CompleteClient

0.9

Erzeugt von Doxygen 1.8.3

Mit Jan 9 2013 16:12:10

Inhaltsverzeichnis

Kapitel 1

Beispielimplementierung eines Client-Programms

1.1 Einführung

Dieses Projekt beinhaltet eine Beispielimplementierung eines Client-Programms, um zu demonstrieren, wie die `Interprozesskommunikation` zu verwenden ist. Alle, bisher im `Projekt Computerkicker (Pro-CK)` verwendete Daten, können mit diesem Client 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 Client-Programm verwendet werden.

Folgende Dateien sollten dazu in das neue Projekt kopiert werden:

- client.h
- client.cpp
- clientinterface.h
- clientuserinterface.h
- defines.h
- observer.h
- publish.h
- udpsocket.h
- udpsocket.cpp

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

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

Im vorliegenden Projekt, wurde eine, von von ClientUserInterface abgeleitete Klasse (ClientUser) erzeugt. Diese Klasse verwaltet die gesamten übertragbaren Daten und stellt eine Konsolen-Benutzerschnittstelle dar, mit der die Daten manipuliert und mit dem Server 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 Clients

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

Dies führt zu folgender Konfiguration:

Konstruktor der von ClientUserInterface abgeleiteten Klasse (ClientUser::ClientUser()) ruft den Konstruktor von Client auf (Client::Client())

Client::Client() ruft den Konstruktor von UDPsocket auf (UDPsocket::UDPsocket()), und

löst die Server-Adresse auf (UDPsocket::getAddrFromString())

UDPsocket::UDPsocket() Startet `Winsock`

Beim Aufruf der Funktion Client::start():

Die von ClientUserInterface abgeleiteten Klasse (ClientUser) startet Client (Client::start())

Client::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 Clients

Nachdem alle Klassen angelegt und gestartet worden sind, kann mit dem Server kommuniziert werden. Hierzu stehen aus Client-Sicht, die von ClientInterface 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 Server übertragen.

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

Die Alive-Pakete, bilden hier eine Ausnahme: eine Alive-Anfrage, wird automatisch beantwortet und eine Alive--Antwort, setzt die interne Toggle-Variable (Client::myAliveToggle) auf true, um anzugeben dass der Server verfügbar ist.

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 Client-Programms

Namen für das auf UDP aufgesetzte Protokoll suchen.

Element ClientUser::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

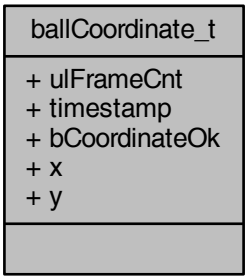
Klassen-Dokumentation

6.1 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

6.1.1 Ausführliche Beschreibung

Struktur der Ballposition.

Definiert in Zeile 337 der Datei defines.h.

6.1.2 Dokumentation der Datenelemente

6.1.2.1 bool ballCoordinate_t::bCoordinateOk

True wenn Ballposition erfolgreich bestimmt weden konnte

Definiert in Zeile 344 der Datei defines.h.

6.1.2.2 double ballCoordinate_t::timestamp

Zeitstempel der linken Kamera

Definiert in Zeile 342 der Datei defines.h.

6.1.2.3 unsigned long ballCoordinate_t::ulFrameCnt

Bildpaare seit Start

Definiert in Zeile 340 der Datei defines.h.

6.1.2.4 float ballCoordinate_t::x

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

Definiert in Zeile 346 der Datei defines.h.

6.1.2.5 float ballCoordinate_t::y

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

Definiert in Zeile 348 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

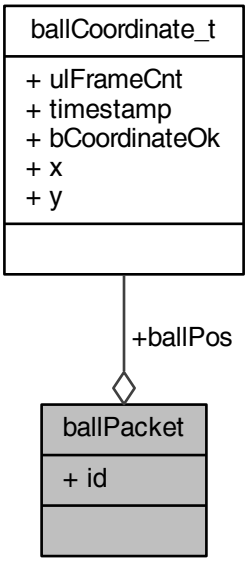
- defines.h

6.2 ballPacket Strukturreferenz

Paket mit der Ballposition.

`#include <defines.h>`

Zusammengehörigkeiten von ballPacket:



Öffentliche Attribute

- unsigned int id: 16
- ballCoordinate_t ballPos

6.2.1 Ausführliche Beschreibung

Paket mit der Ballposition.

Beinhaltet die Paket-ID und eine Struktur mit der Ballposition

Definiert in Zeile 356 der Datei defines.h.

6.2.2 Dokumentation der Datenelemente

6.2.2.1 ballCoordinate_t ballPacket::ballPos

Definiert in Zeile 359 der Datei defines.h.

Wird benutzt von Client::update().

6.2.2.2 unsigned int ballPacket::id

Definiert in Zeile 358 der Datei defines.h.

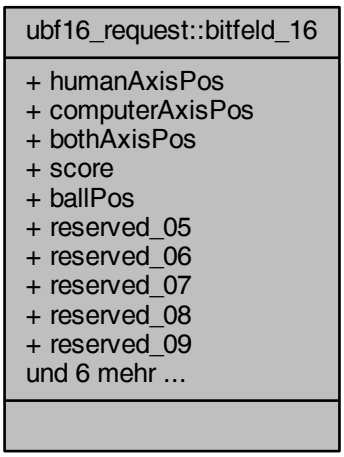
Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- defines.h

6.3 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

6.3.1 Ausführliche Beschreibung

Definiert in Zeile 191 der Datei defines.h.

6.3.2 Dokumentation der Datenelemente

6.3.2.1 unsigned int ubf16_request::bitfeld_16::ballPos

Definiert in Zeile 197 der Datei defines.h.

Wird benutzt von Client::requestBallPos() und Client::subscribeBallPos().

6.3.2.2 unsigned int ubf16_request::bitfeld_16::bothAxisPos

Definiert in Zeile 195 der Datei defines.h.

Wird benutzt von Client::requestAxisPos() und Client::subscribeAxisPos().

6.3.2.3 unsigned int ubf16_request::bitfeld_16::computerAxisPos

Definiert in Zeile 194 der Datei defines.h.

Wird benutzt von Client::requestAxisPos() und Client::subscribeAxisPos().

6.3.2.4 unsigned int ubf16_request::bitfeld_16::humanAxisPos

Definiert in Zeile 193 der Datei defines.h.

Wird benutzt von Client::requestAxisPos() und Client::subscribeAxisPos().

6.3.2.5 unsigned int ubf16_request::bitfeld_16::reserved_05

Definiert in Zeile 198 der Datei defines.h.

6.3.2.6 unsigned int ubf16_request::bitfeld_16::reserved_06

Definiert in Zeile 199 der Datei defines.h.

6.3.2.7 unsigned int ubf16_request::bitfeld_16::reserved_07

Definiert in Zeile 200 der Datei defines.h.

6.3.2.8 unsigned int ubf16_request::bitfeld_16::reserved_08

Definiert in Zeile 201 der Datei defines.h.

6.3.2.9 unsigned int ubf16_request::bitfeld_16::reserved_09

Definiert in Zeile 202 der Datei defines.h.

6.3.2.10 unsigned int ubf16_request::bitfeld_16::reserved_10

Definiert in Zeile 203 der Datei defines.h.

6.3.2.11 unsigned int ubf16_request::bitfeld_16::reserved_11

Definiert in Zeile 204 der Datei defines.h.

6.3.2.12 unsigned int ubf16_request::bitfeld_16::reserved_12

Definiert in Zeile 205 der Datei defines.h.

6.3.2.13 unsigned int ubf16_request::bitfeld_16::reserved_13

Definiert in Zeile 206 der Datei defines.h.

6.3.2.14 unsigned int ubf16_request::bitfeld_16::reserved_14

Definiert in Zeile 207 der Datei defines.h.

6.3.2.15 unsigned int ubf16_request::bitfeld_16::reserved_15

Definiert in Zeile 208 der Datei defines.h.

6.3.2.16 unsigned int ubf16_request::bitfeld_16::score

Definiert in Zeile 196 der Datei defines.h.

Wird benutzt von Client::requestScore() und Client::subscribeScore().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

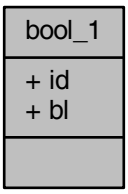
- defines.h

6.4 bool_1 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von bool_1:



Öffentliche Attribute

- unsigned int id: 16
- bool bl

6.4.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein bool-Wert.

Definiert in Zeile 262 der Datei defines.h.

6.4.2 Dokumentation der Datenelemente

6.4.2.1 bool bool_1::bl

Definiert in Zeile 265 der Datei defines.h.

6.4.2.2 unsigned int bool_1::id

Definiert in Zeile 264 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

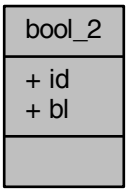
- defines.h

6.5 bool_2 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von bool_2:



Öffentliche Attribute

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

6.5.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und zwei bool-Werte.

Definiert in Zeile 273 der Datei defines.h.

6.5.2 Dokumentation der Datenelemente

6.5.2.1 bool bool_2::bl[2]

Definiert in Zeile 276 der Datei defines.h.

