet werden sollen
in	*axisPos*	Array mit der Translation (Computer und Mensch)
in	*axisAngle*	Array mit der Rotation (Computer und Mensch)

Rückgabe

true wenn die Spielstangenpositionen gesendet wurden

Implementiert in Server.

6.18.3.5 virtual bool ServerInterface::sendBallPos (const **ballCoordinate_t** *ballPos*) [pure virtual]

Implementiert in Server.

6.18.3.6 virtual bool ServerInterface::sendScore (const unsigned short *computer,* const unsigned short *human*) [pure virtual]

Implementiert in Server.

6.18.3.7 virtual bool ServerInterface::start (**ServerUserInterface** ∗const *userInterface*) [pure virtual]

Startet den Server.

Parameter

in	*userInterface*	Zeiger auf den Benutzer des Servers

Rückgabe

true wenn alles beim Starten des Servers geklappt hat

Implementiert in Server.

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

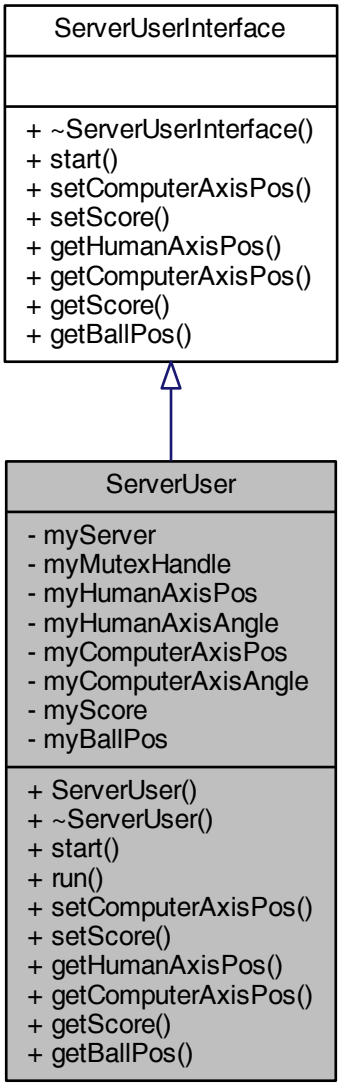
- serverinterface.h

6.19 ServerUser Klassenreferenz

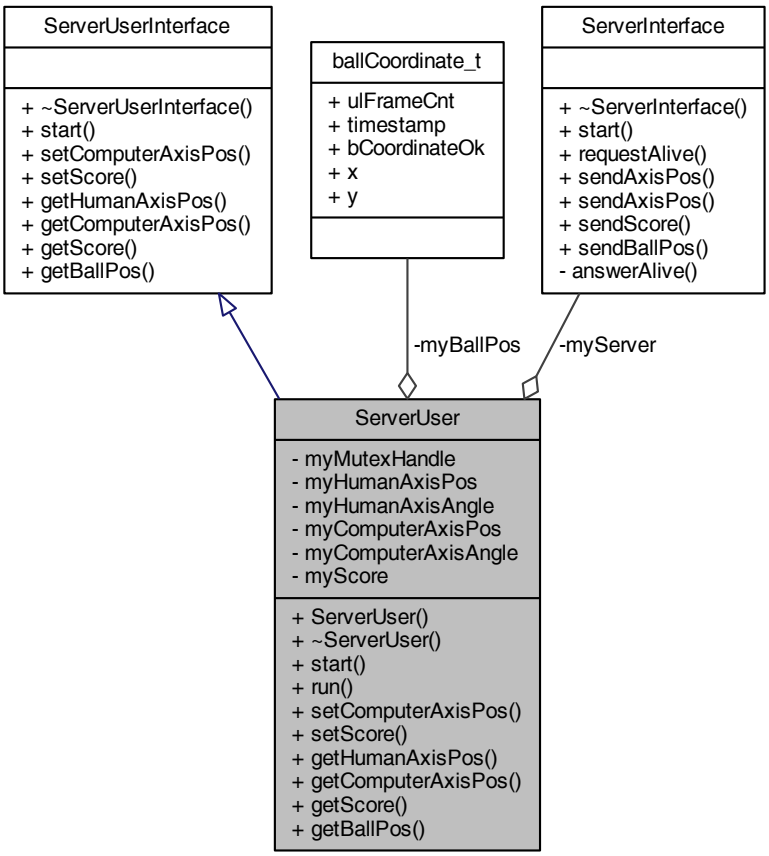
Eine Klasse, die den UDP-Server anlegt und benutzt.

`#include <serveruser.h>`

Klassendiagramm für ServerUser:



Zusammengehörigkeiten von ServerUser:



Öffentliche Methoden

- ServerUser ()

  Konstruktor.
- ~ServerUser ()

  Destruktor.
- bool start ()

  Startet den die Benutzung des Server.
- bool run ()

  Hier findet der Programmablauf stadt.
- bool setComputerAxisPos (const float axisPos[2][4], const float axisAngle[2][4])
- bool setScore (const unsigned short computer, const unsigned short human)
- bool getHumanAxisPos (float(&axisPos)[4], float(&axisAngle)[4])
- bool getComputerAxisPos (float(&axisPos)[4], float(&axisAngle)[4])
- bool getScore (unsigned short &human, unsigned short &computer)
- bool getBallPos (ballCoordinate_t &ballPos)

  leifert die Ballposition zurück

Private Attribute

- ServerInterface * myServer

  Zeiger auf den Server.
- HANDLE myMutexHandle

  Mutex-Handle.
- float myHumanAxisPos [4]

  Spielstangenpositionen (Translation) des Menschen.
- float myHumanAxisAngle [4]

  Spielstangenpositionen (Rotation) des Menschen.
- float myComputerAxisPos [2][4]

  Spielstangenpositionen (Translation) des Computers.
- float myComputerAxisAngle [2][4]

  Spielstangenpositionen (Rotation) des Computers.
- unsigned myScore [2]

  Spielstand.
- ballCoordinate_t myBallPos

  Ballposition.

6.19.1 Ausführliche Beschreibung

Eine Klasse, die den UDP-Server anlegt und benutzt.

Beispielimplementierung wie der UDP-Server zu verwenden ist.

Definiert in Zeile 26 der Datei serveruser.h.

6.19.2 Beschreibung der Konstruktoren und Destruktoren

6.19.2.1 ServerUser::ServerUser ( )

Konstruktor.

Legt die Klasse Server an.

Definiert in Zeile 17 der Datei serveruser.cpp.

Benutzt AXIS_FOUR, AXIS_ONE, AXIS_THREE, AXIS_TWO, COMPUTER, FIELD_X_MAX, FIELD_Y_MAX, HUMAN, MAX_VELOCITY_ROT, MAX_VELOCITY_TRANS, NULL, SERVER_PORT, TRAVERSE_DISTANCE_1, TRAVERSE_DISTANCE_2, TRAVERSE_DISTANCE_3, TRAVERSE_DISTANCE_4, VALUE und VELOCITY.

6.19.2.2 ServerUser::~ServerUser ( )

Destruktor.

Löscht die Klasse Server.

Definiert in Zeile 46 der Datei serveruser.cpp.

Benutzt NULL.

6.19.3 Dokumentation der Elementfunktionen

6.19.3.1 bool ServerUser::getBallPos (ballCoordinate_t & *ballPos*) [virtual]

leifert die Ballposition zurück

Parameter

in	*ballPos*	Referenz auf die Variable in die die Ballposition abgelegt werden soll

Rückgabe

true wenn die Ballposition erfolgreich übergeben wurde

Wird vom Server aufgerufen wenn die Ballposition vom Client abgefragt wird.

Implementiert ServerUserInterface.

Definiert in Zeile 570 der Datei serveruser.cpp.

6.19.3.2 bool ServerUser::getComputerAxisPos (float(&) *axisPos[4],* float(&) *axisAngle[4]*) [virtual]

Implementiert ServerUserInterface.

Definiert in Zeile 517 der Datei serveruser.cpp.

Benutzt AXIS_FOUR, AXIS_ONE und VALUE.

6.19.3.3 bool ServerUser::getHumanAxisPos (float(&) *axisPos[4],* float(&) *axisAngle[4]*) [virtual]

Implementiert ServerUserInterface.

Definiert in Zeile 489 der Datei serveruser.cpp.

Benutzt AXIS_FOUR und AXIS_ONE.

6.19.3.4 bool ServerUser::getScore (unsigned short & *human,* unsigned short & *computer*) [virtual]

Implementiert ServerUserInterface.

Definiert in Zeile 545 der Datei serveruser.cpp.

Benutzt COMPUTER und HUMAN.

6.19.3.5 bool ServerUser::run ( )

Hier findet der Programmablauf stadt.

Rückgabe

true wenn der Ablauf ohne Fehler beendet wurde

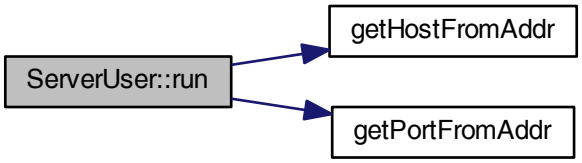
Noch zu erledigen Konsolenausgabe in Funktionen auslagern.

Definiert in Zeile 62 der Datei serveruser.cpp.

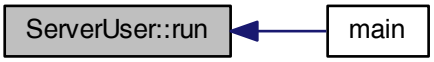
Benutzt ANGLE, AXIS_FOUR, AXIS_ONE, AXIS_THREE, AXIS_TWO, ballCoordinate_t::bCoordinateOk, BOTH, COMPUTER, getHostFromAddr(), getPortFromAddr(), HUMAN, POSITION, ballCoordinate_t::timestamp, ballCoordinate_t::ulFrameCnt, VALUE, VELOCITY, ballCoordinate_t::x und ballCoordinate_t::y.

Wird benutzt von main().

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



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



6.19.3.6 bool ServerUser::setComputerAxisPos (const float *axisPos[2][4],* const float *axisAngle[2][4]*) [virtual]

Implementiert ServerUserInterface.

Definiert in Zeile 430 der Datei serveruser.cpp.

Benutzt AXIS_FOUR, AXIS_ONE, VALUE und VELOCITY.

6.19.3.7 bool ServerUser::setScore (const unsigned short *computer,* const unsigned short *human*) [virtual]

Implementiert ServerUserInterface.

Definiert in Zeile 461 der Datei serveruser.cpp.

Benutzt AXIS_FOUR, AXIS_ONE, COMPUTER und HUMAN.

6.19.3.8 bool ServerUser::start ( ) [virtual]

Startet den die Benutzung des Server.

Rückgabe

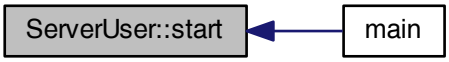
true wenn alles beim Starten des Server geklappt hat

Implementiert ServerUserInterface.

Definiert in Zeile 54 der Datei serveruser.cpp.

Wird benutzt von main().

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



6.19.4 Dokumentation der Datenelemente

6.19.4.1 ballCoordinate_t ServerUser::myBallPos [private]

Ballposition.

Definiert in Zeile 179 der Datei serveruser.h.

6.19.4.2 float ServerUser::myComputerAxisAngle[2][4] [private]

Spielstangenpositionen (Rotation) des Computers.

Der erste Index gibt an ob es sich um den Wert [0] oder die Geschwindigkeit [1] handelt. Der zweite Index gibt an um welche Spielstange es sich handelt (Torward [0], Abwehr [1]. Mittelfeld[2], Angriff[3]).

Definiert in Zeile 167 der Datei serveruser.h.

6.19.4.3 float ServerUser::myComputerAxisPos[2][4] [private]

Spielstangenpositionen (Translation) des Computers.

Der erste Index gibt an ob es sich um den Wert [0] oder die Geschwindigkeit [1] handelt. Der zweite Index gibt an um welche Spielstange es sich handelt (Torward [0], Abwehr [1]. Mittelfeld[2], Angriff[3]).

Definiert in Zeile 158 der Datei serveruser.h.

6.19.4.4 float ServerUser::myHumanAxisAngle[4] [private]

Spielstangenpositionen (Rotation) des Menschen.

Der Index gibt an um welche Spielstange es sich handelt (Torward [0], Abwehr [1]. Mittelfeld[2], Angriff[3]).

Definiert in Zeile 149 der Datei serveruser.h.

6.19.4.5 float ServerUser::myHumanAxisPos[4] [private]

Spielstangenpositionen (Translation) des Menschen.

Der Index gibt an um welche Spielstange es sich handelt (Torward [0], Abwehr [1]. Mittelfeld[2], Angriff[3]).

Definiert in Zeile 141 der Datei serveruser.h.

6.19.4.6 HANDLE ServerUser::myMutexHandle [private]

Mutex-Handle.

Mutex-Handle zum sicheren Datenzugriff da die Funktionen aus ServerUserInterface (ausser start()) von einem anderen Thread aufgerugen werden.

Definiert in Zeile 133 der Datei serveruser.h.

6.19.4.7 unsigned ServerUser::myScore[2] [private]

Spielstand.

Der Index gibt an ob es sich um den Computer [0] oder den menschlichen Spieler [1] handelt.

Definiert in Zeile 174 der Datei serveruser.h.

6.19.4.8 ServerInterface* ServerUser::myServer [private]

Zeiger auf den Server.

Definiert in Zeile 125 der Datei serveruser.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

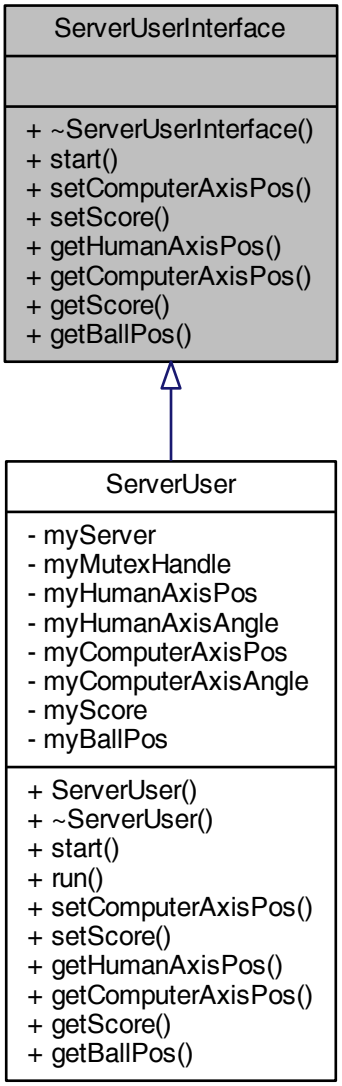
- serveruser.h
- serveruser.cpp

6.20 ServerUserInterface Schnittstellenreferenz

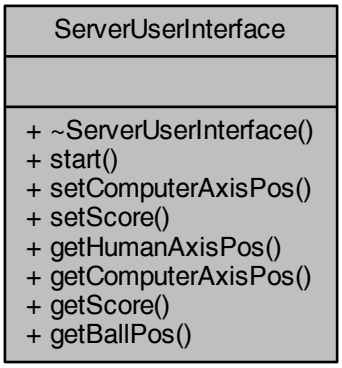
Interface zur Kommunikation mit dem Benutzer des UDP-Servers.

`#include <serveruserinterface.h>`

Klassendiagramm für ServerUserInterface:



Zusammengehörigkeiten von ServerUserInterface:



Öffentliche Methoden

- virtual ~ServerUserInterface (void)

  Virtueller Destruktor.
- virtual bool start ()=0

  Startet den die Benutzung des Servers.
- virtual bool setComputerAxisPos (const float axisPos[2][4], const float axisAngle[2][4])=0
- virtual bool setScore (const unsigned short computer, const unsigned short human)=0
- virtual bool getHumanAxisPos (float(&axisPos)[4], float(&axisAngle)[4])=0
- virtual bool getComputerAxisPos (float(&axisPos)[4], float(&axisAngle)[4])=0
- virtual bool getScore (unsigned short &computer, unsigned short &human)=0
- virtual bool getBallPos (ballCoordinate_t &ballPos)=0

  leifert die Ballposition zurück

6.20.1 Ausführliche Beschreibung

Interface zur Kommunikation mit dem Benutzer des UDP-Servers.

Definiert in Zeile 19 der Datei serveruserinterface.h.

6.20.2 Beschreibung der Konstruktoren und Destruktoren

6.20.2.1 virtual ServerUserInterface::~ServerUserInterface (void) [inline], [virtual]

Virtueller Destruktor.

Definiert in Zeile 25 der Datei serveruserinterface.h.

6.20.3 Dokumentation der Elementfunktionen

6.20.3.1 virtual bool ServerUserInterface::getBallPos (ballCoordinate_t & *ballPos*) [pure virtual]

leifert die Ballposition zurück

Parameter

in	*ballPos*	Referenz auf die Variable in die die Ballposition abgelegt werden soll

Rückgabe

true wenn die Ballposition erfolgreich übergeben wurde

Wird vom Server aufgerufen wenn die Ballposition vom Client abgefragt wird.

Implementiert in ServerUser.

6.20.3.2 virtual bool ServerUserInterface::getComputerAxisPos (float(&) *axisPos[4],* float(&) *axisAngle[4]*) [pure virtual]

Implementiert in ServerUser.

6.20.3.3 virtual bool ServerUserInterface::getHumanAxisPos (float(&) *axisPos[4],* float(&) *axisAngle[4]*) [pure virtual]

Implementiert in ServerUser.

6.20.3.4 virtual bool ServerUserInterface::getScore (unsigned short & *computer,* unsigned short & *human*) [pure virtual]

Implementiert in ServerUser.

6.20.3.5 virtual bool ServerUserInterface::setComputerAxisPos (const float *axisPos[2][4],* const float *axisAngle[2][4]*) [pure virtual]

Implementiert in ServerUser.

6.20.3.6 virtual bool ServerUserInterface::setScore (const unsigned short *computer,* const unsigned short *human*) [pure virtual]

Implementiert in ServerUser.

6.20.3.7 virtual bool ServerUserInterface::start ( ) [pure virtual]

Startet den die Benutzung des Servers.

Rückgabe

true wenn alles beim Starten des Servers geklappt hat

Implementiert in ServerUser.