6.5.2.2 unsigned int bool_2::id

Definiert in Zeile 275 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

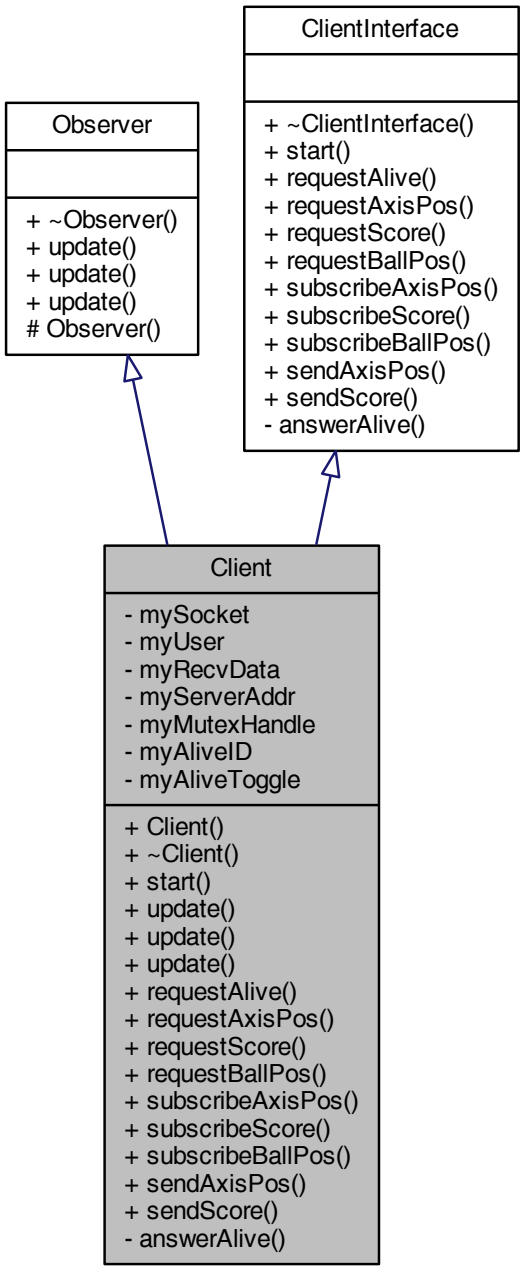
- defines.h

6.6 Client Klassenreferenz

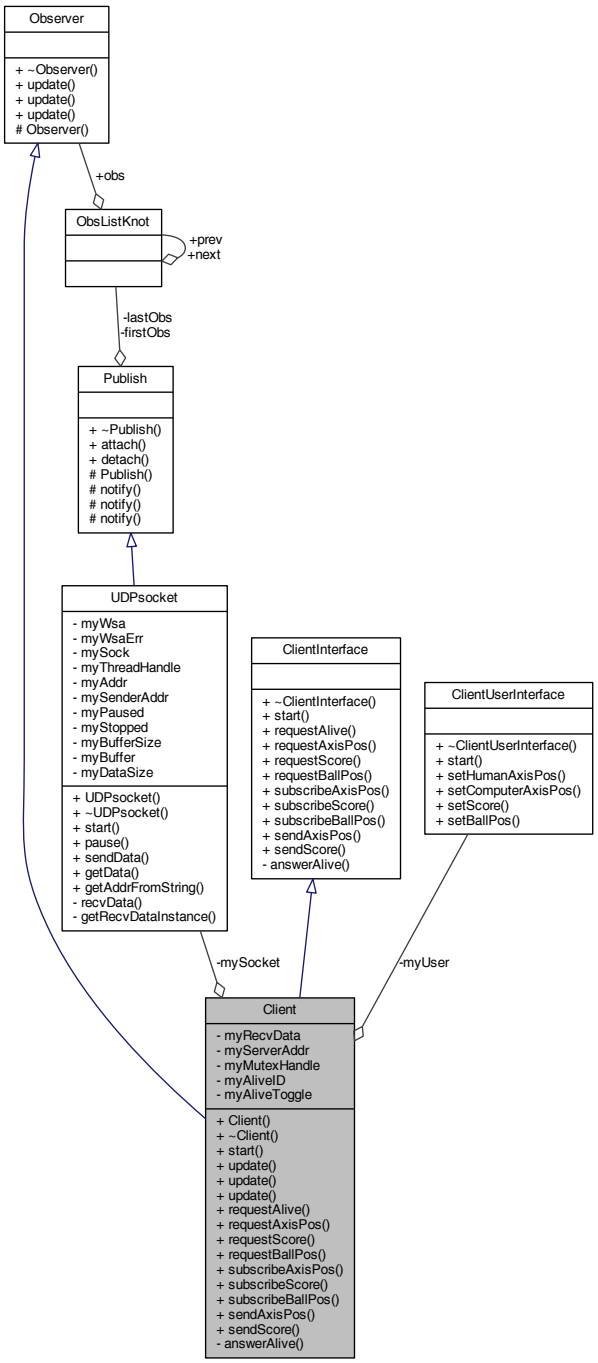
ein UDP-Client

`#include <client.h>`

Klassendiagramm für Client:



Zusammengehörigkeiten von Client:



Öffentliche Methoden

- Client (const unsigned short localPort, const string serverHost, const unsigned short serverPort)

  Konstruktor.
- ~Client ()

  Destruktor.
- bool start (ClientUserInterface *const user)

Startet den Client.

- void update (Publish ∗const pub)

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

  update-Funktion der Observer Klasse
- void update (Publish ∗const pub, const float ∗const fValues, const unsigned int fValuesSize, const int ∗const iValues=NULL, const unsigned int iValuesSize=0)

  update-Funktion der Observer Klasse
- bool requestAlive (void)

  Sendet ein Alive-Anfragepaket.
- bool requestAxisPos (const Gamer gamer=BOTH)

  Fragt die Spielstangenpositionen beim Server an.
- bool requestScore ()

  Fragt die Spielstangenpositionen beim Server an.
- bool requestBallPos ()

  Fragt die Ballposition beim Server an.
- bool subscribeAxisPos (const Gamer gamer=BOTH, const bool subs=true)

  Abonniert die Spielstangenpositionen beim Server.
- bool subscribeScore (const bool subs=true)

  Abonniert den Spielstand beim Server.
- bool subscribeBallPos (const bool subs=true)

  Abonniert die Ballposition beim Server.
- bool sendAxisPos (const float axisPos[2][4], const float axisAngle[2][4])

  Sendet die Spielstangenpositionen an den Server.
- bool sendScore (const unsigned short computer, const unsigned short human)

  Fragt die Spielstangenpositionen beim Server an.

Private Methoden

- bool answerAlive (const long aliveID)

  Beantwortet ein Alive-Paket.

Private Attribute

- UDPsocket ∗ mySocket

  der UDP-Socket
- ClientUserInterface ∗ myUser

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

  Speicher für die empfangenen Daten.
- sockaddr_in myServerAddr

  Adresse des Servers.
- HANDLE myMutexHandle

  Mutex-Handle.
- long myAliveID

  ID des aktuellen Alive-Pakets.
- bool myAliveToggle

  Gibt an ob der Server auf Alive-Anfragen antwortet.

Weitere Geerbte Elemente

6.6.1 Ausführliche Beschreibung

ein UDP-Client

Beispielimplementierung eines UDP-Client unter Verwendung der Klasse UDPsocket.

Definiert in Zeile 26 der Datei client.h.

6.6.2 Beschreibung der Konstruktoren und Destruktoren

6.6.2.1 Client::Client (const unsigned short *localPort*, const string *serverHost*, const unsigned short *serverPort*)

Konstruktor.

Parameter

in	*localPort*	eigenen IP-Port
in	*serverHost*	Rechnername oder IP-Adresse des Servers
in	*serverPort*	Port des Servers

Ermittelt die Server-Adresse und legt den Empfangsspeicher und die Klasse UDPsocket an.

Definiert in Zeile 17 der Datei client.cpp.

Benutzt NULL und RECV_BUFFER_SIZE.

6.6.2.2 Client::~Client ( )

Destruktor.

Schließt den UDP-Socket und gibt den Empfangsspeicher wieder frei.

Definiert in Zeile 38 der Datei client.cpp.

Benutzt NULL.

6.6.3 Dokumentation der Elementfunktionen

6.6.3.1 bool Client::answerAlive (const long *aliveID*) [private], [virtual]

Beantwortet ein Alive-Paket.

Parameter

in	*aliveID*	eindeutige ID, die vom Server gesendet wurde um die Antwort der Anfrage zuzuordnen

Rückgabe

true wenn die Antwort gesendet wurde

Implementiert ClientInterface.

Definiert in Zeile 237 der Datei client.cpp.

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

6.6.3.2 bool Client::requestAlive (void) [virtual]

Sendet ein Alive-Anfragepaket.

Rückgabe

Die Antwort entspricht dem Wert von myAliveToggle, vor dem Funktionsaufruf

Wenn diese Funktion true zurückgibt, so bedeutet dies dass der Server seit dem letzten Aufruf dieser Funktion auf das Alive-Paket geantwortet hat.

Implementiert ClientInterface.

Definiert in Zeile 209 der Datei client.cpp.

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

6.6.3.3 bool Client::requestAxisPos (const Gamer *gamer* = BOTH) [virtual]

Fragt die Spielstangenpositionen beim Server an.

Parameter

in	*gamer*	HUMAN, COMPUTER oder BOTH, je nachdem für wen die Spielstangenpositionen abgefragt werden sollen

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert ClientInterface.

Definiert in Zeile 256 der Datei client.cpp.

Benutzt BOTH, ubf16_request::bitfeld_16::bothAxisPos, COMPUTER, ubf16_request::bitfeld_16::computerAxisPos, ubf16_request::data, HUMAN, ubf16_request::bitfeld_16::humanAxisPos, requestPacket::id, REQUEST, requestPacket::request und ubf16_request::value.

6.6.3.4 bool Client::requestBallPos ( ) [virtual]

Fragt die Ballposition beim Server an.

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert ClientInterface.

Definiert in Zeile 307 der Datei client.cpp.

Benutzt ubf16_request::bitfeld_16::ballPos, ubf16_request::data, requestPacket::id, REQUEST, requestPacket::request und ubf16_request::value.

6.6.3.5 bool Client::requestScore ( ) [virtual]

Fragt die Spielstangenpositionen beim Server an.

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert ClientInterface.

Definiert in Zeile 287 der Datei client.cpp.

Benutzt ubf16_request::data, requestPacket::id, REQUEST, requestPacket::request, ubf16_request::bitfeld_16-::score und ubf16_request::value.