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

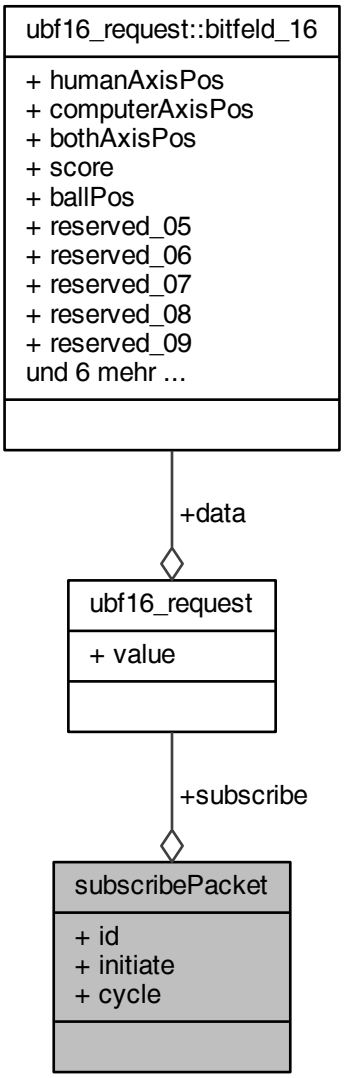
- serveruserinterface.h

6.21 subscribePacket Strukturreferenz

Abonnieren-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von subscribePacket:



Öffentliche Attribute

- unsigned int id: 16

  Paket-ID.
- ubf16_request subscribe

  Bitfeld Bitfeld mit den Informationen, welche Daten abonniert werden sollen.
- bool initiate

Abonnieren oder kündigen Information, ob die Daten abonniert (true) oder gekündigt (false) werden sollen.

- unsigned int cycle

  Frequenz Zykluszeit in ms, mit der die Daten versendet werden sollen. Wird nicht verwendet!

6.21.1 Ausführliche Beschreibung

Abonnieren-Paket für bestimmte Daten.

Definiert in Zeile 227 der Datei defines.h.

6.21.2 Dokumentation der Datenelemente

6.21.2.1 unsigned int subscribePacket::cycle

Frequenz Zykluszeit in ms, mit der die Daten versendet werden sollen. Wird nicht verwendet!

Noch zu erledigen Überlegen ob dies umgesetzt werden soll und wenn ja, dann umsetzen.

Definiert in Zeile 243 der Datei defines.h.

6.21.2.2 unsigned int subscribePacket::id

Paket-ID.

Definiert in Zeile 230 der Datei defines.h.

6.21.2.3 bool subscribePacket::initiate

Abonnieren oder kündigen Information, ob die Daten abonniert (true) oder gekündigt (false) werden sollen.

Definiert in Zeile 236 der Datei defines.h.

Wird benutzt von Server::update().

6.21.2.4 ubf16_request subscribePacket::subscribe

Bitfeld Bitfeld mit den Informationen, welche Daten abonniert werden sollen.

Definiert in Zeile 233 der Datei defines.h.

Wird benutzt von Server::update().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

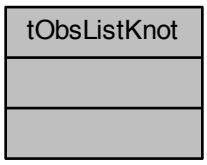
- defines.h

6.22 tObsListKnot Strukturreferenz

Struktur für ein Listenelement.

`#include <publish.h>`

Zusammengehörigkeiten von tObsListKnot:



6.22.1 Ausführliche Beschreibung

Struktur für ein Listenelement.

Struktur für ein Listenelement einer doppelt verketteten Liste, welche alle Observer Elemente enthält, die sich angemeldet haben.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

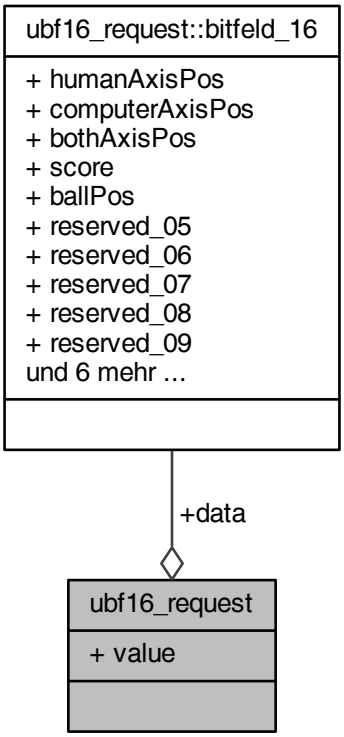
- publish.h

6.23 ubf16_request Variantenreferenz

Bitfeld, bei dem jedes Bit bestimmte Daten anfordert.

`#include <defines.h>`

Zusammengehörigkeiten von ubf16_request:



Klassen

- struct bitfeld_16

Öffentliche Attribute

- unsigned int value: 16
- struct ubf16_request::bitfeld_16 data

6.23.1 Ausführliche Beschreibung

Bitfeld, bei dem jedes Bit bestimmte Daten anfordert.

Definiert in Zeile 188 der Datei defines.h.

6.23.2 Dokumentation der Datenelemente

6.23.2.1 struct ubf16_request::bitfeld_16 ubf16_request::data

Wird benutzt von Server::update().

6.23.2.2 unsigned int ubf16_request::value

Definiert in Zeile 190 der Datei defines.h.

Die Dokumentation für diese Variante wurde erzeugt aufgrund der Datei:

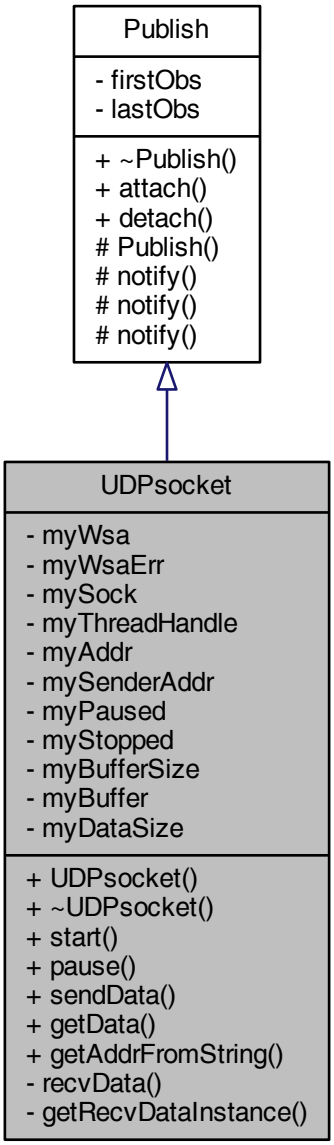
- defines.h

6.24 UDPsocket Klassenreferenz

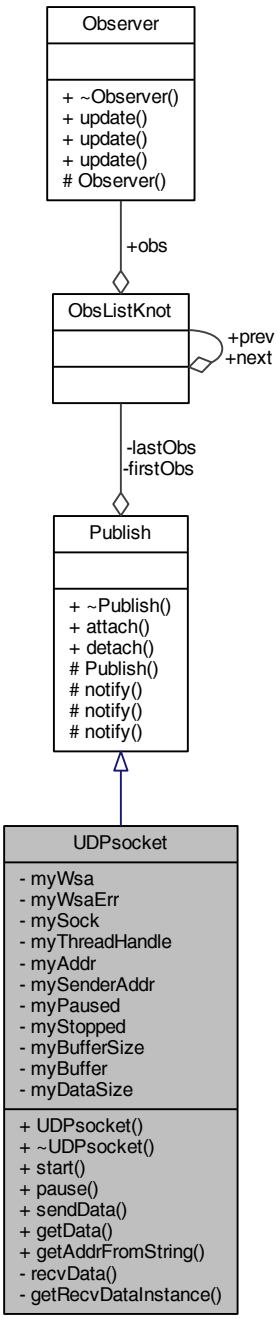
Stellt ein UDP-Socket dar.

`#include <udpsocket.h>`

Klassendiagramm für UDPsocket:



Zusammengehörigkeiten von UDPsocket:



Öffentliche Methoden

- UDPsocket (const unsigned int bufferSize, const unsigned short localPort)

  Konstruktor.
- ~UDPsocket ()

  Destruktor.
- bool start ()

Startet die Socketfunktion.

- void pause (const bool pause=true)

  Pausiert den Empfang von Daten.

- int sendData (const char *const data, const int len, const sockaddr_in recvr)

  Sendet Daten an einen anderen Socket.

- int getData (const char *&data, sockaddr_in &sendr)

  Liefert die empfangenen Daten.

- bool getAddrFromString (const string host, sockaddr_in &addr)

  Ermittelt die IP-Adresse aus einer Zeichenkette.

Private Methoden

- DWORD recvData ()

  Eigener Thread, welcher Pakete empfängt.

Private, statische Methoden

- static DWORD WINAPI getRecvDataInstance (void *instance)

  Gibt den Funktionspointer auf die Funktion recvData() zurück.

Private Attribute

- WSADATA myWsa

  Struktur, welche Informationen über Winsock enthält.

- int myWsaErr

  Wird beim Initialisieren von Winsock gesetzt.

- SOCKET mySock

  Socket Deskriptor.

- HANDLE myThreadHandle

  Handle vom Thread.

- sockaddr_in myAddr

  eigene Adresse

- sockaddr_in mySenderAddr

  Sender-Adresse.

- bool myPaused

  Wird auf true gesetzt wenn der Daten-Empfang pausiert werden soll.

- bool myStopped

  Wird auf true gesetzt wenn der Daten-Empfang entgültig beendet wird.

- unsigned int myBufferSize

  Größe des Empfangsspeichers.

- char * myBuffer

  Empfanfsspeicher.

- int myDataSize

  Anzahl der empfangenen Bytes.

Weitere Geerbte Elemente

6.24.1 Ausführliche Beschreibung

Stellt ein UDP-Socket dar.

Diese Klasse stellt ein UDP-Socket dar, sie kann Pakete an einen anderen Host senden und von diesem empfangen. Ob die Klasse als Server oder Client verwendet wird, hängt davon ab wie sie aufgerufen wird.

Definiert in Zeile 47 der Datei udpsocket.h.

6.24.2 Beschreibung der Konstruktoren und Destruktoren

6.24.2.1 UDPsocket::UDPsocket (const unsigned int *bufferSize,* const unsigned short *localPort*)

Konstruktor.

Parameter

in	*bufferSize*	Größe des Empfangsspeichers (maximal zu erwartende Datenmenge)
in	*localPort*	Port des eigenen Sockets

Bereitet Winsock vor und wandelt die lokale Host-Adresse um.

Definiert in Zeile 15 der Datei udpsocket.cpp.

Benutzt myAddr, myPaused, myStopped, myWsa und myWsaErr.

6.24.2.2 UDPsocket::~UDPsocket ( )

Destruktor.

Beendet den Daten-Empfang, räumt Winsock auf und gibt den Empfangsspeicher wieder frei.

Definiert in Zeile 29 der Datei udpsocket.cpp.

Benutzt myBuffer, myPaused, mySock, myStopped, myThreadHandle und NULL.

6.24.3 Dokumentation der Elementfunktionen

6.24.3.1 bool UDPsocket::getAddrFromString (const string *host,* sockaddr_in & *addr*)

Ermittelt die IP-Adresse aus einer Zeichenkette.

Parameter

in	*host*	Zeichenkette mit der IP-Adresse oder dem Rechnernamen
out	*addr*	Referenz auf die Struktur in der die ermittelte Adresse abgelegt werden soll

Rückgabe

true wenn die Adresse ermittelt werden konnte, ansonsten false

Definiert in Zeile 159 der Datei udpsocket.cpp.

Benutzt NULL.

6.24.3.2 int UDPsocket::getData (const char *& *data,* sockaddr_in & *sendr*)

Liefert die empfangenen Daten.

Parameter

out	*data*	Referenz auf den Zeiger auf den Speicherbereich, in dem die Daten liegen
out	*sendr*	Referenz auf die Variable, in die die Adresse des Senders gespeichert wird

Rückgabe

Anzahl der empfangenen Bytes

Wenn die empfangen Daten ausserhalb von der update()-Funktion des Observers verwendet werden sollen, müssen die Daten kopiert werden da der Speicherbereich beim erneuten eintreffen von Daten überschrieben wird.

Definiert in Zeile 152 der Datei udpsocket.cpp.

Benutzt myBuffer, myDataSize und mySenderAddr.

6.24.3.3 DWORD WINAPI UDPsocket::getRecvDataInstance (void * *instance*) [static], [private]

Gibt den Funktionspointer auf die Funktion recvData() zurück.

Parameter

in	*instance*	Pointer auf die Instanz dieser Klasse

Rückgabe

Funktionspointer auf die Funktion recvData() der aktuellen Instanz

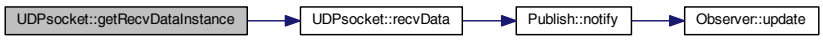
Trick um einen neuen Thread mit einer Memberfunktion zu erzeugen.

Definiert in Zeile 244 der Datei udpsocket.cpp.

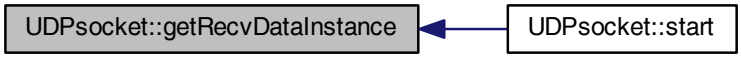
Benutzt NULL und recvData().

Wird benutzt von start().

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



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



6.24.3.4 void UDPsocket::pause (const bool *pause* = true)

Pausiert den Empfang von Daten.

Parameter

in	*pause*	bei true wird der Empfang von Daten pausiert, bei false wird der Empfang fortgesetzt

Wenn pausiert werden weiterhin Pakete empfangen, allerdings wird der Observer darüber nur nicht mehr informiert.

Definiert in Zeile 136 der Datei udpsocket.cpp.

Benutzt myPaused.

6.24.3.5 DWORD UDPsocket::recvData ( ) [private]

Eigener Thread, welcher Pakete empfängt.

Eigener Thread, welcher Pakete empfängt und alle angemeldeten Observer Objekte über das Eintreffen neuer Pakete informiert.

Siehe auch

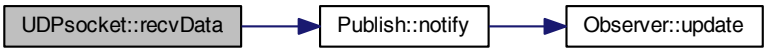
Publish::notify()

Definiert in Zeile 202 der Datei udpsocket.cpp.

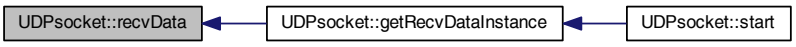
Benutzt myBuffer, myBufferSize, myDataSize, myPaused, mySenderAddr, mySock, myStopped, Publish::notify() und NULL.

Wird benutzt von getRecvDataInstance().

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



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



6.24.3.6 int UDPsocket::sendData (const char *const *data*, const int *len*, const sockaddr_in *recvr*)

Sendet Daten an einen anderen Socket.

Parameter

in	*data*	Pointer auf die zu sendenden Daten
in	*len*	Anzahl der zu sendenden Bytes
in	*recvr*	Adresse des Empfängers der Daten

Rückgabe

Anzahl der tatsächlich gesendeten Bytes

Definiert in Zeile 142 der Datei udpsocket.cpp.

Benutzt mySock.

6.24.3.7 bool UDPsocket::start ( )

Startet die Socketfunktion.

Rückgabe

true bei Erfolg, ansonsten false

Überprüft ob Winsock richtig vorbereitet wurde, legt den Socket an und bindet diesen, legt den Empfangsspeicher an und startet den Thread, der die Daten empfängt.

Definiert in Zeile 85 der Datei udpsocket.cpp.

Benutzt getRecvDataInstance(), myAddr, myBuffer, myBufferSize, myDataSize, myPaused, mySock, myStopped, myThreadHandle, myWsa, myWsaErr und NULL.

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

6.24.4 Dokumentation der Datenelemente

6.24.4.1 sockaddr_in UDPsocket::myAddr [private]

eigene Adresse

Siehe auch

UDPsocket::UDPsocket()

Definiert in Zeile 154 der Datei udpsocket.h.

Wird benutzt von start() und UDPsocket().

6.24.4.2 char* UDPsocket::myBuffer [private]

Empfanfsspeicher.

Siehe auch

UDPsocket::recvData()
UDPsocket::getData()

Empfanfsspeicher in dem die empfangenen Daten abgelegt werden.

Definiert in Zeile 191 der Datei udpsocket.h.

Wird benutzt von getData(), recvData(), start() und ~UDPsocket().

6.24.4.3 unsigned int UDPsocket::myBufferSize [private]

Größe des Empfangsspeichers.

Siehe auch

UDPsocket::UDPsocket()

Definiert in Zeile 182 der Datei udpsocket.h.

Wird benutzt von recvData() und start().

6.24.4.4 int UDPsocket::myDataSize [private]

Anzahl der empfangenen Bytes.

Siehe auch

UDPsocket::recvData()
UDPsocket::getData()
UDPsocket::myBuffer

Anzahl der Bytes die empfangen wurden und im Empfangsspeicher abgelegt sind.

Definiert in Zeile 201 der Datei udpsocket.h.

Wird benutzt von getData(), recvData() und start().

6.24.4.5 bool UDPsocket::myPaused [private]

Wird auf true gesetzt wenn der Daten-Empfang pausiert werden soll.

Siehe auch

UDPclient::pause()
UDPclient::recvData()

Definiert in Zeile 169 der Datei udpsocket.h.

Wird benutzt von pause(), recvData(), start(), UDPsocket() und ~UDPsocket().

6.24.4.6 sockaddr_in UDPsocket::mySenderAddr [private]

Sender-Adresse.

Siehe auch

UDPsocket::recvData()

Sender-Adresse von empfangenen Paketen.

Definiert in Zeile 162 der Datei udpsocket.h.

Wird benutzt von getData() und recvData().

6.24.4.7 SOCKET UDPsocket::mySock [private]

Socket Deskriptor.

Wird in UDPsocket::start() beim Anlegen des Sockets gesetzt.

Definiert in Zeile 139 der Datei udpsocket.h.

Wird benutzt von recvData(), sendData(), start() und ~UDPsocket().

6.24.4.8 bool UDPsocket::myStopped [private]

Wird auf true gesetzt wenn der Daten-Empfang entgültig beendet wird.

Siehe auch

UDPclient::∼UDPclient()
UDPclient::recvData()

Definiert in Zeile 176 der Datei udpsocket.h.

Wird benutzt von recvData(), start(), UDPsocket() und ∼UDPsocket().

6.24.4.9 HANDLE UDPsocket::myThreadHandle [private]

Handle vom Thread.

Handle vom Thread welcher zum Empfangen von Paketen zuständig ist. Wird in UDPsocket::start() beim Erzeugen des Threads gesetzt. Wird benutzt um im Destruktor auf das Ende des Threads zu warten.

Definiert in Zeile 148 der Datei udpsocket.h.

Wird benutzt von start() und ∼UDPsocket().

6.24.4.10 WSADATA UDPsocket::myWsa [private]

Struktur, welche Informationen über Winsock enthält.

Wird in UDPsocket::UDPsocket() gesetzt und in UDPsocket::start() überprüft

Definiert in Zeile 123 der Datei udpsocket.h.

Wird benutzt von start() und UDPsocket().

6.24.4.11 int UDPsocket::myWsaErr [private]

Wird beim Initialisieren von Winsock gesetzt.

Wird beim erfolgreichen Initialisieren von Winsock in UDPsocket::UDPsocket() auf Null (0) gesetzt. Enthält im Fehlerfall einen Fehlercode.

Definiert in Zeile 132 der Datei udpsocket.h.

Wird benutzt von start() und UDPsocket().