6.6.3.6 bool Client::sendAxisPos (const float *axisPos[2][4],* const float *axisAngle[2][4]*) [virtual]

Sendet die Spielstangenpositionen an den Server.

Parameter

in	*axisPos*	Array mit der Translation (Wert und Geschwindigkeit)
in	*axisAngle*	Array mit der Rotation (Wert und Geschwindigkeit)

Rückgabe

true wenn die Spielstangenpositionen gesendet wurde

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]).

Implementiert ClientInterface.

Definiert in Zeile 401 der Datei client.cpp.

Benutzt ANGLE, AXIS_FOUR, AXIS_ONE, float_2_2_4::flt, float_2_2_4::id, POSITION, SET_AXISPOS, VALUE und VELOCITY.

6.6.3.7 bool Client::sendScore (const unsigned short *computer,* const unsigned short *human*) [virtual]

Fragt die Spielstangenpositionen beim Server an.

Parameter

in	*computer*	Spielstand des Computers
in	*human*	Spielstand des menschlichen Spielers

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert ClientInterface.

Definiert in Zeile 427 der Datei client.cpp.

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

6.6.3.8 bool Client::start (ClientUserInterface *const *user*) [virtual]

Startet den Client.

Parameter

in	*user*	Zeiger auf den Benutzer des Clients

Rückgabe

true wenn alles beim Starten des Clients geklappt hat

Fügt den Client als Abonnent zum Socket hinzu und startet den Socket.

Implementiert ClientInterface.

Definiert in Zeile 49 der Datei client.cpp.

6.6.3.9 bool Client::subscribeAxisPos (const Gamer *gamer* = BOTH, const bool *subs* = `true`) [virtual]

Abonniert die Spielstangenpositionen beim Server.

Parameter

in	*gamer*	HUMAN, COMPUTER oder BOTH, je nachdem für wen die Spielstangenpositionen abonniert werden sollen
in	*subs*	true zum Abonnieren, false zum Kündigen auf false um das Abonnement zu kündigen

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert ClientInterface.

Definiert in Zeile 327 der Datei client.cpp.

Benutzt BOTH, ubf16_request::bitfeld_16::bothAxisPos, COMPUTER, ubf16_request::bitfeld_16::computerAxisPos, ubf16_request::data, HUMAN, ubf16_request::bitfeld_16::humanAxisPos, subscribePacket::id, subscribePacket::initiate, SUBSCRIBE, subscribePacket::subscribe und ubf16_request::value.

6.6.3.10 bool Client::subscribeBallPos (const bool *subs* = `true`) [virtual]

Abonniert die Ballposition beim Server.

Parameter

in	*subs*	true zum Abonnieren, false zum Kündigen

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert ClientInterface.

Definiert in Zeile 380 der Datei client.cpp.

Benutzt ubf16_request::bitfeld_16::ballPos, ubf16_request::data, subscribePacket::id, subscribePacket::initiate, SUBSCRIBE, subscribePacket::subscribe und ubf16_request::value.

6.6.3.11 bool Client::subscribeScore (const bool *subs* = `true`) [virtual]

Abonniert den Spielstand beim Server.

Parameter

in	*subs*	true zum Abonnieren, false zum Kündigen auf false um das Abonnement zu kündigen

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert ClientInterface.

Definiert in Zeile 359 der Datei client.cpp.

Benutzt ubf16_request::data, subscribePacket::id, subscribePacket::initiate, ubf16_request::bitfeld_16::score, SUBSCRIBE, subscribePacket::subscribe und ubf16_request::value.

6.6.3.12 void Client::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 66 der Datei client.cpp.

Benutzt ALIVE_ANSWER, ALIVE_REQUEST, ANGLE, ballPacket::ballPos, COMPUTER, float_2_4::flt, float_2_2_4::flt, GET_BALLPOS, GET_BOTH_AXISPOS, GET_COMPUTER_AXISPOS, GET_HUMAN_AXISPOS, GET_SCORE, HUMAN, long_1::lng, POSITION und unsignedShort_2::us.

6.6.3.13 void Client::update (Publish *const *pub*, const int *const *iValues*, const unsigned int *iValuesSize*) [virtual]

update-Funktion der Observer Klasse

Parameter

in	*pub*	Publisher Objekt, welches über ein Ereignis benachichtigt (hier UDPsocket)
in	*iValues*	wird ignoriert
in	*iValuesSize*	wird ignoriert

Wird von UDPsocket aufgerufen wenn Daten eingetroffen sind.

Implementiert Observer.

Definiert in Zeile 193 der Datei client.cpp.

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

update-Funktion der Observer Klasse

Parameter

in	*pub*	Publisher Objekt, welches über ein Ereignis benachichtigt (hier UDPsocket)
in	*fValues*	wird ignoriert
in	*fValuesSize*	wird ignoriert
in	*iValues*	wird ignoriert
in	*iValuesSize*	wird ignoriert

Wird von UDPsocket aufgerufen wenn Daten eingetroffen sind.

Implementiert Observer.

Definiert in Zeile 198 der Datei client.cpp.

6.6.4 Dokumentation der Datenelemente

6.6.4.1 long Client::myAliveID [private]

ID des aktuellen Alive-Pakets.

Wird beim Aufruf der Funktion requestAlive() gesetzt und beim Empfang eines ALIVE_ANSWER-Pakets überprüft.

Definiert in Zeile 199 der Datei client.h.

6.6.4.2 bool Client::myAliveToggle [private]

Gibt an ob der Server auf Alive-Anfragen antwortet.

Siehe auch

requestAlive()

Wird von requestAlive() auf false gesetzt. Beim eintreten der Alive-Antwort, wird die Variable wieder auf true gesetzt. Bei jedem Aufruf von requestAlive(), wird dieser Wert zurückgegeben.

Definiert in Zeile 208 der Datei client.h.

6.6.4.3 HANDLE Client::myMutexHandle [private]

Mutex-Handle.

Mutex-Handle zum sicheren Datenzugriff da myAliveID von einem anderen Thread abgefragt werden kann.

Definiert in Zeile 192 der Datei client.h.

6.6.4.4 const char* Client::myRecvData [private]

Speicher für die empfangenen Daten.

Definiert in Zeile 180 der Datei client.h.

6.6.4.5 sockaddr_in Client::myServerAddr [private]

Adresse des Servers.

Definiert in Zeile 185 der Datei client.h.

6.6.4.6 UDPsocket* Client::mySocket [private]

der UDP-Socket

Definiert in Zeile 170 der Datei client.h.

6.6.4.7 ClientUserInterface* Client::myUser [private]

Zeiger auf den Benutzer dieser Klasse.

Definiert in Zeile 175 der Datei client.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- client.h

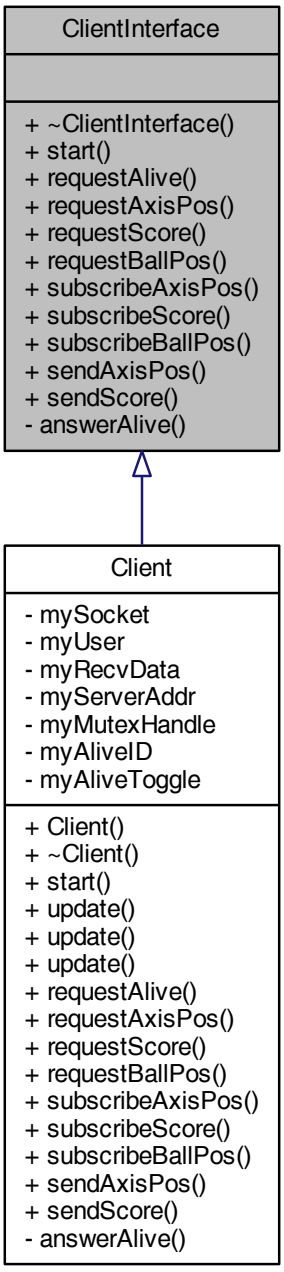
- client.cpp

6.7 ClientInterface Schnittstellenreferenz

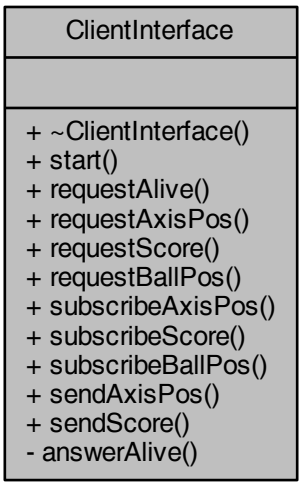
Interface zur Kommunikation mit dem UDP-Client.

`#include <clientinterface.h>`

Klassendiagramm für ClientInterface:



Zusammengehörigkeiten von ClientInterface:



Öffentliche Methoden

- virtual ~ClientInterface (void)

  Virtueller Destruktor.
- virtual bool start (ClientUserInterface *const user)=0

  Startet den Client.
- virtual bool requestAlive (void)=0

  Sendet ein Alive-Anfragepaket.
- virtual bool requestAxisPos (const Gamer gamer)=0

  Fragt die Spielstangenpositionen beim Server an.
- virtual bool requestScore ()=0

  Fragt die Spielstangenpositionen beim Server an.
- virtual bool requestBallPos ()=0

  Fragt die Ballposition beim Server an.
- virtual bool subscribeAxisPos (const Gamer gamer, const bool subs=true)=0

  Abonniert die Spielstangenpositionen beim Server.
- virtual bool subscribeScore (const bool subs=true)=0

  Abonniert den Spielstand beim Server.
- virtual bool subscribeBallPos (const bool subs=true)=0

  Abonniert die Ballposition beim Server.
- virtual bool sendAxisPos (const float axisPos[2][4], const float axisAngle[2][4])=0

  Sendet die Spielstangenpositionen an den Server.
- virtual bool sendScore (const unsigned short computer, const unsigned short human)=0

  Fragt die Spielstangenpositionen beim Server an.