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

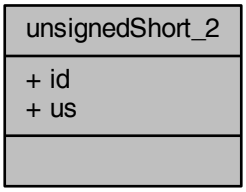
- udpsocket.h
- udpsocket.cpp

6.25 unsignedShort_2 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von unsignedShort_2:



Öffentliche Attribute

- unsigned int id: 16
- unsigned short us [2]

6.25.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein Array aus zwei unsigned short-Werten.

Definiert in Zeile 284 der Datei defines.h.

6.25.2 Dokumentation der Datenelemente

6.25.2.1 unsigned int unsignedShort_2::id

Definiert in Zeile 286 der Datei defines.h.

Wird benutzt von Server::sendScore() und Server::update().

6.25.2.2 unsigned short unsignedShort_2::us[2]

Definiert in Zeile 287 der Datei defines.h.

Wird benutzt von Server::sendScore() und Server::update().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- defines.h

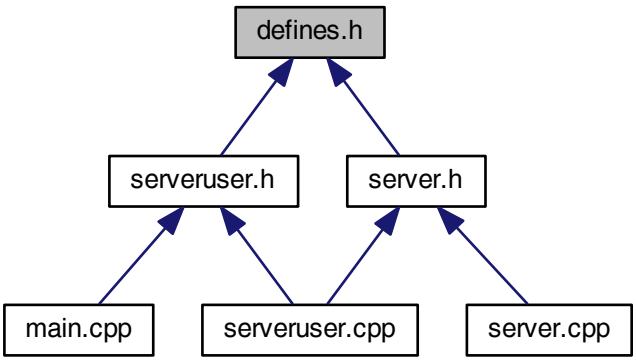
Kapitel 7

Datei-Dokumentation

7.1 defines.h-Dateireferenz

Enthält die defines der verschiedenen Strukturen.

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



Klassen

- union ubf16_request

  Bitfeld, bei dem jedes Bit bestimmte Daten anfordert.
- struct ubf16_request::bitfeld_16
- struct requestPacket

  Anfrage-Paket für bestimmte Daten.
- struct subscribePacket

  Abonnieren-Paket für bestimmte Daten.
- struct long_1

  Anfrage-Paket für bestimmte Daten.
- struct bool_1

  Anfrage-Paket für bestimmte Daten.
- struct bool_2

Anfrage-Paket für bestimmte Daten.

- struct unsignedShort_2

  Anfrage-Paket für bestimmte Daten.
- struct float_1

  Anfrage-Paket für bestimmte Daten.
- struct float_4

  Anfrage-Paket für bestimmte Daten.
- struct float_2_4

  Anfrage-Paket für bestimmte Daten.
- struct float_2_2_4

  Anfrage-Paket für bestimmte Daten.
- struct ballCoordinate_t

  Struktur der Ballposition.
- struct ballPacket

  Paket mit der Ballposition.

Makrodefinitionen

- #define SERVER_HOST "127.0.0.1"

  Host des Servers.

Typdefinitionen

- typedef union ubf16_request ubf16_request

  Bitfeld, bei dem jedes Bit bestimmte Daten anfordert.
- typedef struct requestPacket requestPacket

  Anfrage-Paket für bestimmte Daten.
- typedef struct subscribePacket subscribePacket

  Abonnieren-Paket für bestimmte Daten.
- typedef struct long_1 long_1

  Anfrage-Paket für bestimmte Daten.
- typedef struct bool_1 bool_1

  Anfrage-Paket für bestimmte Daten.
- typedef struct bool_2 bool_2

  Anfrage-Paket für bestimmte Daten.
- typedef struct unsignedShort_2 unsignedShort_2

  Anfrage-Paket für bestimmte Daten.
- typedef struct float_1 float_1

  Anfrage-Paket für bestimmte Daten.
- typedef struct float_4 float_4

  Anfrage-Paket für bestimmte Daten.
- typedef struct float_2_4 float_2_4

  Anfrage-Paket für bestimmte Daten.
- typedef struct float_2_2_4 float_2_2_4

  Anfrage-Paket für bestimmte Daten.
- typedef struct ballCoordinate_t ballCoordinate_t

  Struktur der Ballposition.
- typedef struct ballPacket ballPacket

  Paket mit der Ballposition.

Aufzählungen

- enum Axis_Number { AXIS_ONE, AXIS_TWO, AXIS_THREE, AXIS_FOUR }

  Spielstangen.
- enum Gamer { COMPUTER, HUMAN, BOTH }

  Spieler.
- enum Data_Value { POSITION, ANGLE }

  Um welchen Wert es sich handelt.
- enum Data_Type { VALUE, VELOCITY }

  Um welchen Wert es sich handelt.
- enum Safety { STOP, GRID }

  Um welche Sicherheitsfunktion es sich handelt.
- enum PacketID {
  ALIVE_REQUEST, ALIVE_ANSWER, REQUEST, SUBSCRIBE,
  SET_AXISPOS, SET_SCORE, GET_HUMAN_AXISPOS, GET_COMPUTER_AXISPOS,
  GET_BOTH_AXISPOS, GET_SCORE, GET_BALLPOS }

  ID eines UDP-Pakets.

Variablen

- const unsigned short CLIENT_PORT = 60000

  Port des Clients.
- const unsigned short SERVER_PORT = 50000

  Port des Servers.
- const unsigned int RECV_BUFFER_SIZE = 256

  Größe des Empfangsspeichers von UDPsocket.
- const float FIELD_X_MAX = 1200

  Länge des Spielfeldes in mm.
- const float FIELD_Y_MAX = 690

  Tiefe des Spielfeldes in mm.
- const int TRAVERSE_DISTANCE_1 = 241

  Verfahrweg des Torwards.
- const int TRAVERSE_DISTANCE_2 = 377

  Verfahrweg des Abwehr.
- const int TRAVERSE_DISTANCE_3 = 233

  Verfahrweg des Mittelfeld.
- const int TRAVERSE_DISTANCE_4 = 120

  Verfahrweg des Angriff.
- const float MAX_VELOCITY_TRANS = 3500.f

  Maximalgeschwindigkeit der Translation.
- const float MAX_VELOCITY_ROT = 4285.f

  Maximalgeschwindigkeit der Rotation.

7.1.1 Ausführliche Beschreibung

Enthält die defines der verschiedenen Strukturen.

Autor

Scharel Clemens

Datum

11.12.2012

Version

0.1 Testphase

Definiert in Datei defines.h.

7.1.2 Makro-Dokumentation

7.1.2.1 #define SERVER_HOST "127.0.0.1"

Host des Servers.

Definiert in Zeile 21 der Datei defines.h.

7.1.3 Dokumentation der benutzerdefinierten Typen

7.1.3.1 typedef struct ballCoordinate_t ballCoordinate_t

Struktur der Ballposition.

7.1.3.2 typedef struct ballPacket ballPacket

Paket mit der Ballposition.

Beinhaltet die Paket-ID und eine Struktur mit der Ballposition

7.1.3.3 typedef struct bool_1 bool_1

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein bool-Wert.

7.1.3.4 typedef struct bool_2 bool_2

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und zwei bool-Werte.

7.1.3.5 typedef struct float_1 float_1

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein float-Wert.

7.1.3.6 typedef struct float_2_2_4 float_2_2_4

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein dreidimensionales Array aus zwei mal zwei mal vier float-Werten.

7.1.3.7 typedef struct float_2_4 float_2_4

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein zweidimensionales Array aus zwei mal vier float-Werten.

7.1.3.8 typedef struct float_4 float_4

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein Array aus vier float-Werten.

7.1.3.9 typedef struct long_1 long_1

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein long-Wert.

7.1.3.10 typedef struct requestPacket requestPacket

Anfrage-Paket für bestimmte Daten.

7.1.3.11 typedef struct subscribePacket subscribePacket

Abonnieren-Paket für bestimmte Daten.

7.1.3.12 typedef union ubf16_request ubf16_request

Bitfeld, bei dem jedes Bit bestimmte Daten anfordert.

7.1.3.13 typedef struct unsignedShort_2 unsignedShort_2

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein Array aus zwei unsigned short-Werten.

7.1.4 Dokumentation der Aufzählungstypen

7.1.4.1 enum Axis_Number

Spielstangen.

Wird verwendet um in einem Array auf die entsprechenden Daten zu verweisen.

Aufzählungswerte

AXIS_ONE Stellt den Torward dar (1er Stange)

AXIS_TWO Stellt die Verteidigung dar (2er Stange)

AXIS_THREE Stellt das Mittelfeld dar (5er Stange)

AXIS_FOUR Stellt den Angriff dar (3er Stange)

Definiert in Zeile 83 der Datei defines.h.

7.1.4.2 enum Data_Type

Um welchen Wert es sich handelt.

Wird verwendet um in einem Array auf die entsprechenden Daten zu verweisen.

Aufzählungswerte

VALUE Position der Spielstange

VELOCITY Geschwindigkeit der Spielstange

Definiert in Zeile 131 der Datei defines.h.

7.1.4.3 enum Data_Value

Um welchen Wert es sich handelt.

Wird verwendet um in einem Array auf die entsprechenden Daten zu verweisen.

Aufzählungswerte

POSITION Position der Spielstange

ANGLE Winkel der Spielstange

Definiert in Zeile 117 der Datei defines.h.

7.1.4.4 enum Gamer

Spieler.

Wird verwendet um in einem Array auf die entsprechenden Daten zu verweisen.

Aufzählungswerte

COMPUTER Computer

HUMAN Menschlicher Spieler

BOTH Beide Spieler

Definiert in Zeile 101 der Datei defines.h.

7.1.4.5 enum PacketID

ID eines UDP-Pakets.

Jedes UDP-Paket hat eine eindeutige ID, die beschreibt welche Daten enthalten sind.

Aufzählungswerte

ALIVE_REQUEST Anfrage für ein Alive-Paket

ALIVE_ANSWER Antwort auf ein Alive-Paket

REQUEST Anfrage für verschiedene Daten

SUBSCRIBE Abonnieren von verschiedenen Daten

SET_AXISPOS Veranlasst die SPS die Spielstangen an eine bestimmte Position zu fahren

SET_SCORE Setzt den Spielstand

GET_HUMAN_AXISPOS Enthält die Spielstangenpositionen des menschlichen Spielers

GET_COMPUTER_AXISPOS Enthält die Spielstangenpositionen des Computers

GET_BOTH_AXISPOS Enthält die Spielstangenpositionen beider Spieler

GET_SCORE Enthält den Spielstand

GET_BALLPOS Enthält die Ballposition

Definiert in Zeile 159 der Datei defines.h.

7.1.4.6 enum Safety

Um welche Sicherheitsfunktion es sich handelt.

Wird verwendet um in einem Array auf die entsprechenden Daten zu verweisen.

Aufzählungswerte

STOP Notaus

GRID Lichtgitter (Eingreifschutz)

Definiert in Zeile 145 der Datei defines.h.

7.1.5 Variablen-Dokumentation

7.1.5.1 const unsigned short CLIENT_PORT = 60000

Port des Clients.

Definiert in Zeile 16 der Datei defines.h.

7.1.5.2 const float FIELD_X_MAX = 1200

Länge des Spielfeldes in mm.

Definiert in Zeile 36 der Datei defines.h.

Wird benutzt von ServerUser::ServerUser().

7.1.5.3 const float FIELD_Y_MAX = 690

Tiefe des Spielfeldes in mm.

Definiert in Zeile 41 der Datei defines.h.

Wird benutzt von ServerUser::ServerUser().

7.1.5.4 const float MAX_VELOCITY_ROT = 4285.f

Maximalgeschwindigkeit der Rotation.

Maximalgeschwindigkeit der Rotation, Grad/sek.

Definiert in Zeile 75 der Datei defines.h.

Wird benutzt von ServerUser::ServerUser().

7.1.5.5 const float MAX_VELOCITY_TRANS = 3500.f

Maximalgeschwindigkeit der Translation.

Maximalgeschwindigkeit der Translation in mm/sek.

Definiert in Zeile 68 der Datei defines.h.

Wird benutzt von ServerUser::ServerUser().

7.1.5.6 const unsigned int RECV_BUFFER_SIZE = 256

Größe des Empfangsspeichers von UDPsocket.

Definiert in Zeile 31 der Datei defines.h.

Wird benutzt von Server::Server().

7.1.5.7 const unsigned short SERVER_PORT = 50000

Port des Servers.

Definiert in Zeile 26 der Datei defines.h.

Wird benutzt von ServerUser::ServerUser().

7.1.5.8 const int TRAVERSE_DISTANCE_1 = 241

Verfahrweg des Torwards.

Definiert in Zeile 46 der Datei defines.h.

Wird benutzt von ServerUser::ServerUser().

7.1.5.9 const int TRAVERSE_DISTANCE_2 = 377

Verfahrweg des Abwehr.

Definiert in Zeile 51 der Datei defines.h.

Wird benutzt von ServerUser::ServerUser().

7.1.5.10 const int TRAVERSE_DISTANCE_3 = 233

Verfahrweg des Mittelfeld.

Definiert in Zeile 56 der Datei defines.h.

Wird benutzt von ServerUser::ServerUser().

7.1.5.11 const int TRAVERSE_DISTANCE_4 = 120

Verfahrweg des Angriff.

Definiert in Zeile 61 der Datei defines.h.

Wird benutzt von ServerUser::ServerUser().

7.2 defines.h

```
00001
00010 #ifndef DEFINES_H
00011 #define DEFINES_H
00012
00016 const unsigned short CLIENT_PORT = 60000;
00017
00021 #define SERVER_HOST "127.0.0.1"
```

```
00022
00026 const unsigned short SERVER_PORT = 50000;
00027
00031 const unsigned int RECV_BUFFER_SIZE = 256;
00032
00036 const float FIELD_X_MAX = 1200;
00037
00041 const float FIELD_Y_MAX = 690;
00042
00046 const int TRAVERSE_DISTANCE_1 = 241; //- Verfahrweg der 1er-Stange: 24,1 cm
00047
00051 const int TRAVERSE_DISTANCE_2 = 377; //- Verfahrweg der 2er-Stange: 37,7 cm
00052
00056 const int TRAVERSE_DISTANCE_3 = 233; //- Verfahrweg der 3er-Stange: 23,3 cm
00057
00061 const int TRAVERSE_DISTANCE_4 = 120; //- Verfahrweg der 5er-Stange: 12 cm
00062
00068 const float MAX_VELOCITY_TRANS = 3500.f;
00069
00075 const float MAX_VELOCITY_ROT = 4285.f;
00076
00083 enum Axis_Number
00084 {
00086     AXIS_ONE,
00088     AXIS_TWO,
00090     AXIS_THREE,
00092     AXIS_FOUR
00093 };
00094
00101 enum Gamer
00102 {
00104     COMPUTER,
00106     HUMAN,
00108     BOTH
00109 };
00110
00117 enum Data_Value
00118 {
00120     POSITION,
00122     ANGLE
00123 };
00124
00131 enum Data_Type
00132 {
00134     VALUE,
00136     VELOCITY
00137 };
00138
00145 enum Safety
00146 {
00148     STOP,
00150     GRID
00151 };
00152
00159 enum PacketID
00160 {
00162     ALIVE_REQUEST,
00164     ALIVE_ANSWER,
00166     REQUEST,
00168     SUBSCRIBE,
00170     SET_AXISPOS,
00172     SET_SCORE,
00174     GET_HUMAN_AXISPOS,
00176     GET_COMPUTER_AXISPOS,
00178     GET_BOTH_AXISPOS,
00180     GET_SCORE,
00182     GET_BALLPOS
00183 };
00184
00188 typedef union ubf16_request
00189 {
00190     unsigned int value : 16;
00191     struct bitfeld_16
00192     {
00193         unsigned int humanAxisPos : 1;          // liefert GET_HUMAN_AXISPOS
00194         unsigned int computerAxisPos : 1;       // liefert GET_COMPUTER_AXISPOS
00195         unsigned int bothAxisPos : 1;           // liefert GET_BOTH_AXISPOS
00196         unsigned int score : 1;                 // liefert GET_SCORE
00197         unsigned int ballPos : 1;               // liefert GET_BALLPOS
00198         unsigned int reserved_05 : 1;              // reserviert
00199         unsigned int reserved_06 : 1;              // reserviert
00200         unsigned int reserved_07 : 1;              // reserviert
00201         unsigned int reserved_08 : 1;              // reserviert
00202         unsigned int reserved_09 : 1;              // reserviert
00203         unsigned int reserved_10 : 1;              // reserviert
00204         unsigned int reserved_11 : 1;              // reserviert
00205         unsigned int reserved_12 : 1;              // reserviert
```

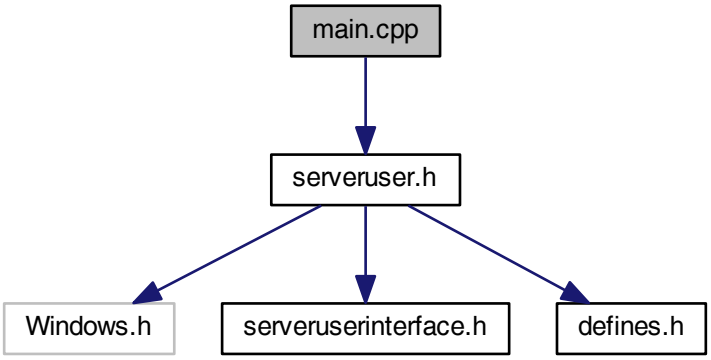
```
00206         unsigned int reserved_13 : 1;              // reserviert
00207         unsigned int reserved_14 : 1;              // reserviert
00208         unsigned int reserved_15 : 1;              // reserviert
00209     } data;
00210 } ubf16_request;
00211
00215 typedef struct requestPacket
00216 {
00218     unsigned int id : 16;
00221     ubf16_request request;
00222 } requestPacket;
00223
00227 typedef struct subscribePacket
00228 {
00230     unsigned int id : 16;
00233     ubf16_request subscribe;
00236     bool initiate;
00243     unsigned int cycle;
00244 } subscribePacket;
00245
00251 typedef struct long_1
00252 {
00253     unsigned int id : 16;
00254     long lng;
00255 } long_1;
00256
00262 typedef struct bool_1
00263 {
00264     unsigned int id : 16;
00265     bool bl;
00266 } bool_1;
00267
00273 typedef struct bool_2
00274 {
00275     unsigned int id : 16;
00276     bool bl[2];
00277 } bool_2;
00278
00284 typedef struct unsignedShort_2
00285 {
00286     unsigned int id : 16;
00287     unsigned short us[2];
00288 } unsignedShort_2;
00289
00295 typedef struct float_1
00296 {
00297     unsigned int id : 16;
00298     float flt;
00299 } float_1;
00300
00306 typedef struct float_4
00307 {
00308     unsigned int id : 16;
00309     float flt[4];
00310 } float_4;
00311
00317 typedef struct float_2_4
00318 {
00319     unsigned int id : 16;
00320     float flt[2][4];
00321 } float_2_4;
00322
00328 typedef struct float_2_2_4
00329 {
00330     unsigned int id : 16;
00331     float flt[2][2][4];
00332 } float_2_2_4;
00333
00337 typedef struct ballCoordinate_t
00338 {
00340     unsigned long ulFrameCnt;
00342     double timestamp;
00344     bool bCoordinateOk;
00346     float x;
00348     float y;
00349 } ballCoordinate_t;
00350
00356 typedef struct ballPacket
00357 {
00358     unsigned int id : 16;
00359     ballCoordinate_t ballPos;
00360 } ballPacket;
00361
00362 #endif //DEFINES_H
```

7.3 main.cpp-Dateireferenz

Enthält die Main-Funktion.

```
#include "serveruser.h"
```

Include-Abhängigkeitsdiagramm für main.cpp:



Funktionen

- int main (int argc, char **argv)
  Startet das Programm.

7.3.1 Ausführliche Beschreibung

Enthält die Main-Funktion.

Autor

Scharel Clemens

Datum

09.01.2013

Version

0.1 Testphase

Autor

Scharel Clemens

Datum

08.01.2013

Version

0.9 Testphase

Definiert in Datei main.cpp.

7.3.2 Dokumentation der Funktionen

7.3.2.1 int main (int *argc*, char ** *argv*)

Startet das Programm.

Parameter

in	*argc*	Anzahl der an die Applikation übergebenen Parameter
in	*argv*	Pointer auf den Beginn des Array mit den Parametern

Rückgabe

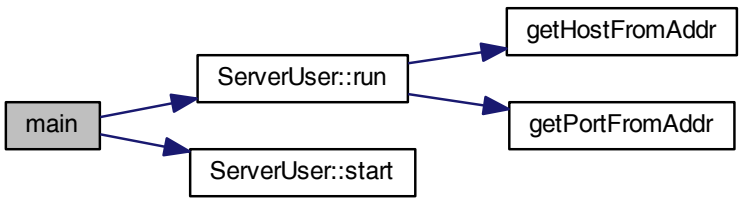
0 wenn die Applikation sachgemäß beendet wird

Legt die Klasse ServerUser an und führt sie aus.

Definiert in Zeile 146 der Datei main.cpp.

Benutzt NULL, ServerUser::run() und ServerUser::start().

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



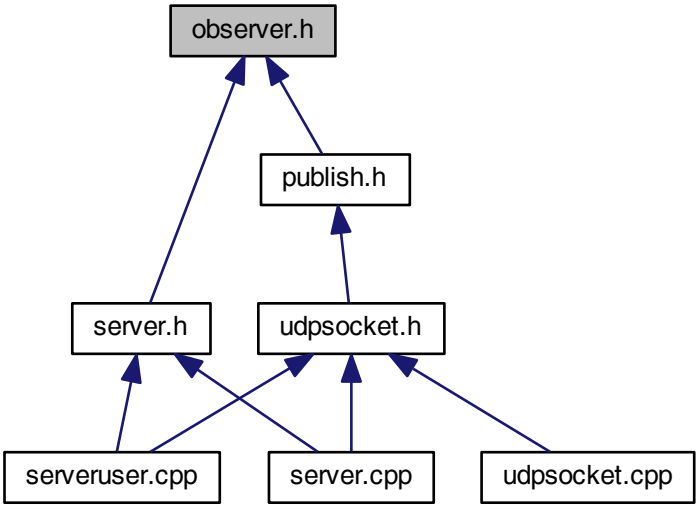
7.4 main.cpp

```
00001
00134 #include "serveruser.h"
00135
00146 int main(int argc, char** argv)
00147 {
00148     ServerUser* m_pServerUser = new ServerUser();
00149
00150     if (m_pServerUser->start())
00151         m_pServerUser->run();
00152
00153     if (m_pServerUser)
00154         delete m_pServerUser;
00155     m_pServerUser = NULL;
00156
00157     return EXIT_SUCCESS;
00158 }
```

7.5 observer.h-Dateireferenz

Enthält die Definition und Implementierung der Klasse Observer.

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



Klassen

- class Observer

  Observer Klasse für das Observer Entwurfsmodell.

7.5.1 Ausführliche Beschreibung

Enthält die Definition und Implementierung der Klasse Observer.

Autor

Scharel Clemens

Datum

08.08.2011

Version

0.1 Entwicklungsphase

Definiert in Datei observer.h.

7.6 observer.h

```
00001
00010 #ifndef OBSERVER_H
00011 #define OBSERVER_H
00012
00013 class Publish;
```

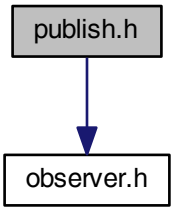
```
00014
00020 class Observer
00021 {
00022 public:
00026     virtual ~Observer() {};
00027
00032     virtual void update(Publish* const pub) = 0;
00033
00040     virtual void update(Publish* const pub, const int* const iValues, const unsigned int
      iValuesSize) = 0;
00041
00050     virtual void update(Publish* const pub, const float* const fValues, const unsigned int
      fValuesSize, const int* const iValues = NULL, const unsigned int iValuesSize = 0) = 0;
00051
00052 protected:
00056     Observer() {}
00057 };
00058
00059 #endif
```

7.7 publish.h-Dateireferenz

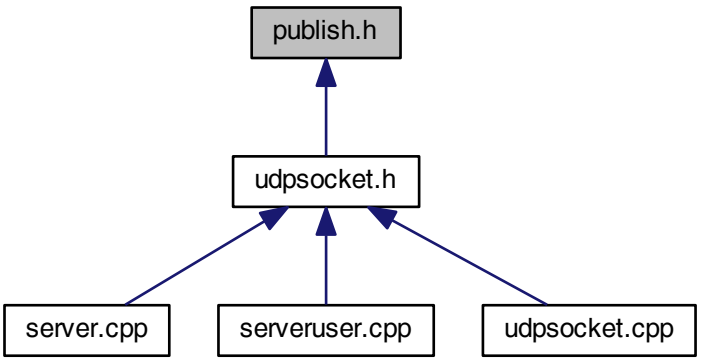
Enthält die Definition und Implementierung der Klasse Publish.

```
#include "observer.h"
```

Include-Abhängigkeitsdiagramm für publish.h:



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



Klassen

- struct ObsListKnot
- class Publish

  Publisher Klasse für das Observer Entwurfsmodell.

Makrodefinitionen

- #define NULL 0

Typdefinitionen

- typedef struct ObsListKnot ObsListKnot_t

7.7.1 Ausführliche Beschreibung

Enthält die Definition und Implementierung der Klasse Publish.

Autor

Scharel Clemens

Datum

10.01.2012

Version

1.0 Grundfunktion vollständig implementiert

Definiert in Datei publish.h.

7.7.2 Makro-Dokumentation

7.7.2.1 #define NULL 0

Definiert in Zeile 15 der Datei publish.h.

Wird benutzt von Publish::attach(), Publish::detach(), UDPsocket::getAddrFromString(), UDPsocket::getRecvDataInstance(), main(), Publish::Publish(), UDPsocket::recvData(), Server::requestAlive(), Server::Server(), ServerUser::ServerUser(), UDPsocket::start(), Server::update(), Server::∼Server(), ServerUser::∼ServerUser() und UDPsocket::∼UDPsocket().

7.7.3 Dokumentation der benutzerdefinierten Typen

7.7.3.1 typedef struct ObsListKnot ObsListKnot_t

7.8 publish.h

```
00001
00010 #ifndef PUBLISH_H
00011 #define PUBLISH_H
00012
00013 #include "observer.h"
00014
```

```
00015 #define NULL 0
00016
00024 typedef struct ObsListKnot
00025 {
00027     Observer* obs;
00029     ObsListKnot* next;
00031     ObsListKnot* prev;
00032 } ObsListKnot_t;
00033
00039 class Publish
00040 {
00041 public:
00045     virtual ~Publish() {}
00046
00053     virtual void attach(Observer* const obs)
00054     {
00055         //  Neues Listenelement erzeugen
00056         ObsListKnot_t* newObs = new ObsListKnot_t;
00057         //  Es gibt kein nächstes Element
00058         newObs->next = NULL;
00059         //  Vorheriges Element ist das bisherige letzte Element
00060         newObs->prev = lastObs;
00061         if (lastObs)
00062             /*  Wenn die Liste nicht leer war wird dieses Element das nächste
00063                 vom bisherigen letzten Element  */
00064             lastObs->next = newObs;
00065         else
00066             //  Ansonsten wird dieses Element das erste der Liste
00067             firstObs = newObs;
00068         //  Dieses Element wird das aktuelle
00069         lastObs = newObs;
00070         //  Neues Listenelement verweist auf das Observer Objekt
00071         lastObs->obs = obs;
00072     }
00073
00080     virtual bool detach(Observer* const obs)
00081     {
00082         bool retval = false;
00083         //  Alle Listenelemente durchgehen bis das betreffende gefunden wurde
00084         ObsListKnot_t* knot = firstObs;
00085         ObsListKnot_t* nextObs;
00086         while (knot)
00087         {
00088             //  Entspricht das Listenelement dem gesuchten?
00089             if (knot->obs == obs)
00090             {
00091                 //  Wenn das zu löschende Listenelement das Erste ist, ...
00092                 if (knot == firstObs)
00093                 {
00094                     //  dann wird das zweite Element zum Ersten
00095                     if (knot->next)
00096                         knot->next->prev = NULL;
00097                     firstObs = knot->next;
00098                 }
00099                 //  Wenn das zu löschende Listenelement das Letzte ist, ...
00100                 else if(knot == lastObs)
00101                 {
00102                     //  dann wird das zweitletzte Element zum Letzten
00103                     if (knot->prev)
00104                         knot->prev->next = NULL;
00105                     lastObs = knot->prev;
00106                 }
00107                 //  Ansonsten
00108                 else
00109                 {
00110                     //  Wird das vorherige Element mit dem Nächsten verbunden
00111                     knot->prev->next = knot->next;
00112                     knot->next->prev = knot->prev;
00113                 }
00114                 nextObs = knot->next;
00115                 //  Listenelement aus dem Speicher löschen
00116                 delete knot;
00117                 retval = true;
00118             }
00119             else
00120                 //  Zum nächsten Listenelement gehen
00121                 nextObs = knot->next;
00122             knot = nextObs;
00123         }
00124         return retval;
00125     }
00126
00127 protected:
00133     Publish() {firstObs = lastObs = NULL;}
00134
00139     virtual void notify(void)
00140     {
```

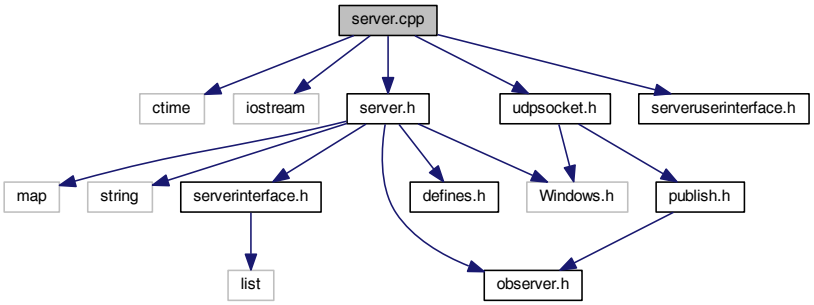
```
00141         //  Alle Listenelemente durchgehen
00142         ObsListKnot_t* knot = firstObs;
00143         while (knot)
00144         {
00145             //  Observer Element benachichtigen
00146             knot->obs->update(this);
00147             //  Zum nächsten Listenelement gehen
00148             knot = knot->next;
00149         }
00150     }
00151
00156     virtual void notify(const int* const iValues, const unsigned int iValuesSize)
00157     {
00158         //  Alle Listenelemente durchgehen
00159         ObsListKnot_t* knot = firstObs;
00160         while (knot)
00161         {
00162             //  Observer Element benachichtigen
00163             knot->obs->update(this, iValues, iValuesSize);
00164             //  Zum nächsten Listenelement gehen
00165             knot = knot->next;
00166         }
00167     }
00168
00173     virtual void notify(const float* const fValues, const unsigned int fValuesSize, const int* const
    iValues = NULL, const unsigned int iValuesSize = 0)
00174     {
00175         //  Alle Listenelemente durchgehen
00176         ObsListKnot_t* knot = firstObs;
00177         while (knot)
00178         {
00179             //  Observer Element benachichtigen
00180             if (iValues && iValuesSize)
00181                 knot->obs->update(this, fValues, fValuesSize, iValues, iValuesSize);
00182             else
00183                 knot->obs->update(this, fValues, fValuesSize);
00184             //  Zum nächsten Listenelement gehen
00185             knot = knot->next;
00186         }
00187     }
00188
00189 private:
00193     ObsListKnot_t* firstObs;
00194
00198     ObsListKnot_t* lastObs;
00199 };
00200
00201 #endif
```

7.9 server.cpp-Dateireferenz

Enthält die Implementierung der Klasse Server.

```
#include <ctime>
#include <iostream>
#include "server.h"
#include "udpsocket.h"
#include "serveruserinterface.h"
```

Include-Abhängigkeitsdiagramm für server.cpp:



7.9.1 Ausführliche Beschreibung

Enthält die Implementierung der Klasse Server.

Autor

Scharel Clemens

Datum

08.03.2012

Version

0.2 Testphase

Definiert in Datei server.cpp.