Private Methoden

- virtual bool answerAlive (const long aliveID)=0

  Beantwortet ein Alive-Paket.

6.7.1 Ausführliche Beschreibung

Interface zur Kommunikation mit dem UDP-Client.

Definiert in Zeile 20 der Datei clientinterface.h.

6.7.2 Beschreibung der Konstruktoren und Destruktoren

6.7.2.1 virtual ClientInterface::~ClientInterface (void) [inline], [virtual]

Virtueller Destruktor.

Definiert in Zeile 26 der Datei clientinterface.h.

6.7.3 Dokumentation der Elementfunktionen

6.7.3.1 virtual bool ClientInterface::answerAlive (const long *aliveID*) [private], [pure virtual]

Beantwortet ein Alive-Paket.

Parameter

in	*aliveID*	eindeutige ID, die vom Server gesendet wurde um die Antwort der Anfrage zuzuordnen

Rückgabe

true wenn die Antwort gesendet wurde

Implementiert in Client.

6.7.3.2 virtual bool ClientInterface::requestAlive (void) [pure virtual]

Sendet ein Alive-Anfragepaket.

Rückgabe

Die Antwort entspricht dem Wert von myAliveToggle, vor dem Funktionsaufruf

Wenn diese Funktion true zurückgibt, so bedeutet dies dass der Server seit dem letzten Aufruf dieser Funktion auf das Alive-Paket geantwortet hat.

Implementiert in Client.

6.7.3.3 virtual bool ClientInterface::requestAxisPos (const Gamer *gamer*) [pure virtual]

Fragt die Spielstangenpositionen beim Server an.

Parameter

in	*gamer*	HUMAN, COMPUTER oder BOTH, je nachdem für wen die Spielstangenpositionen abgefragt werden sollen

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert in Client.

6.7.3.4 virtual bool ClientInterface::requestBallPos ( ) [pure virtual]

Fragt die Ballposition beim Server an.

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert in Client.

6.7.3.5 virtual bool ClientInterface::requestScore ( ) [pure virtual]

Fragt die Spielstangenpositionen beim Server an.

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert in Client.

6.7.3.6 virtual bool ClientInterface::sendAxisPos (const float *axisPos[2][4],* const float *axisAngle[2][4]*) [pure virtual]

Sendet die Spielstangenpositionen an den Server.

Parameter

in	*axisPos*	Array mit der Translation (Wert und Geschwindigkeit)
in	*axisAngle*	Array mit der Rotation (Wert und Geschwindigkeit)

Rückgabe

true wenn die Spielstangenpositionen gesendet wurden

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]).

Implementiert in Client.

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

Fragt die Spielstangenpositionen beim Server an.

Parameter

in	*computer*	Spielstand des Computers
in	*human*	Spielstand des menschlichen Spielers

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert in Client.

6.7.3.8 virtual bool ClientInterface::start (ClientUserInterface *const *user*) [pure virtual]

Startet den Client.

Parameter

in	*user*	Zeiger auf den Benutzer des Clients

Rückgabe

true wenn alles beim Starten des Clients geklappt hat

Implementiert in Client.

6.7.3.9 virtual bool ClientInterface::subscribeAxisPos (const Gamer *gamer,* const bool *subs* = true) [pure virtual]

Abonniert die Spielstangenpositionen beim Server.

Parameter

in	*subs*	true zum Abonnieren, false zum Kündigen auf false um das Abonnement zu kündigen
in	*gamer*	HUMAN, COMPUTER oder BOTH, je nachdem für wen die Spielstangenpositionen abonniert werden sollen

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert in Client.

6.7.3.10 virtual bool ClientInterface::subscribeBallPos (const bool *subs* = true) [pure virtual]

Abonniert die Ballposition beim Server.

Parameter

in	*subs*	true zum Abonnieren, false zum Kündigen

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert in Client.

6.7.3.11 virtual bool ClientInterface::subscribeScore (const bool *subs* = true) [pure virtual]

Abonniert den Spielstand beim Server.

Parameter

in	*subs*	true zum Abonnieren, false zum Kündigen auf false um das Abonnement zu kündigen

Rückgabe

true wenn die Anfrage gesendet wurde

Implementiert in Client.

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

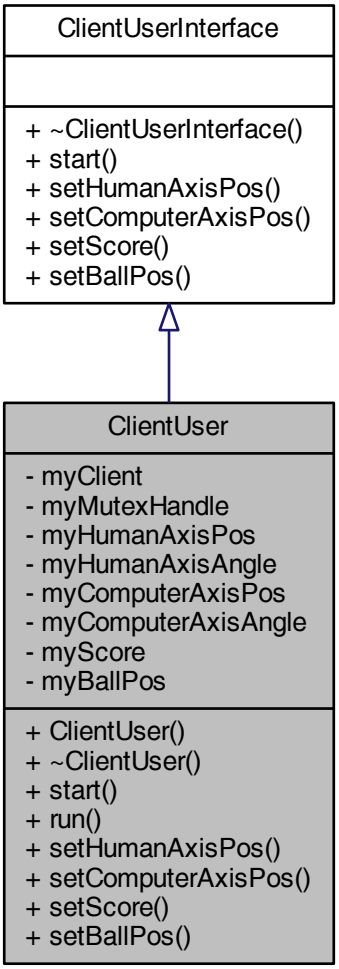
- clientinterface.h

6.8 ClientUser Klassenreferenz

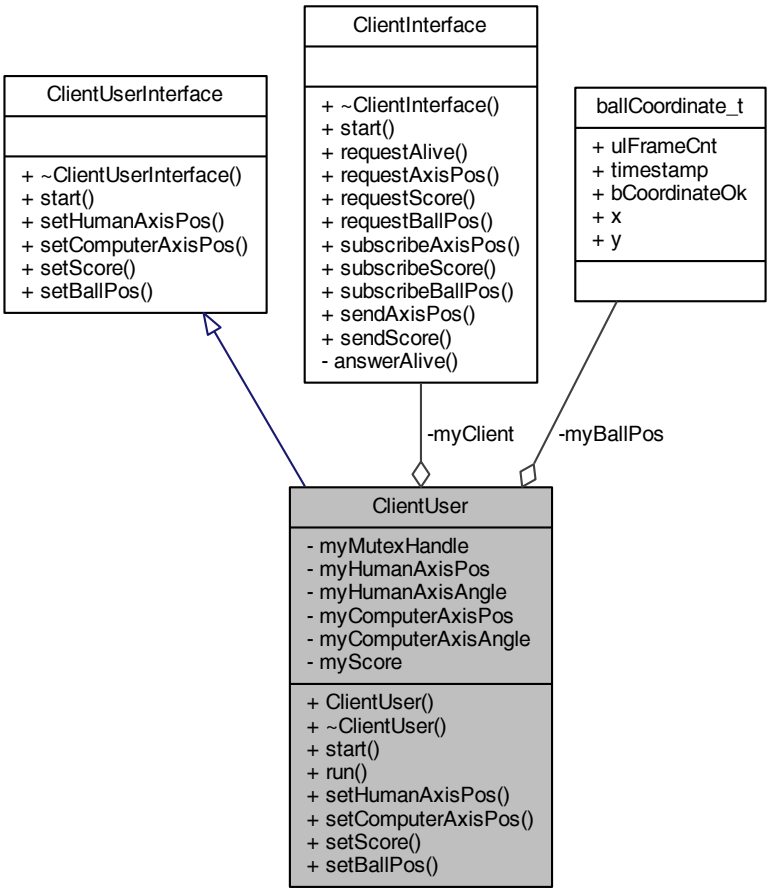
Eine Klasse, die den UDP-Client anlegt und benutzt.

`#include <clientuser.h>`

Klassendiagramm für ClientUser:



Zusammengehörigkeiten von ClientUser:



Öffentliche Methoden

- ClientUser ()

  Konstruktor.
- ~ClientUser ()

  Destruktor.
- bool start ()

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

  Hier findet der Programmablauf stadt.
- bool setHumanAxisPos (const float axisPos[4], const float axisAngle[4])

  setzt die Spielstangenposition des Menschen
- bool setComputerAxisPos (const float axisPos[4], const float axisAngle[4])

  setzt die Spielstangenposition des Computers
- bool setScore (const unsigned short human, const unsigned short computer)

  setzt den Spielstand
- bool setBallPos (const ballCoordinate_t ballPos)

  setzt die Ballposition

Private Attribute

- ClientInterface * myClient

  Zeiger auf den Client.
- 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.

6.8.1 Ausführliche Beschreibung

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

Beispielimplementierung wie der UDP-Client zu verwenden ist.

Definiert in Zeile 26 der Datei clientuser.h.

6.8.2 Beschreibung der Konstruktoren und Destruktoren

6.8.2.1 ClientUser::ClientUser ( )

Konstruktor.

Legt die Klasse Client an.

Definiert in Zeile 16 der Datei clientuser.cpp.

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

6.8.2.2 ClientUser::~ClientUser ( )

Destruktor.

Löscht die Klasse Client.

Definiert in Zeile 45 der Datei clientuser.cpp.

Benutzt NULL.

6.8.3 Dokumentation der Elementfunktionen

6.8.3.1 bool ClientUser::run ( )

Hier findet der Programmablauf stadt.

Rückgabe

true wenn der Ablauf ohne Fehler beendet wurde

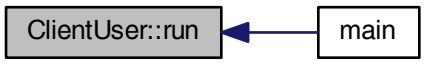
Noch zu erledigen Konsolenausgabe in Funktionen auslagern.

Definiert in Zeile 61 der Datei clientuser.cpp.

Benutzt ANGLE, AXIS_FOUR, AXIS_ONE, AXIS_THREE, AXIS_TWO, BOTH, COMPUTER, HUMAN, POSITION, VALUE und VELOCITY.

Wird benutzt von main().