7.10 server.cpp

```
00001
00010 #include <ctime>
00011 #include <iostream>
00012 using namespace std;
00013
00014 #include "server.h"
00015 #include "udpsocket.h"
00016 #include "serveruserinterface.h"
00017
00018 Server::Server(const unsigned short localPort)
00019 {
00020     myMutexHandle = CreateMutex(NULL, FALSE, NULL);
00021     if (!myMutexHandle)
00022         cout << "Fehler beim erzeugen des Mutex: " << GetLastError() << endl;
00023
00024     mySocket = new UDPsocket(RECV_BUFFER_SIZE, localPort);
00025 }
00026
00027 Server::~Server()
00028 {
00029     if (mySocket)
00030     {
00031         mySocket->detach(this);
00032         delete mySocket;
00033     }
00034     mySocket = NULL;
00035     myUser = NULL;
00036 }
```

```
00037
00038 bool Server::start(ServerUserInterface* const user)
00039 {
00040     bool retval = false;
00041
00042     if (myRecvData && mySocket)
00043     {
00044         myUser = user;
00045         mySocket->attach(this);
00046         if (mySocket->start())
00047         {
00048             //cout << "Server gestartet!" << endl;
00049             retval = true;
00050         }
00051     }
00052     return retval;
00053 }
00054
00055 void Server::update(Publish* const pub)
00056 {
00057     if (pub == mySocket)
00058     {
00059         //cout << "Neues Paket empfangen: ";
00060         int iResult;
00061         unsigned int iID;
00062         sockaddr_in clientAddr;
00063
00064         //  Daten vom Socket abholen
00065         iResult = mySocket->getData(myRecvData, clientAddr);
00066         //  Das Paket muss mindestens 2 Bytes (16 bit) groß sein
00067         if (iResult < 2)
00068             cout << "Ungueltiges Paket empfangen!" << endl;
00069         else
00070         {
00071             iID = *myRecvData;
00072             //  Hier müssen alle Paket-ID'S abgefragt werden die empfangen werden sollen
00073             switch (iID)
00074             {
00075             case ALIVE_REQUEST:
00076                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00077                 if (iResult == sizeof(long_1))
00078                 {
00079                     //  Zeiger auf die Struktur anlegen
00080                     long_1* alive;
00081                     //  Zeifer auf die Struktur verwenden um die Daten aus dem Empfangsspeicher zu lesen
00082                     alive = (long_1*)myRecvData;
00083                     //  Die Alive-Anfrage, mit der Alive-ID beantworten
00084                     answerAlive(alive->lng, clientAddr);
00085                 }
00086                 break;
00087
00088             case ALIVE_ANSWER:
00089                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00090                 if (iResult == sizeof(long_1))
00091                 {
00092                     map<sockaddr_in, clientValues, addrLessThan>* clients;
00093                     map<sockaddr_in, clientValues, addrLessThan>::iterator it;
00094                     //  Zeiger auf die Struktur anlegen
00095                     long_1* alive;
00096                     //  Zeiger auf die Struktur verwenden um die Daten aus dem Empfangsspeicher zu lesen
00097                     alive = (long_1*)myRecvData;
00098
00099                     for (unsigned int i = 1; i <= 5; i++)
00100                     {
00101                         switch (i)
00102                         {
00103                         case 1: clients = &(myHumanAxisPosClients); break;
00104                         case 2: clients = &(myComputerAxisPosClients); break;
00105                         case 3: clients = &(myBothAxisPosClients); break;
00106                         case 4: clients = &(myScoreClients); break;
00107                         case 5: clients = &(myBallPosClients); break;
00108                         default: clients = NULL;
00109                         }
00110
00111                         DWORD waitResult = WaitForSingleObject(myMutexHandle, 10);
00112                         if (waitResult == WAIT_OBJECT_0)
00113                         {
00114                             it = clients->find(clientAddr);
00115                             if (it != clients->end())
00116                             {
00117                                 if (alive->lng == it->second.aliveID)
00118                                     it->second.aliveToggle = true;
00119                             }
00120                             if (!ReleaseMutex(myMutexHandle))
00121                                 cout << "Mutex konnte nicht wieder freigegeben werden! (requestAlive())" <<
      endl;
00122                         }
```

```
00123                   }
00124               }
00125               break;
00126
00127           //  Anfrage für Daten
00128           case REQUEST:
00129               //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00130               if (iResult == sizeof(requestPacket))
00131               {
00132                   requestPacket* request;
00133                   request = (requestPacket*)myRecvData;
00134
00135                   //  Spielstangenposition des Menschen
00136                   if (request->request.data.humanAxisPos)
00137                   {
00138                       //  Struktur des Antwortpakets anlegen
00139                       float_2_4 axisPos;
00140                       axisPos.id = GET_HUMAN_AXISPOS;
00141                       //  Daten vom Benutzer dieser Klasse per Referenz abholen
00142                       if (myUser->getHumanAxisPos(axisPos.flt[POSITION],
00143                                                   axisPos.flt[ANGLE]))
00144                           //  Antwort-Paket an den Client senden
00145                           mySocket->sendData((const char*) &axisPos, sizeof(
    float_2_4), clientAddr);
00146                   }
00147
00148                   //  Spielstangenposition des Computers
00149                   if (request->request.data.computerAxisPos)
00150                   {
00151                       //  Struktur des Antwortpakets anlegen
00152                       float_2_4 axisPos;
00153                       axisPos.id = GET_COMPUTER_AXISPOS;
00154                       //  Daten vom Benutzer dieser Klasse per Referenz abholen
00155                       if (myUser->getComputerAxisPos(axisPos.flt[POSITION],
00156                                                      axisPos.flt[ANGLE]))
00157                           //  Antwort-Paket an den Client senden
00158                           mySocket->sendData((const char*) &axisPos, sizeof(
    float_2_4), clientAddr);
00159                   }
00160
00161                   //  Spielstangenpositionen beider Spieler
00162                   if (request->request.data.bothAxisPos)
00163                   {
00164                       //  Struktur des Antwortpakets anlegen
00165                       float_2_2_4 axisPos;
00166                       axisPos.id = GET_BOTH_AXISPOS;
00167                       //  Daten vom Benutzer dieser Klasse per Referenz abholen
00168                       if (myUser->getHumanAxisPos(axisPos.flt[HUMAN][
    POSITION],
00169                                                   axisPos.flt[HUMAN][
    ANGLE]) &&
00170                           myUser->getComputerAxisPos(axisPos.flt[COMPUTER][POSITION],
00171                                                      axisPos.flt[COMPUTER][ANGLE]))
00172                           //  Antwort-Paket an den Client senden
00173                           mySocket->sendData((const char*) &axisPos, sizeof(
    float_2_2_4), clientAddr);
00174                   }
00175
00176                   //  Spielstand
00177                   if (request->request.data.score)
00178                   {
00179                       //  Struktur des Antwortpakets anlegen
00180                       unsignedShort_2 score;
00181                       score.id = GET_SCORE;
00182                       //  Daten vom Benutzer dieser Klasse per Referenz abholen
00183                       if (myUser->getScore(score.us[COMPUTER], score.
    us[HUMAN]))
00184                           //  Antwort-Paket an den Client senden
00185                           mySocket->sendData((const char*) &score, sizeof(
    unsignedShort_2), clientAddr);
00186                   }
00187
00188                   //  Ballposition
00189                   if (request->request.data.ballPos)
00190                   {
00191                       //  Struktur des Antwortpakets anlegen
00192                       ballPacket ballPos;
00193                       ballPos.id = GET_BALLPOS;
00194
00195                       //  Daten vom Benutzer dieser Klasse per Referenz abholen
00196                       if (myUser->getBallPos(ballPos.ballPos))
00197                           //  Antwort-Paket an den Client senden
00198                           mySocket->sendData((const char*) &ballPos, sizeof(
    ballPacket), clientAddr);
00199                   }
00200               }
00201               break;
```

```
00202
00203             //  Abonnieren von Daten
00204             case SUBSCRIBE:
00205                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00206                 if (iResult == sizeof(subscribePacket))
00207                 {
00208                     subscribePacket* subscribe;
00209                     subscribe = (subscribePacket*)myRecvData;
00210
00211                     //  Spielstangenposition des Menschen
00212                     if (subscribe->subscribe.data.humanAxisPos)
00213                     {
00214                         if (subscribe->initiate)
00215                         {
00216                             clientValues vals = {clock(), true};
00217                             myHumanAxisPosClients.insert(pair<sockaddr_in, clientValues>(clientAddr, vals))
      ;
00218                         }
00219                         else
00220                             myHumanAxisPosClients.erase(clientAddr);
00221                     }
00222
00223                     //  Spielstangenposition des Computers
00224                     if (subscribe->subscribe.data.computerAxisPos)
00225                     {
00226                         if (subscribe->initiate)
00227                         {
00228                             clientValues vals = {clock(), true};
00229                             myComputerAxisPosClients.insert(pair<sockaddr_in, clientValues>(clientAddr,
      vals));
00230                         }
00231                         else
00232                             myComputerAxisPosClients.erase(clientAddr);
00233                     }
00234
00235                     //  Spielstangenpositionen beider Spieler
00236                     if (subscribe->subscribe.data.bothAxisPos)
00237                     {
00238                         if (subscribe->initiate)
00239                         {
00240                             clientValues vals = {clock(), true};
00241                             myBothAxisPosClients.insert(pair<sockaddr_in, clientValues>(clientAddr, vals));
00242                         }
00243                         else
00244                             myBothAxisPosClients.erase(clientAddr);
00245                     }
00246
00247                     //  Spielstand
00248                     if (subscribe->subscribe.data.score)
00249                     {
00250                         if (subscribe->initiate)
00251                         {
00252                             clientValues vals = {clock(), true};
00253                             myScoreClients.insert(pair<sockaddr_in, clientValues>(clientAddr, vals));
00254                         }
00255                         else
00256                             myScoreClients.erase(clientAddr);
00257                     }
00258
00259                     //  Ballposition
00260                     if (subscribe->subscribe.data.ballPos)
00261                     {
00262                         if (subscribe->initiate)
00263                         {
00264                             clientValues vals = {clock(), true};
00265                             myBallPosClients.insert(pair<sockaddr_in, clientValues>(clientAddr, vals));
00266                         }
00267                         else
00268                             myBallPosClients.erase(clientAddr);
00269                     }
00270                 }
00271                 break;
00272
00273             //  Fahrbefehle für die Spielstangen des Computers
00274             case SET_AXISPOS:
00275                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00276                 if (iResult == sizeof(float_2_2_4))
00277                 {
00278                     float_2_2_4* axisPos;
00279                     axisPos = (float_2_2_4*)myRecvData;
00280
00281                     //  Daten an den Benutzer dieser Klasse übergeben
00282                     myUser->setComputerAxisPos(axisPos->flt[POSITION], axisPos->
      flt[ANGLE]);
00283                 }
00284                 break;
00285
```

```
00286             //  Spielstand soll gesetzt werden
00287             case SET_SCORE:
00288                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00289                 if (iResult == sizeof(unsignedShort_2))
00290                 {
00291                     unsignedShort_2* score;
00292                     score = (unsignedShort_2*)myRecvData;
00293 
00294                     //  Daten an den Benutzer dieser Klasse übergeben
00295                     myUser->setScore(score->us[COMPUTER], score->us[
      HUMAN]);
00296                     }
00297                 break;
00298             }
00299         }
00300     }
00301 }
00302 
00303 void Server::update(Publish* const pub, const int* const iValues, const unsigned int
      iValuesSize)
00304 {
00305     update(pub);
00306 }
00307 
00308 void Server::update(Publish* const pub, const float* const fValues, const unsigned int
       fValuesSize, const int* const iValues, const unsigned int iValuesSize)
00309 {
00310     update(pub);
00311 }
00312 
00313 /*********************************************************************************
00314 *   Hier müssen die virtuellen Memberfunktionen der Basisklasse                  *
00315 *   ServerInterface implementiert werden.                                       *
00316 *   z.B.: zum Senden von bestimmten Daten, ...                                  *
00317 *********************************************************************************/
00318 
00319 list<sockaddr_in> Server::requestAlive(const bool deleteClients)
00320 {
00321     list<sockaddr_in> retval;
00322     list<sockaddr_in> partRetval[5];
00323     list<sockaddr_in>::iterator listIt;
00324 
00325     map<sockaddr_in, clientValues, addrLessThan>::iterator it;
00326     map<sockaddr_in, clientValues, addrLessThan>* clients;
00327 
00328     DWORD waitResult;
00329 
00330     // Anfrage-Paket erstellen
00331     long_1 request;
00332     request.id = ALIVE_REQUEST;
00333 
00334     for (unsigned int i = 0; i < 5; i++)
00335     {
00336         switch (i)
00337         {
00338         case 0: clients = &(myHumanAxisPosClients); break;
00339         case 1: clients = &(myComputerAxisPosClients); break;
00340         case 2: clients = &(myBothAxisPosClients); break;
00341         case 3: clients = &(myScoreClients); break;
00342         case 4: clients = &(myBallPosClients); break;
00343         default: clients = NULL;
00344         }
00345 
00346         for (it = clients->begin(); it != clients->end(); it++)
00347         {
00348             waitResult = WaitForSingleObject(myMutexHandle, 10);
00349             if (waitResult == WAIT_OBJECT_0)
00350             {
00351                 it->second.aliveID = clock();
00352                 if (it->second.aliveToggle == false)
00353                     partRetval[i].push_back(it->first);
00354                 it->second.aliveToggle = false;
00355 
00356                 if (!ReleaseMutex(myMutexHandle))
00357                     cout << "Mutex konnte nicht wieder freigegeben werden! (requestAlive())" << endl;
00358             }
00359 
00360             if (!deleteClients)
00361             {
00362                 request.lng = it->second.aliveID;
00363                 //  Anfrage-Paket versenden
00364                 mySocket->sendData((const char*) &request, sizeof(long_1), it->first);
00365             }
00366         }
00367 
00368         if (deleteClients)
00369         {
```

```
00370             //  Clients löschen
00371             for (listIt = partRetval[i].begin(); listIt != partRetval[i].end(); ++listIt)
00372                 clients->erase(*listIt);
00373         }
00374         retval.splice(retval.end(), partRetval[i]);
00375     }
00376     return retval;
00377 }
00378
00379 bool Server::answerAlive(const long aliveID, const sockaddr_in client)
00380 {
00381     bool retval = false;
00382     int bytesSend = 0;
00383
00384     // Anfrage-Paket erstellen
00385     long_1 request;
00386     request.id = ALIVE_ANSWER;
00387     request.lng = aliveID;
00388
00389     //  Anfrage-Paket versenden
00390     bytesSend = mySocket->sendData((const char*) &request, sizeof(long_1), client);
00391
00392     //  Überprüfen ob alle Daten gesendet wurden
00393     if (bytesSend == sizeof(long_1))
00394         retval = true;
00395     return retval;
00396 }
00397
00398 bool Server::sendAxisPos(const unsigned int gamer, const float axisPos[4], const float
    axisAngle[4])
00399 {
00400     bool retval = true;
00401     int bytesSend = 0;
00402     map<sockaddr_in, clientValues, addrLessThan>::iterator it;
00403     map<sockaddr_in, clientValues, addrLessThan>* clients;
00404
00405     //  Spielstangenposition-Paket erstellen
00406     float_2_4 posPacket;
00407
00408     switch (gamer)
00409     {
00410     case HUMAN:
00411         posPacket.id = GET_HUMAN_AXISPOS;
00412         clients = &(myHumanAxisPosClients);
00413         break;
00414     case COMPUTER:
00415         posPacket.id = GET_COMPUTER_AXISPOS;
00416         clients = &(myComputerAxisPosClients);
00417         break;
00418     default:
00419         retval = false;
00420     }
00421     for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00422     {
00423         posPacket.flt[POSITION][axis] = axisPos[axis];
00424         posPacket.flt[ANGLE][axis] = axisAngle[axis];
00425     }
00426
00427     //  Spielstangenposition-Paket an alle Clients senden
00428     for (it = clients->begin(); it != clients->end(); it++)
00429     {
00430         bytesSend = mySocket->sendData((const char*) &posPacket, sizeof(
    float_2_4), it->first);
00431
00432         //  Nur wenn das Senden an alle Client erfolgreich war true zurückgeben
00433         if (bytesSend != sizeof(float_2_4) || retval == false)
00434             retval = false;
00435     }
00436     return retval;
00437 }
00438
00439 bool Server::sendAxisPos(const unsigned int gamer, const float axisPos[2][4], const
    float axisAngle[2][4])
00440 {
00441     bool retval = true;
00442     int bytesSend = 0;
00443     map<sockaddr_in, clientValues, addrLessThan>::iterator it;
00444     map<sockaddr_in, clientValues, addrLessThan>* clients;
00445
00446     //  Spielstangenposition-Paket erstellen
00447     float_2_2_4 posPacket;
00448
00449     switch (gamer)
00450     {
00451     case HUMAN:
00452         posPacket.id = GET_HUMAN_AXISPOS;
00453         clients = &(myHumanAxisPosClients);
```

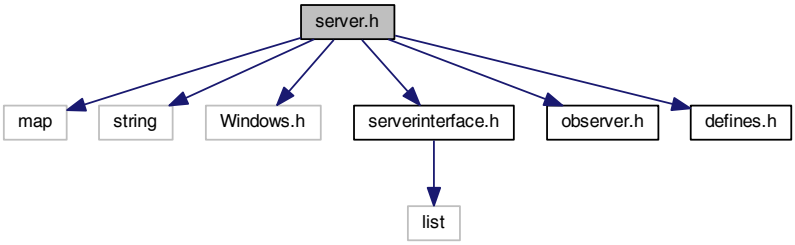
```
00454         break;
00455     case COMPUTER:
00456         posPacket.id = GET_COMPUTER_AXISPOS;
00457         clients = &(myComputerAxisPosClients);
00458         break;
00459     default:
00460         retval = false;
00461     }
00462     for (unsigned int tmpGamer = COMPUTER; tmpGamer <= HUMAN; tmpGamer++)
00463     {
00464         for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00465         {
00466             posPacket.flt[tmpGamer][POSITION][axis] = axisPos[tmpGamer][axis];
00467             posPacket.flt[tmpGamer][ANGLE][axis] = axisAngle[tmpGamer][axis];
00468         }
00469     }
00470
00471     //  Spielstangenposition-Paket an alle Clients senden
00472     for (it = clients->begin(); it != clients->end(); it++)
00473     {
00474         bytesSend = mySocket->sendData((const char*) &posPacket, sizeof(
      float_2_2_4), it->first);
00475
00476         //  Nur wenn das Senden an alle Client erfolgreich war true zurückgeben
00477         if (bytesSend != sizeof(float_2_2_4) || retval == false)
00478             retval = false;
00479     }
00480     return retval;
00481 }
00482
00483 bool Server::sendScore(const unsigned short computer, const unsigned short human)
00484 {
00485     bool retval = true;
00486     int bytesSend = 0;
00487     map<sockaddr_in, clientValues, addrLessThan>::iterator it;
00488     map<sockaddr_in, clientValues, addrLessThan>* clients = &(myScoreClients);
00489
00490     //  Spielstand-Paket erstellen
00491     unsignedShort_2 scorePacket;
00492     scorePacket.id = GET_SCORE;
00493     scorePacket.us[COMPUTER] = computer;
00494     scorePacket.us[HUMAN] = human;
00495
00496     //  Spielstand-Paket an alle Clients senden
00497     for (it = clients->begin(); it != clients->end(); it++)
00498     {
00499         bytesSend = mySocket->sendData((const char*) &scorePacket, sizeof(
      unsignedShort_2), it->first);
00500
00501         //  Nur wenn das Senden an alle Client erfolgreich war true zurückgeben
00502         if (bytesSend != sizeof(unsignedShort_2) || retval == false)
00503             retval = false;
00504     }
00505     return retval;
00506 }
00507
00508 bool Server::sendBallPos(const ballCoordinate_t ballPos)
00509 {
00510     bool retval = true;
00511     int bytesSend = 0;
00512     map<sockaddr_in, clientValues, addrLessThan>::iterator it;
00513     map<sockaddr_in, clientValues, addrLessThan>* clients = &(myBallPosClients);
00514
00515     //  Ballposition-Paket erstellen
00516     ballPacket ballPosPacket;
00517     ballPosPacket.id = GET_BALLPOS;
00518     ballPosPacket.ballPos = ballPos;
00519
00520     //  Ballposition-Paket an alle Clients senden
00521     for (it = clients->begin(); it != clients->end(); it++)
00522     {
00523         bytesSend = mySocket->sendData((const char*) &ballPosPacket, sizeof(
      ballPacket), it->first);
00524
00525         //  Nur wenn das Senden an alle Client erfolgreich war true zurückgeben
00526         if (bytesSend != sizeof(ballPacket) || retval == false)
00527             retval = false;
00528     }
00529     return retval;
00530 }
```

7.11 server.h-Dateireferenz

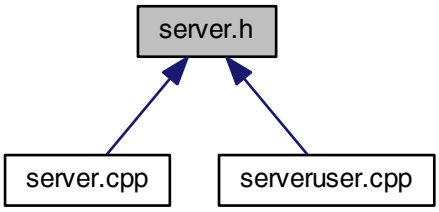
Enthält die Definition der Klasse Server.

```
#include <map>
#include <string>
#include <Windows.h>
#include "serverinterface.h"
#include "observer.h"
#include "defines.h"
```

Include-Abhängigkeitsdiagramm für server.h:



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



Klassen

- struct clientValues

  Struktur für die Daten, die mit den Clientadressen in die map eingefügt werden.
- struct addrLessThan

  Zum vergleich zweier sockaddr_in Strukturen.
- class Server

  ein UDP-Server

Typdefinitionen

- typedef struct clientValues clientValues

7.11.1 Ausführliche Beschreibung

Enthält die Definition der Klasse Server.

Autor

Scharel Clemens

Datum

08.03.2012

Version

0.2 Testphase

Definiert in Datei server.h.

7.11.2 Dokumentation der benutzerdefinierten Typen

7.11.2.1 typedef struct clientValues clientValues

7.12 server.h

```
00001
00010 #ifndef SERVER_H
00011 #define SERVER_H
00012
00013 #include <map>
00014 #include <string>
00015 #include <Windows.h>
00016
00017 #include "serverinterface.h"
00018 #include "observer.h"
00019 //#include "clientlist.h"
00020 #include "defines.h"
00021
00022 class UDPsocket;
00023 class ServerUserInterface;
00024 //struct sockaddr_in;
00025
00033 typedef struct clientValues
00034 {
00036     long aliveID;
00037
00039     bool aliveToggle;
00040 } clientValues;
00041
00047 struct addrLessThan:public binary_function<const struct sockaddr_in, const struct sockaddr_in,
      bool>
00048 {
00049     bool operator()(const struct sockaddr_in addr1, const struct sockaddr_in addr2) const
00050     {
00051         bool retVal = true;
00052
00053         string addrStr1 = inet_ntoa(addr1.sin_addr);
00054         string addrStr2 = inet_ntoa(addr2.sin_addr);
00055
00056         if(addrStr1 > addrStr2)
00057         retVal = false;
00058         else if(addrStr1 == addrStr2)
00059         retVal = (addr1.sin_port < addr2.sin_port);
00060
00061         return retVal;
00062     }
00063 };
00064
00071 class Server : public Observer, public ServerInterface
00072 {
00073 public:
00077     Server(const unsigned short localPort);
00078
00082     ~Server();
```

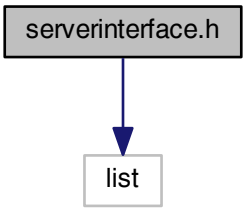
```
00083
00087     bool start(ServerUserInterface* const user);
00088
00096     void update(Publish* const pub);
00097
00106     virtual void update(Publish* const pub, const int* const iValues, const unsigned int
      iValuesSize);
00107
00118     virtual void update(Publish* const pub, const float* const fValues, const unsigned int
      fValuesSize, const int* const iValues = NULL, const unsigned int iValuesSize = 0);
00119
00120     /*************************************************************************
00121     *   Hier müssen die virtuellen Memberfunktionen der Basisklasse          *
00122     *   ServerInterface definiert werden.                                    *
00123     *   z.B.: zum Senden von bestimmten Daten oder zum Antworten auf Anfragen  *
00124     *************************************************************************/
00125
00135     list<sockaddr_in> requestAlive(const bool deleteClients = false);
00136
00144     bool sendAxisPos(const unsigned int gamer, const float axisPos[4], const float axisAngle[4])
      ;
00145
00153     bool sendAxisPos(const unsigned int gamer, const float axisPos[2][4], const float axisAngle[
      2][4]);
00154
00155     /*
00156      *  @brief              sendet den Spielstand
00157      *  @param[in] computer der Spielstand des Computers
00158      *  @param[in] human    der Spielstand des menschlichen Spielers
00159      *  @return             true wenn er Spielstand erfolgreich an alle Clients gesendet wurde
00160      *
00161      *  Sendet den Spielstand an alle abonnierten Clients.
00162      */
00163     bool sendScore(const unsigned short computer, const unsigned short human);
00164
00165     /*
00166      *  @brief              sendet die Ballposition
00167      *  @param[in] ballPos  die neue Ballposition
00168      *  @return             true wenn die Ballposition erfolgreich an alle Clients gesendet wurde
00169      *
00170      *  Sendet die Ballposition an alle abonnierten Clients.
00171      */
00172     bool sendBallPos(const ballCoordinate_t ballPos);
00173
00174 private:
00178     UDPsocket* mySocket;
00179
00183     ServerUserInterface* myUser;
00184
00188     const char* myRecvData;
00189
00193     map<sockaddr_in, clientValues, addrLessThan> myHumanAxisPosClients;
00194
00198     map<sockaddr_in, clientValues, addrLessThan> myComputerAxisPosClients;
00199
00203     map<sockaddr_in, clientValues, addrLessThan> myBothAxisPosClients;
00204
00208     map<sockaddr_in, clientValues, addrLessThan> myScoreClients;
00209
00213     map<sockaddr_in, clientValues, addrLessThan> myBallPosClients;
00214
00218     HANDLE myMutexHandle;
00219
00226     bool answerAlive(const long aliveID, const sockaddr_in client);
00227 };
00228
00229 #endif // SERVER_H
```

7.13 serverinterface.h-Dateireferenz

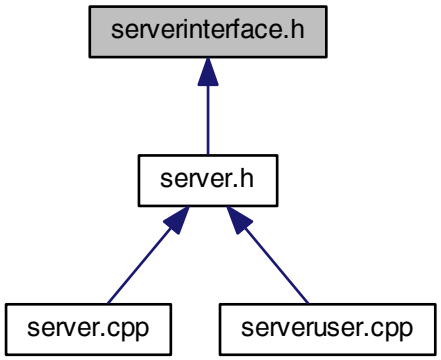
Enthält das Interface zur Kommunikation mit dem UDP-Server.

```
#include <list>
```

Include-Abhängigkeitsdiagramm für serverinterface.h:



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



Klassen

- interface ServerInterface

  Interface zur Kommunikation mit dem UDP-Server.

7.13.1 Ausführliche Beschreibung

Enthält das Interface zur Kommunikation mit dem UDP-Server.

Autor

Scharel Clemens

Datum

07.02.2012

Version

0.1 Testphase

Definiert in Datei serverinterface.h.

7.14 serverinterface.h

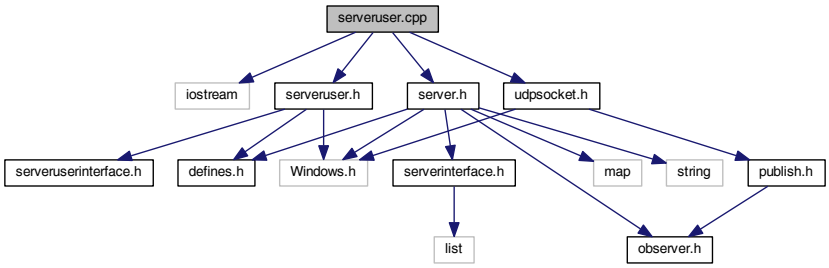
```
00001
00010 #ifndef SERVERINTERFACE_H
00011 #define SERVERINTERFACE_H
00012
00013 #include <list>
00014
00015 class ServerUserInterface;
00016 struct ballCoordinate_t;
00017 enum Gamer;
00018
00023 class ServerInterface
00024 {
00025 public:
00029     virtual ~ServerInterface(void) {};
00030
00036     virtual bool start(ServerUserInterface* const userInterface) = 0;
00037
00038     /*************************************************************************
00039     *   Hier müssen die rein virtuellen Memberfunktionen definiert werden,     *
00040     *   welche vom Server implementiert werden müssen.                         *
00041     *   Diese Funktionen werden vom Benutzer des Servers aufgerufen.           *
00042     *   z.B.: zum Senden von bestimmten Daten oder zum Antworten auf Anfragen  *
00043     *************************************************************************/
00044
00054     virtual list<sockaddr_in> requestAlive(const bool deleteClients = false) = 0;
00055
00063     virtual bool sendAxisPos(const unsigned int gamer, const float axisPos[4], const float
      axisAngle[4]) = 0;
00064
00072     virtual bool sendAxisPos(const unsigned int gamer, const float axisPos[2][4], const float
      axisAngle[2][4]) = 0;
00073
00074     /*
00075      *  @brief              sendet den Spielstand
00076      *  @param[in] computer der Spielstand des Computers
00077      *  @param[in] human    der Spielstand des menschlichen Spielers
00078      *  @return             true wenn er Spielstand erfolgreich an alle Clients gesendet wurde
00079      *
00080      *  Sendet den Spielstand an alle abonnierten Clients.
00081      */
00082     virtual bool sendScore(const unsigned short computer, const unsigned short human) = 0;
00083
00084     /*
00085      *  @brief              sendet die Ballposition
00086      *  @param[in] ballPos  die neue Ballposition
00087      *  @return             true wenn die Ballposition erfolgreich an alle Clients gesendet wurde
00088      *
00089      *  Sendet die Ballposition an alle abonnierten Clients.
00090      */
00091     virtual bool sendBallPos(const ballCoordinate_t ballPos) = 0;
00092
00093 private:
00100     virtual bool answerAlive(const long aliveID, const sockaddr_in client) = 0;
00101 };
00102
00103 #endif  // SERVERINTERFACE_H
```

7.15 serveruser.cpp-Dateireferenz

Enthält die Implementierung der Klasse ServerUser.

```
#include <iostream>
#include "serveruser.h"
#include "server.h"
#include "udpsocket.h"
```

Include-Abhängigkeitsdiagramm für serveruser.cpp:



7.15.1 Ausführliche Beschreibung

Enthält die Implementierung der Klasse ServerUser.

Autor

Scharel Clemens

Datum

07.02.2012

Version

0.1 Testphase

Definiert in Datei serveruser.cpp.

7.16 serveruser.cpp

```
00001
00010 #include <iostream>
00011 using namespace std;
00012
00013 #include "serveruser.h"
00014 #include "server.h"
00015 #include "udpsocket.h"
00016
00017 ServerUser::ServerUser()
00018 {
00019     //  Startwerte festlegen
00020     myComputerAxisPos[VALUE][AXIS_ONE] = myHumanAxisPos[AXIS_ONE] =
      TRAVERSE_DISTANCE_1 / 2;
00021     myComputerAxisPos[VALUE][AXIS_TWO] = myHumanAxisPos[AXIS_TWO] =
      TRAVERSE_DISTANCE_2 / 2;
00022     myComputerAxisPos[VALUE][AXIS_THREE] = myHumanAxisPos[
      AXIS_THREE] = TRAVERSE_DISTANCE_3 / 2;
00023     myComputerAxisPos[VALUE][AXIS_FOUR] = myHumanAxisPos[AXIS_FOUR] =
      TRAVERSE_DISTANCE_4 / 2;
00024
00025     for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00026     {
00027         myComputerAxisPos[VELOCITY][axis] = MAX_VELOCITY_TRANS;
00028         myComputerAxisAngle[VELOCITY][axis] = MAX_VELOCITY_ROT;
00029         myHumanAxisAngle[axis] = 0.f;
00030         myComputerAxisAngle[VALUE][axis] = 0.f;
00031     }
00032     myScore[HUMAN] = myScore[COMPUTER] = 0;
00033
00034     myBallPos.bCoordinateOk = false;
```

```
00035     myBallPos.timestamp = 0;
00036     myBallPos.ulFrameCnt = 0;
00037     myBallPos.x = FIELD_X_MAX / 2.f;
00038     myBallPos.y = FIELD_Y_MAX / 2.f;
00039 
00040     myMutexHandle = CreateMutex(NULL, FALSE, NULL);
00041     if (!myMutexHandle)
00042         cout << "Fehler beim erzeugen des Mutex: " << GetLastError() << endl;
00043     myServer = new Server(SERVER_PORT);
00044 }
00045 
00046 ServerUser::~ServerUser()
00047 {
00048     if (myServer)
00049         delete myServer;
00050     myServer = NULL;
00051     CloseHandle(myMutexHandle);
00052 }
00053 
00054 bool ServerUser::start()
00055 {
00056     bool retval = false;
00057     if (myServer->start(this))
00058         retval = true;
00059     return retval;
00060 }
00061 
00062 bool ServerUser::run()
00063 {
00064     bool retval = false;
00065     bool end = false;
00066     char input;
00067 
00068     do
00069     {
00070         bool sendRet = false;
00071         list<sockaddr_in> clients;
00072         cout
00073             << "[A] Daten ausgeben    |    [S] Daten senden" << endl
00074             << "[E] Daten eingeben    |    [K] Alive anfragen" << endl
00075             << "                      |" << endl
00076             << "[Q] Beenden           |" << endl
00077             << "------------------------------------------------------------" << endl
00078             << "Option waehlen: ";
00079         cin >> input;
00080         cout << endl << endl;
00081         switch (input)
00082         {
00083         case 'a':
00084         case 'A':
00085             cout << "Spielstangenpositionen:" << endl;
00086             cout << "Mensch:" << endl;
00087             for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00088                 cout << axis << ": " << myHumanAxisPos[axis] << "mm \t" << myHumanAxisAngle[axis] << char(2
    48) << endl;
00089             cout << "Computer:" << endl;
00090             for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00091                 cout
00092                 << axis << ": "
00093                 << myComputerAxisPos[VALUE][axis] << "mm (" << myComputerAxisPos[
    VELOCITY][axis] << "mm/s) \t"
00094                 << myComputerAxisAngle[VALUE][axis] << char(248) << " (" << myComputerAxisAngle[
    VELOCITY][axis] << char(248) << "/s)" << endl;
00095 
00096             cout << endl;
00097 
00098             cout << "Ballposition:" << endl;
00099             cout
00100                 << "X: " << myBallPos.x << endl
00101                 << "Y: " << myBallPos.y << endl;
00102 
00103             cout << endl;
00104 
00105             cout << "Spielstand:" << endl;
00106             for (unsigned int gamer = COMPUTER; gamer <= HUMAN; gamer++)
00107             {
00108                 if (gamer == HUMAN)
00109                     cout << "Mensch:   ";
00110                 if (gamer == COMPUTER)
00111                     cout << "Computer: ";
00112                 cout << myScore[gamer] << endl;
00113             }
00114             break;
00115 
00116         case 'e':
00117         case 'E':
00118             cout
```

```
00119                 << "[P] Spielstangenpositionen" << endl
00120                 << "[B] Ballposition" << endl
00121                 << "[S] Spielstand" << endl
00122                 << "------------------------------------------------------------" << endl
00123                 << "Option waehlen: ";
00124             cin >> input;
00125             cout << endl << endl;
00126             switch (input)
00127             {
00128             case 'p':
00129             case 'P':
00130                 int gamerToSet;
00131                 int axisToSet;
00132                 float newAxisPos[2][2];
00133 
00134                 cout
00135                     << "[H] Mensch" << endl
00136                     << "[C] Computer" << endl
00137                     << "------------------------------------------------------------" << endl
00138                     << "Option waehlen: ";
00139                 cin >> input;
00140                 cout << endl << endl;
00141                 switch (input)
00142                 {
00143                 case 'h':
00144                 case 'H':
00145                     gamerToSet = HUMAN;
00146                     break;
00147                 case 'c':
00148                 case 'C':
00149                     gamerToSet = COMPUTER;
00150                     break;
00151                 default:
00152                     gamerToSet = -1;
00153                     break;
00154                 }
00155 
00156                 if (gamerToSet >= 0)
00157                 {
00158                     cout
00159                         << "[1] Torward    |   [3] Mittelfeld" << endl
00160                         << "[2] Abwehr     |   [4] Sturm" << endl
00161                         << "------------------------------------------------------------" << endl
00162                         << "Option waehlen: ";
00163                     cin >> input;
00164                     cout << endl << endl;
00165                     switch (input)
00166                     {
00167                     case '1':
00168                         axisToSet = AXIS_ONE;
00169                         break;
00170                     case '2':
00171                         axisToSet = AXIS_TWO;
00172                         break;
00173                     case '3':
00174                         axisToSet = AXIS_THREE;
00175                         break;
00176                     case '4':
00177                         axisToSet = AXIS_FOUR;
00178                         break;
00179                     default:
00180                         axisToSet = -1;
00181                         break;
00182                     }
00183 
00184                     if (axisToSet >= 0)
00185                     {
00186                         cout << "Linearposition eingeben: ";
00187                         cin >> newAxisPos[POSITION][VALUE];
00188                         if (gamerToSet == COMPUTER)
00189                         {
00190                             cout << "Lineargeschwindigkeit eingeben: ";
00191                             cin >> newAxisPos[POSITION][VELOCITY];
00192                         }
00193                         cout << "Rotationswinkel eingeben: ";
00194                         cin >> newAxisPos[ANGLE][VALUE];
00195                         if (gamerToSet == COMPUTER)
00196                         {
00197                             cout << "Rotationsgeschwindigkeit eingeben: ";
00198                             cin >> newAxisPos[ANGLE][VELOCITY];
00199                         }
00200 
00201                         if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00202                         {
00203                             switch (gamerToSet)
00204                             {
00205                             case COMPUTER:
```

```
00206                                   myComputerAxisPos[VALUE][axisToSet] = newAxisPos[
      POSITION][VALUE];
00207                                   myComputerAxisPos[VELOCITY][axisToSet] = newAxisPos[
      POSITION][VELOCITY];
00208                                   myComputerAxisAngle[VALUE][axisToSet] = newAxisPos[
      ANGLE][VALUE];
00209                                   myComputerAxisAngle[VELOCITY][axisToSet] = newAxisPos[
      ANGLE][VELOCITY];
00210                                   break;
00211                               case HUMAN:
00212                                   myHumanAxisPos[axisToSet] = newAxisPos[POSITION][
      VALUE];
00213                                   myHumanAxisAngle[axisToSet] = newAxisPos[ANGLE][
      VALUE];
00214                                   break;
00215                               }
00216                               ReleaseMutex(myMutexHandle);
00217                           }
00218                       }
00219                   }
00220                   break;
00221
00222               case 'b':
00223               case 'B':
00224                   ballCoordinate_t newBallPos;
00225
00226                   cout << "Frame: ";
00227                   cin >> newBallPos.ulFrameCnt;
00228                   cout << "Zeitstempel: ";
00229                   cin >> newBallPos.timestamp;
00230                   cout << "Koordinate gueltig? [J] oder [N]: ";
00231                   cin >> input;
00232                   switch (input)
00233                   {
00234                   case 'j':
00235                   case 'J':
00236                       newBallPos.bCoordinateOk = true;
00237                       break;
00238                   default:
00239                       newBallPos.bCoordinateOk = false;
00240                       break;
00241                   }
00242                   cout << "Ballposition X-Koordinate: ";
00243                   cin >> newBallPos.x;
00244                   cout << "Ballposition Y-Koordinate: ";
00245                   cin >> newBallPos.y;
00246                   if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00247                   {
00248                       myBallPos = newBallPos;
00249                       ReleaseMutex(myMutexHandle);
00250                   }
00251                   break;
00252
00253               case 's':
00254               case 'S':
00255                   int scoreToSet;
00256                   unsigned short score;
00257
00258                   cout
00259                       << "[H] Mensch" << endl
00260                       << "[C] Computer" << endl
00261                       << "-----------------------------------------------------------------" << endl
00262                       << "Option waehlen: ";
00263                   cin >> input;
00264                   cout << endl << endl;
00265                   switch (input)
00266                   {
00267                   case 'h':
00268                   case 'H':
00269                       scoreToSet = HUMAN;
00270                       break;
00271                   case 'c':
00272                   case 'C':
00273                       scoreToSet = COMPUTER;
00274                       break;
00275                   default:
00276                       scoreToSet = -1;
00277                       break;
00278                   }
00279
00280                   if (scoreToSet >= 0)
00281                   {
00282                       cout << "Spielstand eingeben: ";
00283                       cin >> score;
00284
00285                       if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00286                       {
```

```
00287                         myScore[scoreToSet] = score;
00288                         ReleaseMutex(myMutexHandle);
00289                     }
00290                 }
00291                 break;
00292             }
00293             break;
00294 
00295         case 's':
00296         case 'S':
00297             cout
00298                 << "[P] Spielstangenpositionen" << endl
00299                 << "[B] Ballposition" << endl
00300                 << "[S] Spielstand" << endl
00301                 << "------------------------------------------------------------" << endl
00302                 << "Option waehlen: ";
00303                 cin >> input;
00304                 cout << endl << endl;
00305             switch (input)
00306             {
00307             case 'p':
00308             case 'P':
00309                 int gamerToSet;
00310 
00311                 cout
00312                     << "[H] Mensch" << endl
00313                     << "[C] Computer" << endl
00314                     << "[B] Mensch & Computer" << endl
00315                     << "------------------------------------------------------------" << endl
00316                     << "Option waehlen: ";
00317                 cin >> input;
00318                 cout << endl << endl;
00319                 switch (input)
00320                 {
00321                 case 'h':
00322                 case 'H':
00323                     if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00324                     {
00325                         sendRet = myServer->sendAxisPos(HUMAN, myHumanAxisPos, myHumanAxisAngle);
00326                         ReleaseMutex(myMutexHandle);
00327                     }
00328                     break;
00329                 case 'c':
00330                 case 'C':
00331                     if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00332                     {
00333                         sendRet = myServer->sendAxisPos(COMPUTER, myComputerAxisPos[
    VALUE], myComputerAxisAngle[VALUE]);
00334                         ReleaseMutex(myMutexHandle);
00335                     }
00336                     break;
00337                 case 'b':
00338                 case 'B':
00339                     float tmpAxisPos[2][4], tmpAxisAngle[2][4];
00340 
00341                     if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00342                     {
00343                         for (unsigned int axis = AXIS_ONE; axis <=
    AXIS_FOUR; axis++)
00344                         {
00345                             tmpAxisPos[COMPUTER][axis] = myComputerAxisPos[
    VALUE][axis];
00346                             tmpAxisAngle[COMPUTER][axis] = myComputerAxisAngle[
    VALUE][axis];
00347                             tmpAxisPos[HUMAN][axis] = myHumanAxisPos[axis];
00348                             tmpAxisAngle[HUMAN][axis] = myHumanAxisAngle[axis];
00349                         }
00350                         ReleaseMutex(myMutexHandle);
00351                     }
00352                     sendRet = myServer->sendAxisPos(BOTH, tmpAxisPos, tmpAxisAngle);
00353                     break;
00354                 default:
00355                     gamerToSet = -1;
00356                     break;
00357                 }
00358                 break;
00359 
00360             case 'b':
00361             case 'B':
00362                 if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00363                 {
00364                     sendRet = myServer->sendBallPos(myBallPos);
00365                     ReleaseMutex(myMutexHandle);
00366                 }
00367                 break;
00368 
00369             case 's':
```

```
00370             case 'S':
00371                 if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00372                 {
00373                     sendRet = myServer->sendScore(myScore[COMPUTER], myScore[
    HUMAN]);
00374                     ReleaseMutex(myMutexHandle);
00375                 }
00376                 break;
00377             if (!sendRet)
00378                 cout << "Paket wurde nicht gesendet!" << endl;
00379             }
00380             break;
00381
00382         case 'k':
00383         case 'K':
00384             cout << "Sollen nicht-antwortende Clients geloescht werden? [J] oder [N]: ";
00385             cin >> input;
00386             switch (input)
00387             {
00388             case 'j':
00389             case 'J':
00390                 clients = myServer->requestAlive(true);
00391                 break;
00392             default:
00393                 clients = myServer->requestAlive();
00394                 break;
00395             }
00396             if (clients.empty())
00397                 cout << "Alle Clients haben auf die letzte Anfrage geantwortet." << endl;
00398             else
00399             {
00400                 cout << "Folgende Clients haben nicht auf die letzte Anfrage geantwortet:" << endl;
00401                 while (!clients.empty())
00402                 {
00403                     string hostAddr;
00404                     unsigned short port;
00405                     if (getHostFromAddr(clients.front(), hostAddr) &&
00406                         getPortFromAddr(clients.front(), port))
00407                         cout << hostAddr << "::" << port << endl;
00408                     else
00409                         cout << "Unbekannte Serveradresse!" << endl;
00410                     clients.pop_front();
00411                 }
00412             }
00413             break;
00414
00415         case 'q':
00416         case 'Q':
00417             end = true;
00418             break;
00419         default:
00420             cout << "Ungueltiges Zeichen!" << endl;
00421         }
00422
00423         cout << endl << endl << endl;
00424     } while(!end);
00425
00426     retval = true;
00427     return retval;
00428 }
00429
00430 bool ServerUser::setComputerAxisPos(const float axisPos[2][4], const float
    axisAngle[2][4])
00431 {
00432     bool retval = false;
00433     DWORD waitResult;
00434
00435     //  Sicheren Datenzugriff auf myAxisPos und myAxisAngel da diese Funktion von einem anderen Thread
     aufgerufen wird
00436     waitResult = WaitForSingleObject(myMutexHandle, 10);
00437     if (waitResult == WAIT_OBJECT_0)
00438     {
00439         //  Spielstangenpositionen übernehmen
00440         for (unsigned int value = VALUE; value <= VELOCITY; value ++)
00441         {
00442             for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00443             {
00444                 myComputerAxisPos[value][axis] = axisPos[value][axis];
00445                 myComputerAxisAngle[value][axis] = axisAngle[value][axis];
00446             }
00447         }
00448
00449         //  Mutex wieder freigeben
00450         if (!ReleaseMutex(myMutexHandle))
00451         {
00452             cout << "Mutex konnte nicht wieder freigegeben werden! (setComputerAxisPos())" << endl;
00453         }
```

```
00454         retval = true;
00455     }
00456     else
00457         cout << "Fehler beim Warten auf Mutex! (setComputerAxisPos())" << endl;
00458     return retval;
00459 }
00460
00461 bool ServerUser::setScore(const unsigned short computer, const unsigned short human)
00462 {
00463     bool retval = false;
00464     DWORD waitResult;
00465
00466     //  Sicheren Datenzugriff auf myAxisPos und myAxisAngel da diese Funktion von einem anderen Thread
       aufgerufen wird
00467     waitResult = WaitForSingleObject(myMutexHandle, 10);
00468     if (waitResult == WAIT_OBJECT_0)
00469     {
00470         //  Spielstangenpositionen übernehmen
00471         for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00472         {
00473             myScore[HUMAN] = human;
00474             myScore[COMPUTER] = computer;
00475         }
00476
00477         //  Mutex wieder freigeben
00478         if (!ReleaseMutex(myMutexHandle))
00479         {
00480             cout << "Mutex konnte nicht wieder freigegeben werden! (setScore())" << endl;
00481         }
00482         retval = true;
00483     }
00484     else
00485         cout << "Fehler beim Warten auf Mutex! (setScore())" << endl;
00486     return retval;
00487 }
00488
00489 bool ServerUser::getHumanAxisPos(float (&axisPos)[4], float (&axisAngle)[4])
00490 {
00491     bool retval = false;
00492     DWORD waitResult;
00493
00494     //  Sicheren Datenzugriff auf myAxisPos und myAxisAngel da diese Funktion von einem anderen Thread
       aufgerufen wird
00495     waitResult = WaitForSingleObject(myMutexHandle, 10);
00496     if (waitResult == WAIT_OBJECT_0)
00497     {
00498         //  Spielstangenpositionen übergeben
00499         for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00500         {
00501             axisPos[axis] = myHumanAxisPos[axis];
00502             axisAngle[axis] = myHumanAxisAngle[axis];
00503         }
00504
00505         //  Mutex wieder freigeben
00506         if (!ReleaseMutex(myMutexHandle))
00507         {
00508             cout << "Mutex konnte nicht wieder freigegeben werden! (getHumanAxisPos())" << endl;
00509         }
00510         retval = true;
00511     }
00512     else
00513         cout << "Fehler beim Warten auf Mutex! (getHumanAxisPos())" << endl;
00514     return retval;
00515 }
00516
00517 bool ServerUser::getComputerAxisPos(float (&axisPos)[4], float (&axisAngle)[4
      ])
00518 {
00519     bool retval = false;
00520     DWORD waitResult;
00521
00522     //  Sicheren Datenzugriff auf myAxisPos und myAxisAngel da diese Funktion von einem anderen Thread
       aufgerufen wird
00523     waitResult = WaitForSingleObject(myMutexHandle, 10);
00524     if (waitResult == WAIT_OBJECT_0)
00525     {
00526         //  Spielstangenpositionen übergeben
00527         for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00528         {
00529             axisPos[axis] = myComputerAxisPos[VALUE][axis];
00530             axisAngle[axis] = myComputerAxisAngle[VALUE][axis];
00531         }
00532
00533         //  Mutex wieder freigeben
00534         if (!ReleaseMutex(myMutexHandle))
00535         {
00536             cout << "Mutex konnte nicht wieder freigegeben werden! (getComputerAxisPos())" << endl;
```