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



6.8.3.2 bool ClientUser::setBallPos (const ballCoordinate_t *ballPos*) [virtual]

setzt die Ballposition

Parameter

in	*ballPos*	Ballposition die vom Client empfangen wurde

Rückgabe

true wenn die Ballposition erfolgreich übernommen wurde

Wird vom Client aufgerufen wenn eine Ballposition vom Server eingetroffen ist.

Implementiert ClientUserInterface.

Definiert in Zeile 458 der Datei clientuser.cpp.

Benutzt AXIS_FOUR und AXIS_ONE.

6.8.3.3 bool ClientUser::setComputerAxisPos (const float *axisPos[4]*, const float *axisAngle[4]*) [virtual]

setzt die Spielstangenposition des Computers

Parameter

in	*axisPos*	Translation die vom Client empfangen wurde
in	*axisAngle*	Rotation die vom Client empfangen wurde

Rückgabe

true wenn die Spielstangenpositionen erfolgreich übernommen wurden

Wird vom Client aufgerufen wenn eine Spielstangenposition vom Server eingetroffen ist.

Implementiert ClientUserInterface.

Definiert in Zeile 391 der Datei clientuser.cpp.

Benutzt AXIS_FOUR, AXIS_ONE und VALUE.

6.8.3.4 bool ClientUser::setHumanAxisPos (const float *axisPos[4],* const float *axisAngle[4]*) [virtual]

setzt die Spielstangenposition des Menschen

Parameter

in	*axisPos*	Translation die vom Client empfangen wurde
in	*axisAngle*	Rotation die vom Client empfangen wurde

Rückgabe

true wenn die Spielstangenpositionen erfolgreich übernommen wurden

Wird vom Client aufgerufen wenn eine Spielstangenposition vom Server eingetroffen ist.

Implementiert ClientUserInterface.

Definiert in Zeile 357 der Datei clientuser.cpp.

Benutzt AXIS_FOUR und AXIS_ONE.

6.8.3.5 bool ClientUser::setScore (const unsigned short *human,* const unsigned short *computer*) [virtual]

setzt den Spielstand

Parameter

in	*human*	Spielstand des Menschen
in	*computer*	Spielstand des Computers

Rückgabe

true wenn der Spielstand erfolgreich übernommen wurde

Wird vom Client aufgerufen wenn ein Spielstand vom Server eingetroffen ist.

Implementiert ClientUserInterface.

Definiert in Zeile 425 der Datei clientuser.cpp.

Benutzt AXIS_FOUR, AXIS_ONE, COMPUTER und HUMAN.

6.8.3.6 bool ClientUser::start ( ) [virtual]

Startet den die Benutzung des Clients.

Rückgabe

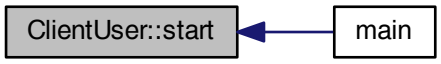
true wenn alles beim Starten des Clients geklappt hat

Implementiert ClientUserInterface.

Definiert in Zeile 53 der Datei clientuser.cpp.

Wird benutzt von main().

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



6.8.4 Dokumentation der Datenelemente

6.8.4.1 ballCoordinate_t ClientUser::myBallPos [private]

Ballposition.

Definiert in Zeile 159 der Datei clientuser.h.

6.8.4.2 ClientInterface* ClientUser::myClient [private]

Zeiger auf den Client.

Definiert in Zeile 105 der Datei clientuser.h.

6.8.4.3 float ClientUser::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 147 der Datei clientuser.h.

6.8.4.4 float ClientUser::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 138 der Datei clientuser.h.

6.8.4.5 float ClientUser::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 129 der Datei clientuser.h.

6.8.4.6 float ClientUser::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 121 der Datei clientuser.h.

6.8.4.7 HANDLE ClientUser::myMutexHandle [private]

Mutex-Handle.

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

Definiert in Zeile 113 der Datei clientuser.h.

6.8.4.8 unsigned ClientUser::myScore[2] [private]

Spielstand.

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

Definiert in Zeile 154 der Datei clientuser.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

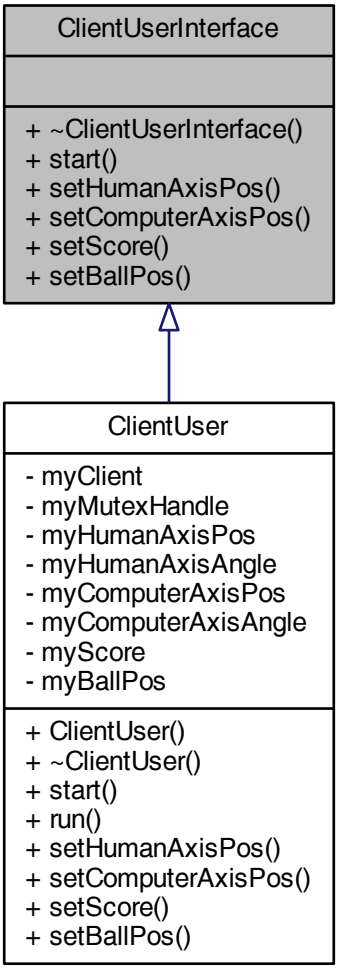
- clientuser.h
- clientuser.cpp

6.9 ClientUserInterface Schnittstellenreferenz

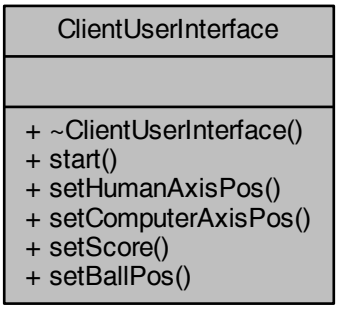
Interface zur Kommunikation mit dem Benutzer des UDP-Clients.

`#include <clientuserinterface.h>`

Klassendiagramm für ClientUserInterface:



Zusammengehörigkeiten von ClientUserInterface:



Öffentliche Methoden

- virtual ~ClientUserInterface (void)

  Virtueller Destruktor.
- virtual bool start ()=0

  Startet den die Benutzung des Clients.
- virtual bool setHumanAxisPos (const float axisPos[4], const float axisAngle[4])=0

  setzt die Spielstangenposition des Menschen
- virtual bool setComputerAxisPos (const float axisPos[4], const float axisAngle[4])=0

  setzt die Spielstangenposition des Computers
- virtual bool setScore (const unsigned short human, const unsigned short computer)=0

  setzt den Spielstand
- virtual bool setBallPos (const ballCoordinate_t ballPos)=0

  setzt die Ballposition

6.9.1 Ausführliche Beschreibung

Interface zur Kommunikation mit dem Benutzer des UDP-Clients.

Definiert in Zeile 19 der Datei clientuserinterface.h.

6.9.2 Beschreibung der Konstruktoren und Destruktoren

6.9.2.1 virtual ClientUserInterface::~ClientUserInterface (void) [inline], [virtual]

Virtueller Destruktor.

Definiert in Zeile 25 der Datei clientuserinterface.h.

6.9.3 Dokumentation der Elementfunktionen

6.9.3.1 virtual bool ClientUserInterface::setBallPos (const ballCoordinate_t *ballPos*) [pure virtual]

setzt die Ballposition

Parameter

in	*ballPos*	Ballposition die vom Client empfangen wurde

Rückgabe

true wenn die Ballposition erfolgreich übernommen wurde

Wird vom Client aufgerufen wenn eine Ballposition vom Server eingetroffen ist.

Implementiert in ClientUser.

6.9.3.2 virtual bool ClientUserInterface::setComputerAxisPos (const float *axisPos[4]*, const float *axisAngle[4]*) [pure virtual]

setzt die Spielstangenposition des Computers

Parameter

in	*axisPos*	Translation die vom Client empfangen wurde
in	*axisAngle*	Rotation die vom Client empfangen wurde

Rückgabe

true wenn die Spielstangenpositionen erfolgreich übernommen wurden

Wird vom Client aufgerufen wenn eine Spielstangenposition vom Server eingetroffen ist.

Implementiert in ClientUser.

6.9.3.3 virtual bool ClientUserInterface::setHumanAxisPos (const float *axisPos[4]*, const float *axisAngle[4]*) [pure virtual]

setzt die Spielstangenposition des Menschen

Parameter

in	*axisPos*	Translation die vom Client empfangen wurde
in	*axisAngle*	Rotation die vom Client empfangen wurde

Rückgabe

true wenn die Spielstangenpositionen erfolgreich übernommen wurden

Wird vom Client aufgerufen wenn eine Spielstangenposition vom Server eingetroffen ist.

Implementiert in ClientUser.

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

setzt den Spielstand

Parameter

in	*human*	Spielstand des Menschen
in	*computer*	Spielstand des Computers

Rückgabe

true wenn der Spielstand erfolgreich übernommen wurde

Wird vom Client aufgerufen wenn ein Spielstand vom Server eingetroffen ist.

Implementiert in ClientUser.

6.9.3.5 virtual bool ClientUserInterface::start ( ) [pure virtual]

Startet den die Benutzung des Clients.

Rückgabe

true wenn alles beim Starten des Clients geklappt hat

Implementiert in ClientUser.

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

- clientuserinterface.h

6.10 float_1 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von float_1:

Öffentliche Attribute

- unsigned int id: 16
- float flt

6.10.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein float-Wert.

Definiert in Zeile 295 der Datei defines.h.

6.10.2 Dokumentation der Datenelemente

6.10.2.1 float float_1::flt

Definiert in Zeile 298 der Datei defines.h.

6.10.2.2 unsigned int float_1::id

Definiert in Zeile 297 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

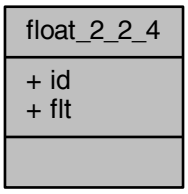
- defines.h

6.11 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]

6.11.1 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.

6.11.2 Dokumentation der Datenelemente

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

Definiert in Zeile 331 der Datei defines.h.

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

6.11.2.2 unsigned int float_2_2_4::id

Definiert in Zeile 330 der Datei defines.h.