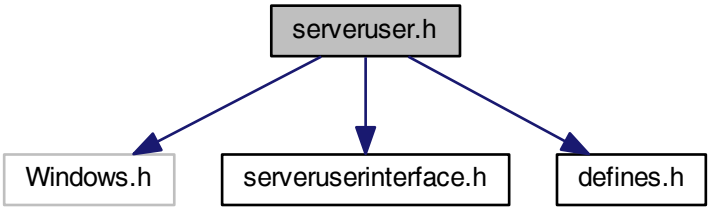
```
00537         }
00538         retval = true;
00539     }
00540     else
00541         cout << "Fehler beim Warten auf Mutex! (getComputerAxisPos())" << endl;
00542     return retval;
00543 }
00544
00545 bool ServerUser::getScore(unsigned short &human, unsigned short &computer)
00546 {
00547     bool retval = false;
00548     DWORD waitResult;
00549
00550     //  Sicheren Datenzugriff auf myScore da diese Funktion von einem anderen Thread aufgerufen wird
00551     waitResult = WaitForSingleObject(myMutexHandle, 10);
00552     if (waitResult == WAIT_OBJECT_0)
00553     {
00554         //  Spielstand übergeben
00555         human = myScore[HUMAN];
00556         computer = myScore[COMPUTER];
00557
00558         //  Mutex wieder freigeben
00559         if (!ReleaseMutex(myMutexHandle))
00560         {
00561             cout << "Mutex konnte nicht wieder freigegeben werden! (getScore())" << endl;
00562         }
00563         retval = true;
00564     }
00565     else
00566         cout << "Fehler beim Warten auf Mutex! (getScore())" << endl;
00567     return retval;
00568 }
00569
00570 bool ServerUser::getBallPos(ballCoordinate_t &ballPos)
00571 {
00572     bool retval = false;
00573     DWORD waitResult;
00574
00575     //  Sicheren Datenzugriff auf myBallPos da diese Funktion von einem anderen Thread aufgerufen wird
00576     waitResult = WaitForSingleObject(myMutexHandle, 10);
00577     if (waitResult == WAIT_OBJECT_0)
00578     {
00579         //  Ballposition
00580         ballPos = myBallPos;
00581
00582         //  Mutex wieder freigeben
00583         if (!ReleaseMutex(myMutexHandle))
00584         {
00585             cout << "Mutex konnte nicht wieder freigegeben werden! (getBallPos())" << endl;
00586         }
00587         retval = true;
00588     }
00589     else
00590         cout << "Fehler beim Warten auf Mutex! (getBallPos())" << endl;
00591     return retval;
00592 }
```

7.17 serveruser.h-Dateireferenz

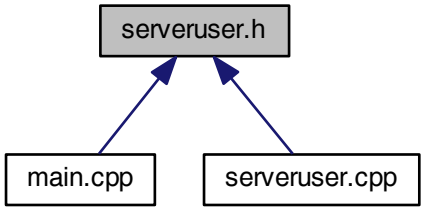
Enthält die Definition der Klasse ServerUser.

```
#include <Windows.h>
#include "serveruserinterface.h"
#include "defines.h"
```

Include-Abhängigkeitsdiagramm für serveruser.h:



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



Klassen

- class ServerUser

  Eine Klasse, die den UDP-Server anlegt und benutzt.

7.17.1 Ausführliche Beschreibung

Enthält die Definition der Klasse ServerUser.

Autor

Scharel Clemens

Datum

13.02.2012

Version

0.1 Testphase

Definiert in Datei serveruser.h.

7.18 serveruser.h

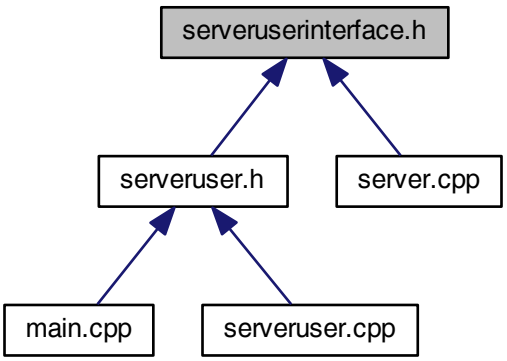
```
00001
00010 #ifndef SERVERUSER_H
00011 #define SERVERUSER_H
00012
00013 #include <Windows.h>
00014
00015 #include "serveruserinterface.h"
00016 #include "defines.h"
00017
00018 class ServerInterface;
00019
00026 class ServerUser : public ServerUserInterface
00027 {
00028 public:
00034     ServerUser();
00035
00041     ~ServerUser();
00042
00047     bool start();
00048
00054     bool run();
00055
00056     /*****************************************************************************
00057     *   Hier müssen die virtuellen Memberfunktionen der Basisklasse            *
00058     *   ServerUserInterface definiert werden.                                  *
00059     *   z.B.: zum Übergeben von empfangenen Daten, ...                         *
00060     *****************************************************************************/
00061
00062     /*
00063      *  @brief                  setzt die Spielstangenposition des Computers
00064      *  @param[in] axisPos      Translation die vom Server empfangen wurde
00065      *  @param[in] axisAngle    Rotation die vom Server empfangen wurde
00066      *  @return                 true wenn die Spielstangenpositionen erfolgreich übernommen wurden
00067      *
00068      *  Wird vom Server aufgerufen wenn eine Spielstangenposition (Fahrbefehl) vom Client eingetroffen ist.
00069      */
00070     bool setComputerAxisPos(const float axisPos[2][4], const float axisAngle[2][4]);
00071
00072     /*
00073      *  @brief              setzt den Spielstand
00074      *  @param[in] human    Spielstand des Menschen
00075      *  @param[in] computer Spielstand des Computers
00076      *  @return             true wenn der Spielstand erfolgreich übernommen wurde
00077      *
00078      *  Wird vom Server aufgerufen wenn ein Spielstand vom Client eingetroffen ist.
00079      */
00080     bool setScore(const unsigned short computer, const unsigned short human);
00081
00082     /*
00083      *  @brief                  liefert die Spielstangenposition des Menschen zurück
00084      *  @param[out] axisPos     Referenz auf die Variable in die die Translation abgelegt werden soll
00085      *  @param[out] axisAngle   Referenz auf die Variable in die die Rotation abgelegt werden soll
00086      *  @return                 true wenn die Spielstangenpositionen erfolgreich übergeben wurden
00087      *
00088      *  Wird vom Server aufgerufen wenn eine Spielstangenposition vom Client abgefragt wird.
00089      */
00090     bool getHumanAxisPos(float (&axisPos)[4], float (&axisAngle)[4]);
00091
00092     /*
00093      *  @brief                  liefert die Spielstangenposition des Computers zurück
00094      *  @param[out] axisPos     Referenz auf die Variable in die die Translation abgelegt werden soll
00095      *  @param[out] axisAngle   Referenz auf die Variable in die die Rotation abgelegt werden soll
00096      *  @return                 true wenn die Spielstangenpositionen erfolgreich übergeben wurden
00097      *
00098      *  Wird vom Server aufgerufen wenn eine Spielstangenposition vom Client abgefragt wird.
00099      */
00100     bool getComputerAxisPos(float (&axisPos)[4], float (&axisAngle)[4]);
00101
00102     /*
00103      *  @brief                  iefert den Spielstand zurück
00104      *  @param[out] human       Referenz auf die Variable in die der Spielstand des Menschen abgelegt
      werden soll
00105      *  @param[out] computer    Referenz auf die Variable in die der Spielstand des Computers abgelegt
      werden soll
00106      *  @return                 true wenn der Spielstand erfolgreich übergeben wurde
00107      *
00108      *  Wird vom Server aufgerufen wenn der Spielstand vom Client abgefragt wird.
00109      */
00110     bool getScore(unsigned short &human, unsigned short &computer);
00111
00119     bool getBallPos(ballCoordinate_t &ballPos);
00120
00121 private:
00125     ServerInterface* myServer;
```

```
00126
00133     HANDLE myMutexHandle;
00134
00141     float myHumanAxisPos[4];
00142
00149     float myHumanAxisAngle[4];
00150
00158     float myComputerAxisPos[2][4];
00159
00167     float myComputerAxisAngle[2][4];
00168
00174     unsigned myScore[2];
00175
00179     ballCoordinate_t myBallPos;
00180 };
00181
00182 #endif //SERVERUSER_H
```

7.19 serveruserinterface.h-Dateireferenz

Enthält das Interface zur Kommunikation mit dem Benutzer des UDP-Servers.

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



Klassen

- interface ServerUserInterface

  Interface zur Kommunikation mit dem Benutzer des UDP-Servers.

7.19.1 Ausführliche Beschreibung

Enthält das Interface zur Kommunikation mit dem Benutzer des UDP-Servers.

Autor

Scharel Clemens

Datum

08.02.2012

Version

0.1 Testphase

Definiert in Datei serveruserinterface.h.

7.20 serveruserinterface.h

```
00001
00010 #ifndef SERVERUSERINTERFACE_H
00011 #define SERVERUSERINTERFACE_H
00012
00013 struct ballCoordinate_t;
00014
00019 class ServerUserInterface
00020 {
00021 public:
00025     virtual ~ServerUserInterface(void) {};
00026
00031     virtual bool start() = 0;
00032
00033     /*************************************************************************
00034     *   Hier müssen die rein virtuellen Memberfunktionen definiert werden,     *
00035     *   welche vom Benutzer des Servers implementiert werden müssen.           *
00036     *   Diese Funktionen werden vom Server aufgerufen.                         *
00037     *   z.B.: zum Übergeben von empfangenen Daten, ...                         *
00038     *************************************************************************/
00039
00040     /*
00041      *  @brief                  setzt die Spielstangenposition des Computers
00042      *  @param[in] axisPos      Translation die vom Server empfangen wurde
00043      *  @param[in] axisAngle    Rotation die vom Server empfangen wurde
00044      *  @return                 true wenn die Spielstangenpositionen erfolgreich übernommen wurden
00045      *
00046      *  Wird vom Server aufgerufen wenn eine Spielstangenposition (Fahrbefehl) vom Client eingetroffen ist.
00047      */
00048     virtual bool setComputerAxisPos(const float axisPos[2][4], const float axisAngle[2][4
      ]) = 0;
00049
00050     /*
00051      *  @brief              setzt den Spielstand
00052      *  @param[in] computer Spielstand des Computers
00053      *  @param[in] human    Spielstand des Menschen
00054      *  @return             true wenn der Spielstand erfolgreich übernommen wurde
00055      *
00056      *  Wird vom Server aufgerufen wenn ein Spielstand vom Client eingetroffen ist.
00057      */
00058     virtual bool setScore(const unsigned short computer, const unsigned short human) = 0;
00059
00060     /*
00061      *  @brief                  liefert die Spielstangenposition des Menschen zurück
00062      *  @param[out] axisPos     Referenz auf die Variable in die die Translation abgelegt werden soll
00063      *  @param[out] axisAngle   Referenz auf die Variable in die die Rotation abgelegt werden soll
00064      *  @return                 true wenn die Spielstangenpositionen erfolgreich übergeben wurden
00065      *
00066      *  Wird vom Server aufgerufen wenn eine Spielstangenposition vom Client abgefragt wird.
00067      */
00068     virtual bool getHumanAxisPos(float (&axisPos)[4], float (&axisAngle)[4]) = 0;
00069
00070     /*
00071      *  @brief                  liefert die Spielstangenposition des Computers zurück
00072      *  @param[out] axisPos     Referenz auf die Variable in die die Translation abgelegt werden soll
00073      *  @param[out] axisAngle   Referenz auf die Variable in die die Rotation abgelegt werden soll
00074      *  @return                 true wenn die Spielstangenpositionen erfolgreich übergeben wurden
00075      *
00076      *  Wird vom Server aufgerufen wenn eine Spielstangenposition vom Client abgefragt wird.
00077      */
00078     virtual bool getComputerAxisPos(float (&axisPos)[4], float (&axisAngle)[4]) = 0;
00079
00080     /*
00081      *  @brief                  iefert den Spielstand zurück
00082      *  @param[out] computer    Referenz auf die Variable in die der Spielstand des Computers abgelegt
      werden soll
00083      *  @param[out] human       Referenz auf die Variable in die der Spielstand des Menschen abgelegt
      werden soll
00084      *  @return                 true wenn der Spielstand erfolgreich übergeben wurde
00085      *
00086      *  Wird vom Server aufgerufen wenn der Spielstand vom Client abgefragt wird.
00087      */
00088     virtual bool getScore(unsigned short &computer, unsigned short &human) = 0;
00089
00097     virtual bool getBallPos(ballCoordinate_t &ballPos) = 0;
```

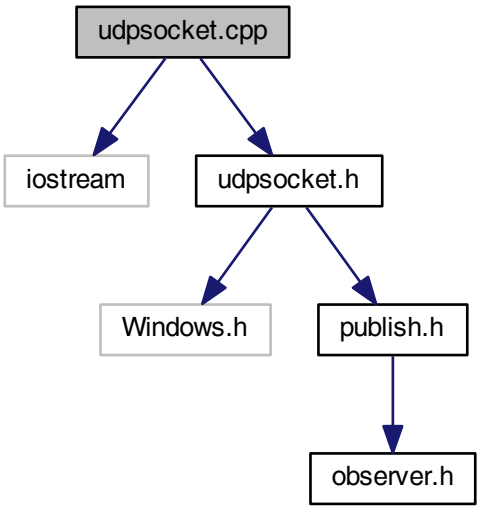
```
00098 };
00099
00100 #endif  // SERVERUSERINTERFACE_H
```

7.21 udpsocket.cpp-Dateireferenz

Enthält die Implementierung der Klasse UDPsocket.