Wird benutzt von Client::sendAxisPos().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

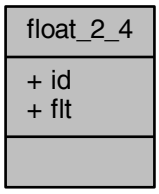
- defines.h

6.12 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]

6.12.1 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.

6.12.2 Dokumentation der Datenelemente

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

Definiert in Zeile 320 der Datei defines.h.

Wird benutzt von Client::update().

6.12.2.2 unsigned int float_2_4::id

Definiert in Zeile 319 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- defines.h

6.13 float_4 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von float_4:

Öffentliche Attribute

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

6.13.1 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.

6.13.2 Dokumentation der Datenelemente

6.13.2.1 float float_4::flt[4]

Definiert in Zeile 309 der Datei defines.h.

6.13.2.2 unsigned int float_4::id

Definiert in Zeile 308 der Datei defines.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

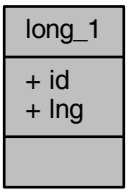
- defines.h

6.14 long_1 Strukturreferenz

Anfrage-Paket für bestimmte Daten.

`#include <defines.h>`

Zusammengehörigkeiten von long_1:



Öffentliche Attribute

- unsigned int id: 16
- long lng

6.14.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein long-Wert.

Definiert in Zeile 251 der Datei defines.h.

6.14.2 Dokumentation der Datenelemente

6.14.2.1 unsigned int long_1::id

Definiert in Zeile 253 der Datei defines.h.

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

6.14.2.2 long long_1::lng

Definiert in Zeile 254 der Datei defines.h.

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

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

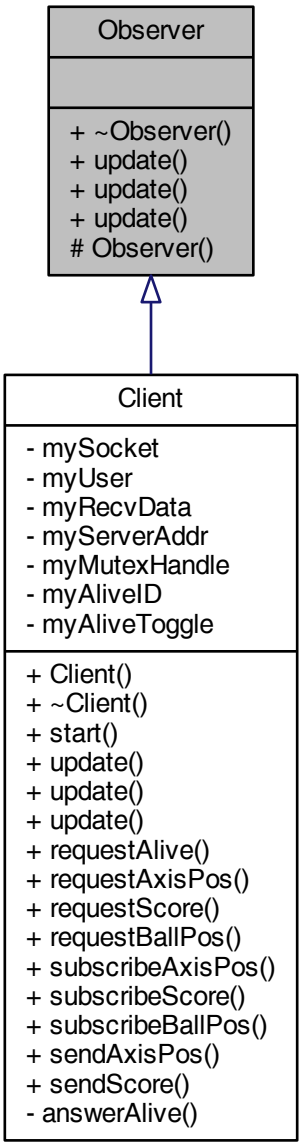
- defines.h

6.15 Observer Klassenreferenz

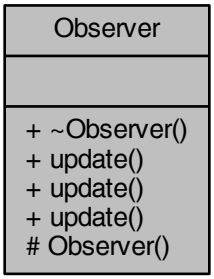
Observer Klasse für das Observer Entwurfsmodell.

`#include <observer.h>`

Klassendiagramm für Observer:



Zusammengehörigkeiten von Observer:



Öffentliche Methoden

- virtual ~Observer ()

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

  update-Funktion der Observer Klasse
- 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.

6.15.1 Ausführliche Beschreibung

Observer Klasse für das Observer Entwurfsmodell.

Siehe auch

Publish

Definiert in Zeile 20 der Datei observer.h.

6.15.2 Beschreibung der Konstruktoren und Destruktoren

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

Destruktor.

Definiert in Zeile 26 der Datei observer.h.

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

Konstruktor.

Definiert in Zeile 59 der Datei observer.h.

6.15.3 Dokumentation der Elementfunktionen

6.15.3.1 virtual void Observer::update (Publish *const *pub) [pure 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 in Client.

Wird benutzt von Publish::notify().

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

6.15.3.2 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 Client.

6.15.3.3 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 Client.

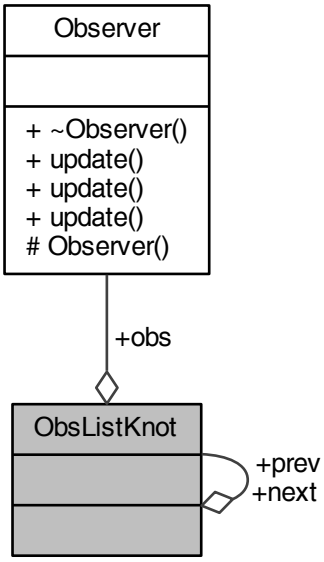
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- observer.h

6.16 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.

6.16.1 Ausführliche Beschreibung

Definiert in Zeile 24 der Datei publish.h.

6.16.2 Dokumentation der Datenelemente

6.16.2.1 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().

6.16.2.2 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().

6.16.2.3 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:

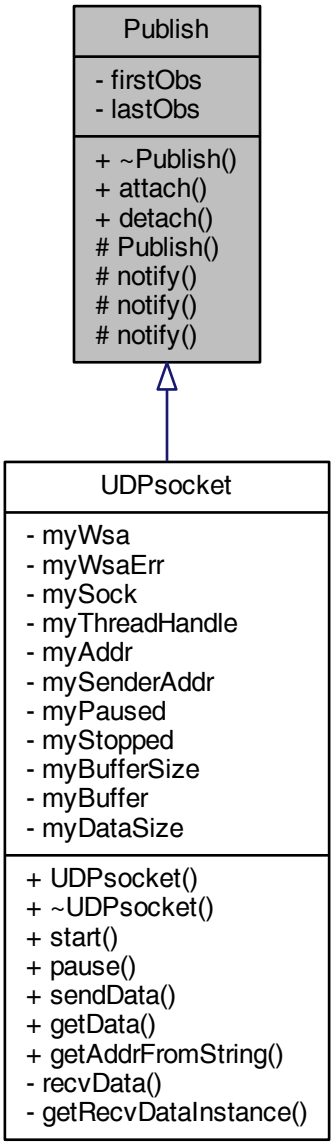
- publish.h

6.17 Publish Klassenreferenz

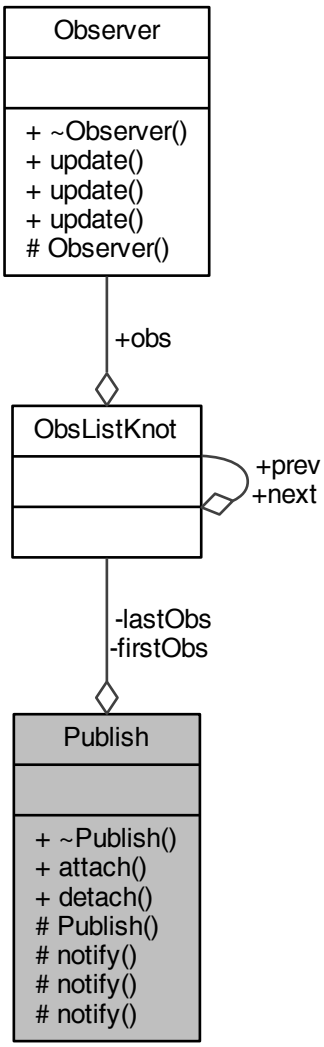
Publisher Klasse für das Observer Entwurfsmodell.

`#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.

6.17.1 Ausführliche Beschreibung

Publisher Klasse für das Observer Entwurfsmodell.

Siehe auch

Observer

Definiert in Zeile 39 der Datei publish.h.

6.17.2 Beschreibung der Konstruktoren und Destruktoren

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

Destruktor.

Definiert in Zeile 45 der Datei publish.h.

6.17.2.2 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.

6.17.3 Dokumentation der Elementfunktionen

6.17.3.1 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.

6.17.3.2 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.

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

Benachichtigt alle eingetragenen Observer Objekte.

Siehe auch

Observer::update()

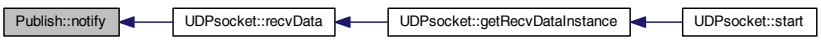
Definiert in Zeile 139 der Datei publish.h.

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:



6.17.3.4 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:

6.17.3.5 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:

6.17.4 Dokumentation der Datenelemente

6.17.4.1 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().

6.17.4.2 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:

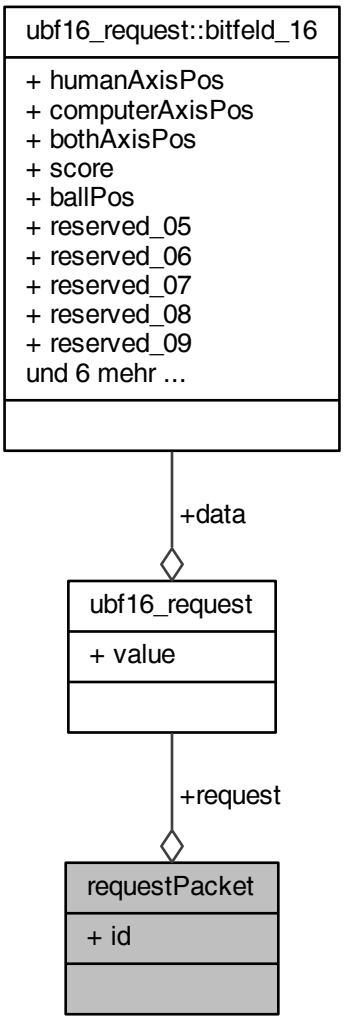
- publish.h

6.18 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.

6.18.1 Ausführliche Beschreibung

Anfrage-Paket für bestimmte Daten.

Definiert in Zeile 215 der Datei defines.h.

6.18.2 Dokumentation der Datenelemente

6.18.2.1 unsigned int requestPacket::id

Paket-ID.

Definiert in Zeile 218 der Datei defines.h.

Wird benutzt von Client::requestAxisPos(), Client::requestBallPos() und Client::requestScore().