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

Include-Abhängigkeitsdiagramm für udpsocket.cpp:



Funktionen

- bool getHostFromAddr (const sockaddr_in addr, string &host)

  Ermittelt die IP-Adresse als Zeichenkette aus einer Struktur.
- bool getPortFromAddr (const sockaddr_in addr, unsigned short &port)

  Ermittelt den Port aus einer Struktur.

7.21.1 Ausführliche Beschreibung

Enthält die Implementierung der Klasse UDPsocket.

Autor

Scharel Clemens

Datum

17.01.2012

Version

1.0 Grundfunktion vollständig implementiert

Definiert in Datei udpsocket.cpp.

7.21.2 Dokumentation der Funktionen

7.21.2.1 bool getHostFromAddr (const sockaddr_in *addr*, string & *host*)

Ermittelt die IP-Adresse als Zeichenkette aus einer Struktur.

Parameter

in	*addr*	Struktur, die die Adresse enthält
out	*host*	Zeichenkette, in die die Adresse abgelegt werden kann (Durch Punkte getrennte IP-Adresse)

Rückgabe

true wenn die Adresse ermittelt werden konnte, ansonsten false

Definiert in Zeile 254 der Datei udpsocket.cpp.

Wird benutzt von ServerUser::run().

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

7.21.2.2 bool getPortFromAddr (const sockaddr_in *addr*, unsigned short & *port*)

Ermittelt den Port aus einer Struktur.

Parameter

in	*addr*	Struktur, die die Adresse enthält
out	*port*	Variable, in die der ermittelte Port abgelegt werden soll

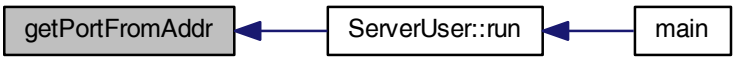
Rückgabe

true wenn der Port ermittelt werden konnte, ansonsten false

Definiert in Zeile 262 der Datei udpsocket.cpp.

Wird benutzt von ServerUser::run().

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



7.22 udpsocket.cpp

```
00001
00010 #include <iostream>
00011 using namespace std;
00012
00013 #include "udpsocket.h"
00014
00015 UDPsocket::UDPsocket(const unsigned int bufferSize, const unsigned short localPort)
00016     : myBufferSize(bufferSize)
00017 {
00018     myPaused = true;
00019     myStopped = true;
00020     //  Windows anweisen Winsock 2.0 zu initialisieren
00021     myWsaErr = WSAStartup(MAKEWORD(2,0),&myWsa);
00022
00023     //  Eigene Socket-Adresse umwandeln
00024     myAddr.sin_family = AF_INET;
00025     myAddr.sin_port = htons(localPort);
00026     myAddr.sin_addr.s_addr = htonl(INADDR_ANY);
00027 }
00028
00029 UDPsocket::~UDPsocket()
00030 {
00031     myPaused = true;
00032     myStopped = true;
00033
00034     if (myThreadHandle)
00035     {
00036         //  Warten auf das Ende von recvData(), max. 100 msec lang
00037         DWORD reply = WaitForSingleObjectEx(myThreadHandle, 100, true);
00038         if(reply != WAIT_OBJECT_0)
00039         {
00040             cout << "Empfangs-Thread konnte nicht sachgemaess beendet werden";
00041             if (!TerminateThread(myThreadHandle, EXIT_SUCCESS))
00042                 cout << ": " << GetLastError() << endl;
00043             else
00044                 cout << "!" << endl;
00045         }
00046
00047         /*switch (reply)
00048         {
00049         case WAIT_ABANDONED:
00050             //cout << "WaitForSingleObjectEx: WAIT_ABANDONED" << endl;
00051             break;
00052         case WAIT_IO_COMPLETION:
00053             //cout << "WaitForSingleObjectEx: WAIT_IO_COMPLETION" << endl;
00054             break;
00055         case WAIT_OBJECT_0:
00056             // Empfangs-Thread wurde richtig beendet
00057             //cout << "WaitForSingleObjectEx: WAIT_OBJECT_0" << endl;
00058             break;
00059         case WAIT_TIMEOUT:
00060             //cout << "WaitForSingleObjectEx: WAIT_TIMEOUT" << endl;
00061             //cout << "Empfangs-Tread wird beendet." << endl;
00062             if (!TerminateThread(myThreadHandle, 0))
```

```
00063                 cout << "Empfangs-Thread konnte nicht sachgemaess beendet werden!" << endl;
00064             //cout << "Empfangs-Tread wurde beendet." << endl;
00065             break;
00066         case WAIT_FAILED:
00067             //cout << "WaitForSingleObjectEx: WAIT_FAILED" << endl;
00068             break;
00069         default:
00070             //cout << "WaitForSingleObjectEx: Fehler beim Warten auf recvData()" << endl;
00071             break;
00072         }*/
00073     }
00074
00075     if (mySock)
00076         closesocket(mySock);
00077     //  Winsock aufräumen
00078     WSACleanup();
00079
00080     //  Buffer freigeben
00081     delete[] myBuffer;
00082     myBuffer = NULL;
00083 }
00084
00085 bool UDPsocket::start()
00086 {
00087     bool retval = false;
00088
00089     //  Überprüfen ob start() schon einmal aufgerugen wurde
00090     if (myPaused && myStopped)
00091     {
00092         int iResult;
00093         //  Überprüfen ob WSA im Konstruktor gestartet werden konnte
00094         if (myWsaErr || LOBYTE(myWsa.wVersion) != 2 || HIBYTE(myWsa.wVersion) != 0)
00095             cout << "Winsock liegt nicht in der benötigten Version (2.0) vor!" << endl;
00096         else
00097         {
00098             //  Neuen Socket anlegen
00099             mySock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
00100             if (mySock == INVALID_SOCKET)
00101                 cout << "Fehler beim Anlegen des Sockets: " << WSAGetLastError() << "!" << endl;
00102             else
00103             {
00104                 //  Socket binden
00105                 iResult = bind(mySock, (SOCKADDR*)&myAddr, sizeof(
      myAddr));
00106                 if (iResult != 0)
00107                     cout << "Fehler beim Binden des Sockets: " << WSAGetLastError() << "!" << endl;
00108                 else
00109                 {
00110                     //  Empfangsspeicher anlegen
00111                     myBuffer = new char[myBufferSize];
00112                     myDataSize = SOCKET_ERROR;
00113
00114                     //  Neuen Thread zum Empfangen der Pakete erzeugen
00115                     myThreadHandle = NULL;
00116                     myThreadHandle = CreateThread(NULL, 0,
      getRecvDataInstance,
00117                                                   (void*)this, 0, NULL);
00118                     if (!myThreadHandle)
00119                         cout << "Empfangs-Thread konnte nicht erzeugt werden!" << endl;
00120                     else
00121                     {
00122                         //cout << "UDP-Socket gestartet!" << endl;
00123                         myPaused = false;
00124                         myStopped = false;
00125                         retval = true;
00126                     }
00127                 }
00128             }
00129         }
00130     }
00131     else
00132         retval = true;
00133     return retval;
00134 }
00135
00136 void UDPsocket::pause(const bool pause)
00137 {
00138     //  Wenn pausiert erhält der Observer keine Notification mehr
00139     myPaused = pause;
00140 }
00141
00142 int UDPsocket::sendData(const char* const data, const int len, const sockaddr_in recvr)
00143 {
00144     //  Daten senden
00145     int iResult = SOCKET_ERROR;
00146     iResult = sendto(mySock, data, len, 0, (SOCKADDR*)&recvr, sizeof(recvr));
00147     if (iResult != len)
```

```
00148         cout << "Fehler beim Senden eines Paketes: " << WSAGetLastError() << "!" << endl;
00149     return iResult;
00150 }
00151
00152 int UDPsocket::getData(const char* &data, sockaddr_in &sendr)
00153 {
00154     data = myBuffer;
00155     sendr = mySenderAddr;
00156     return myDataSize;
00157 }
00158
00159 bool UDPsocket::getAddrFromString(const string host, sockaddr_in &addr)
00160 {
00161     bool retval = false;
00162
00163     // Parameter prüfen
00164     if (!host.empty())
00165     {
00166         unsigned long ulIP;
00167         // eine IP in geeignete Form für IN_ADDR bringen
00168         ulIP = inet_addr(host.c_str());
00169
00170         /* bei einem fehler liefert inet_addr den Rückgabewert INADDR_NONE */
00171         if (ulIP != INADDR_NONE)
00172         {
00173             //cout << "IP-Adresse aufgeloest: " << host << endl;
00174             addr.sin_addr.s_addr = ulIP;
00175             retval = true;
00176         }
00177         else
00178         {
00179             /* Hostname in geeignete Form für IN_ADDR bringen */
00180             struct hostent *he;
00181             he = gethostbyname(host.c_str());
00182             if (he != NULL)
00183             {
00184                 //cout << "Probiere die IP-Adresse aufzuloesen..." << endl;
00185                 int i = 0;
00186                 struct in_addr ip4;
00187                 while (he->h_addr_list[i] != 0) // (evtl. auskommentieren)
00188                     ip4.s_addr = *(u_long *) he->h_addr_list[i++];
00189                 ulIP = inet_addr(inet_ntoa(ip4));
00190                 if (ulIP != INADDR_NONE && ulIP != INADDR_ANY)
00191                 {
00192                     addr.sin_addr.s_addr = ulIP;
00193                     retval = true;
00194                 }
00195             }
00196         }
00197     }
00198     return retval;
00199 }
00200
00201 //  Stellt ein eigener Thread dar
00202 DWORD UDPsocket::recvData()
00203 {
00204     DWORD retval;
00205     int iResult;
00206     int senderAddrSize = sizeof(mySenderAddr);
00207
00208     //  Daten empfangen bis der Client gestoppt wird
00209     do
00210     {
00211         //  Blockierende Funktion, welche auf Daten wartet
00212         iResult = recvfrom(mySock, myBuffer, myBufferSize, 0, (SOCKADDR*)&
      mySenderAddr, &senderAddrSize);
00213         //iResult = 12;
00214         if (iResult == SOCKET_ERROR)
00215             cout << "Fehler beim Empfangen eines Pakets: " << WSAGetLastError() << "!" << endl;
00216         //else if (iResult == 0)
00217         //    cout << "Socket wurde beendet!" << endl;
00218         else
00219         {
00220             myDataSize = iResult;
00221             //  Observer nur benachichtigen wenn nicht pausiert
00222             if (!myPaused)
00223                 notify();
00224         }
00225     } while(iResult != SOCKET_ERROR && myStopped == false);
00226
00227     //  Client wird gestoppt
00228     if (myStopped)
00229     {
00230         //  Socket schliessen
00231         if (closesocket(mySock) == SOCKET_ERROR)
00232             cout << "Socket konnte nicht sachgemaess geschlossen werden!" << endl;
00233         else
```

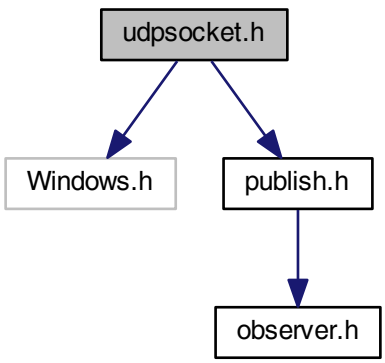
```
00234             mySock = NULL;
00235     }
00236
00237     if (iResult != 0)
00238         retval = -1;
00239     else
00240         retval = 0;
00241     return retval;
00242 }
00243
00244 DWORD WINAPI UDPsocket::getRecvDataInstance(void* instance)
00245 {
00246     UDPsocket* This = (UDPsocket*) instance;
00247     DWORD retval = NULL;
00248     //  Zeiger auf die Funktion recvData()
00249     retval = This->recvData();
00250     return retval;
00251 }
00252
00253
00254 bool getHostFromAddr(const sockaddr_in addr, string& host)
00255 {
00256     bool retval = false;
00257     host = inet_ntoa(addr.sin_addr);
00258     retval = true;
00259     return retval;
00260 }
00261
00262 bool getPortFromAddr(const sockaddr_in addr, unsigned short& port)
00263 {
00264     bool retval = false;
00265     port = ntohs(addr.sin_port);
00266     retval = true;
00267     return retval;
00268 }
```

7.23 udpsocket.h-Dateireferenz

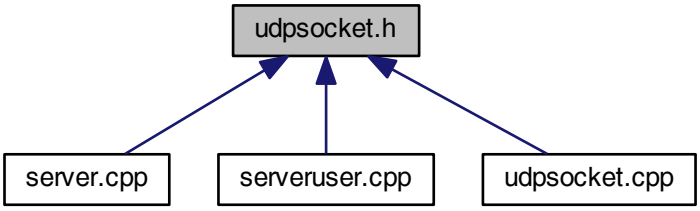
Enthält die Definition der Klasse UDPsocket.

```
#include <Windows.h>
#include "publish.h"
```

Include-Abhängigkeitsdiagramm für udpsocket.h:



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



Klassen

- class UDPsocket

  Stellt ein UDP-Socket dar.

Funktionen

- bool getHostFromAddr (const sockaddr_in addr, string &host)

  Ermittelt die IP-Adresse als Zeichenkette aus einer Struktur.
- bool getPortFromAddr (const sockaddr_in addr, unsigned short &port)

  Ermittelt den Port aus einer Struktur.

7.23.1 Ausführliche Beschreibung

Enthält die Definition der Klasse UDPsocket.

Autor

Scharel Clemens

Datum

07.12.2012

Version

1.1 Grundfunktion vollständig implementiert Diese Datei muss dafür sorgen, dass "sockaddr_in" definiert ist!

Definiert in Datei udpsocket.h.

7.23.2 Dokumentation der Funktionen

7.23.2.1 bool getHostFromAddr (const sockaddr_in *addr*, string & *host*)

Ermittelt die IP-Adresse als Zeichenkette aus einer Struktur.

Parameter

in	*addr*	Struktur, die die Adresse enthält
out	*host*	Zeichenkette, in die die Adresse abgelegt werden kann (Durch Punkte getrennte IP-Adresse)

Rückgabe

true wenn die Adresse ermittelt werden konnte, ansonsten false

Definiert in Zeile 254 der Datei udpsocket.cpp.

Wird benutzt von ServerUser::run().

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

7.23.2.2 bool getPortFromAddr (const sockaddr_in *addr*, unsigned short & *port*)

Ermittelt den Port aus einer Struktur.

Parameter

in	*addr*	Struktur, die die Adresse enthält
out	*port*	Variable, in die der ermittelte Port abgelegt werden soll

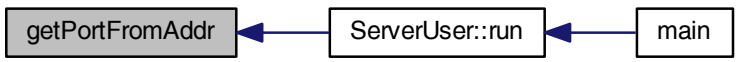
Rückgabe

true wenn der Port ermittelt werden konnte, ansonsten false

Definiert in Zeile 262 der Datei udpsocket.cpp.

Wird benutzt von ServerUser::run().

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



7.24 udpsocket.h

```
00001
```

```
00012 #ifndef UDPSOCKET_H
00013 #define UDPSOCKET_H
00014
00015 #include <Windows.h>
00016
00017 #pragma comment(lib, "ws2_32.lib")
00018
00019 #include "publish.h"
00020
00027 bool getHostFromAddr(const sockaddr_in addr, string &host);
00028
00035 bool getPortFromAddr(const sockaddr_in addr, unsigned short &port);
00036
00037
00047 class UDPsocket : public Publish
00048 {
00049 public:
00058     UDPsocket(const unsigned int bufferSize, const unsigned short localPort);
00059
00065     ~UDPsocket();
00066
00076     bool start();
00077
00086     void pause(const bool pause = true);
00087
00095     int sendData(const char* const data, const int len, const sockaddr_in recvr);
00096
00107     int getData(const char* &data, sockaddr_in &sendr);
00108
00115     bool getAddrFromString(const string host, sockaddr_in &addr);
00116
00117 private:
00123     WSADATA myWsa;
00124
00132     int myWsaErr;
00133
00139     SOCKET mySock;
00140
00148     HANDLE myThreadHandle;
00149
00154     sockaddr_in myAddr;
00155
00162     sockaddr_in mySenderAddr;
00163
00169     bool myPaused;
00170
00176     bool myStopped;
00177
00182     unsigned int myBufferSize;
00183
00191     char* myBuffer;
00192
00201     int myDataSize;
00202
00210     DWORD recvData();
00211
00220     static DWORD WINAPI getRecvDataInstance(void* instance);
00221 };
00222
00223 #endif  // UDPCLIENT_H
```

Index