6.18.2.2 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 Client::requestAxisPos(), Client::requestBallPos() und Client::requestScore().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

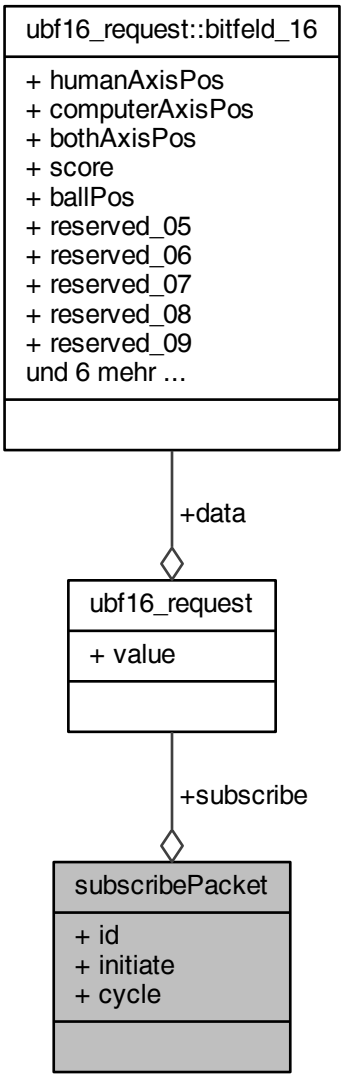
- defines.h

6.19 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!

6.19.1 Ausführliche Beschreibung

Abonnieren-Paket für bestimmte Daten.

Definiert in Zeile 227 der Datei defines.h.

6.19.2 Dokumentation der Datenelemente

6.19.2.1 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.

6.19.2.2 unsigned int subscribePacket::id

Paket-ID.

Definiert in Zeile 230 der Datei defines.h.

Wird benutzt von Client::subscribeAxisPos(), Client::subscribeBallPos() und Client::subscribeScore().

6.19.2.3 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 Client::subscribeAxisPos(), Client::subscribeBallPos() und Client::subscribeScore().

6.19.2.4 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 Client::subscribeAxisPos(), Client::subscribeBallPos() und Client::subscribeScore().

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

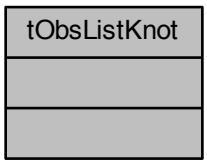
- defines.h

6.20 tObsListKnot Strukturreferenz

Struktur für ein Listenelement.

`#include <publish.h>`

Zusammengehörigkeiten von tObsListKnot:



6.20.1 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:

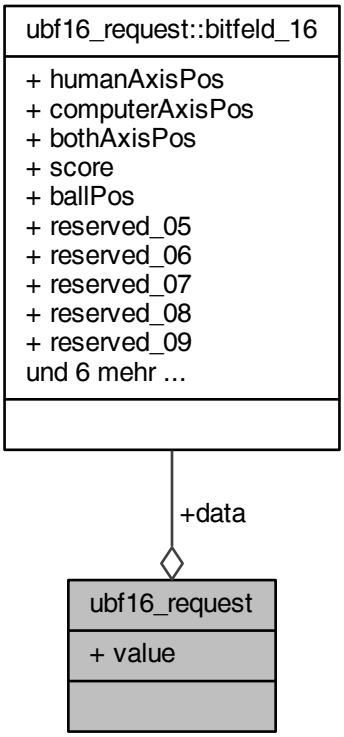
- publish.h

6.21 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

6.21.1 Ausführliche Beschreibung

Bitfeld, bei dem jedes Bit bestimmte Daten anfordert.

Definiert in Zeile 188 der Datei defines.h.

6.21.2 Dokumentation der Datenelemente

6.21.2.1 struct ubf16_request::bitfeld_16 ubf16_request::data

Wird benutzt von Client::requestAxisPos(), Client::requestBallPos(), Client::requestScore(), Client::subscribeAxisPos(), Client::subscribeBallPos() und Client::subscribeScore().

6.21.2.2 unsigned int ubf16_request::value

Definiert in Zeile 190 der Datei defines.h.

Wird benutzt von Client::requestAxisPos(), Client::requestBallPos(), Client::requestScore(), Client::subscribeAxisPos(), Client::subscribeBallPos() und Client::subscribeScore().

Die Dokumentation für diese Variante wurde erzeugt aufgrund der Datei:

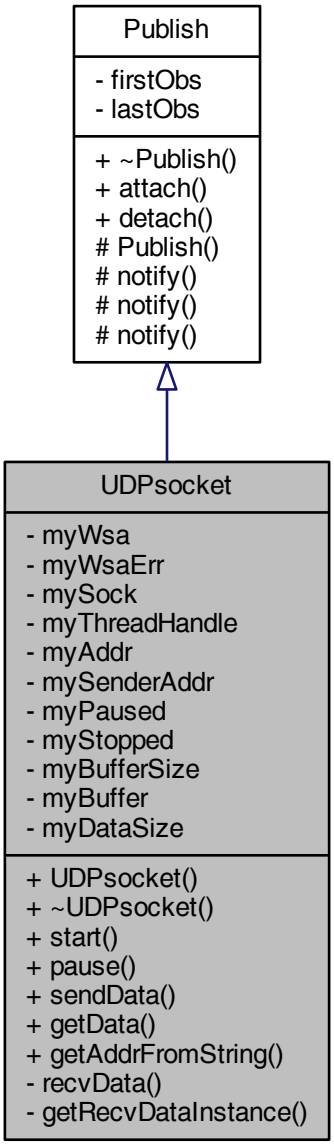
- defines.h

6.22 UDPsocket Klassenreferenz

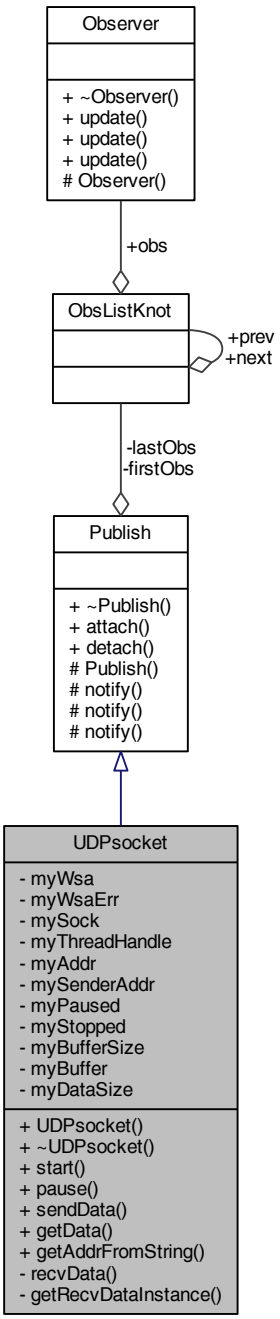
Stellt ein UDP-Socket dar.

`#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

6.22.1 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.

6.22.2 Beschreibung der Konstruktoren und Destruktoren

6.22.2.1 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.

6.22.2.2 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.

6.22.3 Dokumentation der Elementfunktionen

6.22.3.1 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.

6.22.3.2 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.

6.22.3.3 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

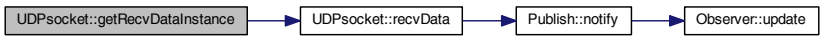
Trick um einen neuen Thread mit einer Memberfunktion zu erzeugen.

Definiert in Zeile 244 der Datei udpsocket.cpp.

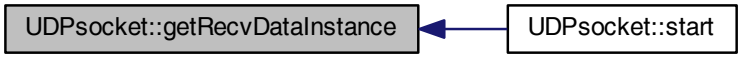
Benutzt NULL und recvData().

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:



6.22.3.4 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.

6.22.3.5 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

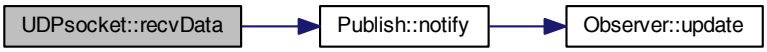
Publish::notify()

Definiert in Zeile 202 der Datei udpsocket.cpp.

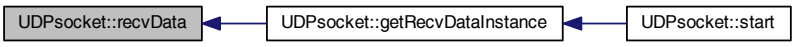
Benutzt myBuffer, myBufferSize, myDataSize, myPaused, mySenderAddr, mySock, myStopped, Publish::notify() und NULL.

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:



6.22.3.6 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.

6.22.3.7 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:

6.22.4 Dokumentation der Datenelemente

6.22.4.1 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().

6.22.4.2 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().

6.22.4.3 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().

6.22.4.4 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().

6.22.4.5 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().

6.22.4.6 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().

6.22.4.7 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().

6.22.4.8 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().

6.22.4.9 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().

6.22.4.10 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().

6.22.4.11 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:

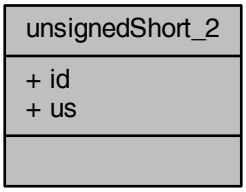
- udpsocket.h
- udpsocket.cpp

6.23 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]

6.23.1 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.

6.23.2 Dokumentation der Datenelemente

6.23.2.1 unsigned int unsignedShort_2::id

Definiert in Zeile 286 der Datei defines.h.

Wird benutzt von Client::sendScore().

6.23.2.2 unsigned short unsignedShort_2::us[2]

Definiert in Zeile 287 der Datei defines.h.

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

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- defines.h

Kapitel 7

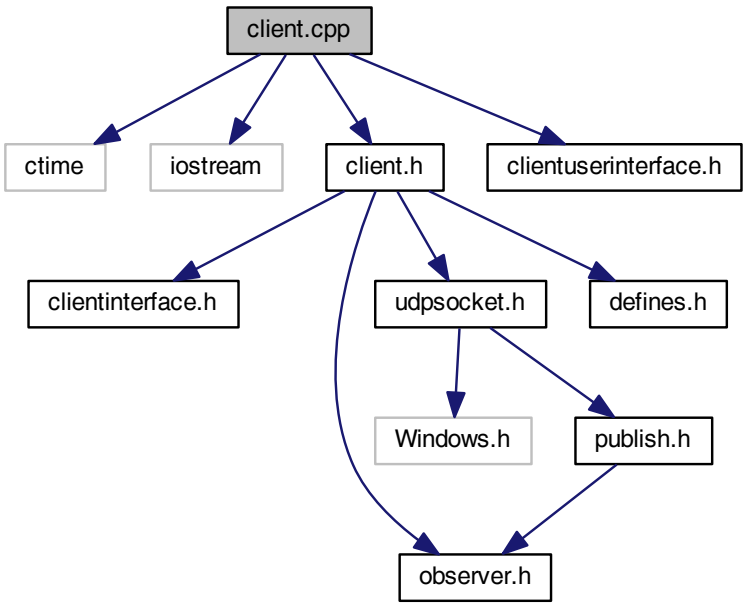
Datei-Dokumentation

7.1 client.cpp-Dateireferenz

Enthält die Implementierung der Klasse Client.

```
#include <ctime>
#include <iostream>
#include "client.h"
#include "clientuserinterface.h"
```

Include-Abhängigkeitsdiagramm für client.cpp:



7.1.1 Ausführliche Beschreibung

Enthält die Implementierung der Klasse Client.

Autor

Scharel Clemens

Datum

07.12.2012

Version

0.1 Testphase

Definiert in Datei client.cpp.

7.2 client.cpp

```
00001
00010 #include <ctime>
00011 #include <iostream>
00012 using namespace std;
00013
00014 #include "client.h"
00015 #include "clientuserinterface.h"
00016
00017 Client::Client(const unsigned short localPort, const string serverHost, const unsigned short
      serverPort)
00018 {
00019     myAliveToggle = false;
00020
00021     myMutexHandle = CreateMutex(NULL, FALSE, NULL);
00022     if (!myMutexHandle)
00023         cout << "Fehler beim erzeugen des Mutex: " << GetLastError() << endl;
00024
00025     mySocket = new UDPsocket(RECV_BUFFER_SIZE, localPort);
00026
00027     //  Server-Socket-Adresse umwandeln
00028     myServerAddr.sin_family = AF_INET;
00029     myServerAddr.sin_port = htons(serverPort);
00030     if (!mySocket->getAddrFromString(serverHost, myServerAddr))
00031     {
00032         cout << "Server-Adresse konnte nicht aufgeloest werden: " << serverHost.data() << endl;
00033         delete mySocket;
00034         mySocket = NULL;
00035     }
00036 }
00037
00038 Client::~Client()
00039 {
00040     if (mySocket)
00041     {
00042         mySocket->detach(this);
00043         delete mySocket;
00044     }
00045     mySocket = NULL;
00046     myUser = NULL;
00047 }
00048
00049 bool Client::start(ClientUserInterface* const user)
00050 {
00051     bool retval = false;
00052
00053     if (mySocket)
00054     {
00055         myUser = user;
00056         mySocket->attach(this);
00057         if (mySocket->start())
00058         {
00059             //cout << "Client gestartet!" << endl;
00060             retval = true;
00061         }
00062     }
00063     return retval;
00064 }
00065
00066 void Client::update(Publish* const pub)
00067 {
00068     if (pub == mySocket)
```

```
00069     {
00070         //cout << "Antwort empfangen: ";
00071         int iResult;
00072         unsigned int iID;
00073         sockaddr_in serverAddr;
00074 
00075         //  Daten vom Socket abholen
00076         iResult = mySocket->getData(myRecvData, serverAddr);
00077         //  Das Paket muss mindestens 2 Bytes (16 bit) groß sein
00078         if (iResult < 2)
00079             cout << "Ungueltiges Paket empfangen!" << endl;
00080         else
00081         {
00082             iID = *myRecvData;
00083             //  Hier müssen alle Paket-ID'S abgefragt werden die empfangen werden sollen
00084             switch (iID)
00085             {
00086             case ALIVE_REQUEST:
00087                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00088                 if (iResult == sizeof(long_1))
00089                 {
00090                     //  Zeiger auf die Struktur anlegen
00091                     long_1* alive;
00092                     //  Zeifer auf die Struktur verwenden um die Daten aus dem Empfangsspeicher zu lesen
00093                     alive = (long_1*)myRecvData;
00094                     //  Die Alive-Anfrage, mit der Alive-ID beantworten
00095                     answerAlive(alive->lng);
00096                 }
00097                 break;
00098 
00099             case ALIVE_ANSWER:
00100                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00101                 if (iResult == sizeof(long_1))
00102                 {
00103                     //  Zeiger auf die Struktur anlegen
00104                     long_1* alive;
00105                     //  Zeifer auf die Struktur verwenden um die Daten aus dem Empfangsspeicher zu lesen
00106                     alive = (long_1*)myRecvData;
00107                     //  Die Alive-Antwort auswerten
00108                     DWORD waitResult = WaitForSingleObject(myMutexHandle, 10);
00109                     if (waitResult == WAIT_OBJECT_0)
00110                     {
00111                         if (alive->lng == myAliveID)
00112                             myAliveToggle = true;
00113 
00114                         if (!ReleaseMutex(myMutexHandle))
00115                             cout << "Mutex konnte nicht wieder freigegeben werden! (requestAlive())" <<
    endl;
00116                     }
00117                 }
00118                 break;
00119 
00120             case GET_HUMAN_AXISPOS:
00121                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00122                 if (iResult == sizeof(float_2_4))
00123                 {
00124                     //  Zeiger auf die Struktur anlegen
00125                     float_2_4* axisPos;
00126                     //  Zeifer auf die Struktur verwenden um die Daten aus dem Empfangsspeicher zu lesen
00127                     axisPos = (float_2_4*)myRecvData;
00128                     //  Dem Client-Benutzer die empfangenen Daten übergeben
00129                     myUser->setHumanAxisPos(axisPos->flt[POSITION], axisPos->
    flt[ANGLE]);
00130                 }
00131                 break;
00132 
00133             case GET_COMPUTER_AXISPOS:
00134                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00135                 if (iResult == sizeof(float_2_4))
00136                 {
00137                     //  Zeiger auf die Struktur anlegen
00138                     float_2_4* axisPos;
00139                     //  Zeifer auf die Struktur verwenden um die Daten aus dem Empfangsspeicher zu lesen
00140                     axisPos = (float_2_4*)myRecvData;
00141                     //  Dem Client-Benutzer die empfangenen Daten übergeben
00142                     myUser->setComputerAxisPos(axisPos->flt[POSITION], axisPos->
    flt[ANGLE]);
00143                 }
00144                 break;
00145 
00146             case GET_BOTH_AXISPOS:
00147                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00148                 if (iResult == sizeof(float_2_2_4))
00149                 {
00150                     //  Zeiger auf die Struktur anlegen
00151                     float_2_2_4* axisPos;
00152                     //  Zeifer auf die Struktur verwenden um die Daten aus dem Empfangsspeicher zu lesen
```

```
00153                     axisPos = (float_2_2_4*)myRecvData;
00154                     //  Dem Client-Benutzer die empfangenen Daten übergeben
00155                     myUser->setComputerAxisPos(axisPos->flt[COMPUTER][
    POSITION], axisPos->flt[COMPUTER][ANGLE]);
00156                     myUser->setHumanAxisPos(axisPos->flt[HUMAN][POSITION], axisPos->
    flt[HUMAN][ANGLE]);
00157                 }
00158                 break;
00159
00160             case GET_SCORE:
00161                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00162                 if (iResult == sizeof(unsignedShort_2))
00163                 {
00164                     //  Zeiger auf die Struktur anlegen
00165                     unsignedShort_2* score;
00166                     //  Zeifer auf die Struktur verwenden um die Daten aus dem Empfangsspeicher zu lesen
00167                     score = (unsignedShort_2*)myRecvData;
00168                     //  Dem Client-Benutzer die empfangenen Daten übergeben
00169                     myUser->setScore(score->us[HUMAN], score->us[
    COMPUTER]);
00170                 }
00171                 break;
00172
00173             case GET_BALLPOS:
00174                 //  Überprüfen ob die Menge der empfangenen Daten der erwarteten Struktur entspricht
00175                 if (iResult == sizeof(ballPacket))
00176                 {
00177                     //  Zeiger auf die Struktur anlegen
00178                     ballPacket* ballPos;
00179                     //  Zeifer auf die Struktur verwenden um die Daten aus dem Empfangsspeicher zu lesen
00180                     ballPos = (ballPacket*)myRecvData;
00181                     //  Dem Client-Benutzer die empfangenen Daten übergeben
00182                     myUser->setBallPos(ballPos->ballPos);
00183                 }
00184                 break;
00185             default:
00186                 cout << "Unbekanntes Paket empfangen!" << endl;
00187                 break;
00188             }
00189         }
00190     }
00191 }
00192
00193 void Client::update(Publish* const pub, const int* const iValues, const unsigned int
    iValuesSize)
00194 {
00195     update(pub);
00196 }
00197
00198 void Client::update(Publish* const pub, const float* const fValues, const unsigned int
     fValuesSize, const int* const iValues, const unsigned int iValuesSize)
00199 {
00200     update(pub);
00201 }
00202
00203 /*****************************************************************************
00204 *   Hier müssen die virtuellen Memberfunktionen der Basisklasse               *
00205 *   ClientInterface implementiert werden.                                     *
00206 *   z.B.: zum Anfragen/Abonnieren von bestimmten Daten, ...                   *
00207 *****************************************************************************/
00208
00209 bool Client::requestAlive(void)
00210 {
00211     bool retval = false;
00212     DWORD waitResult;
00213
00214     // Anfrage-Paket erstellen
00215     long_1 request;
00216     request.id = ALIVE_REQUEST;
00217
00218     waitResult = WaitForSingleObject(myMutexHandle, 10);
00219     if (waitResult == WAIT_OBJECT_0)
00220     {
00221         myAliveID = clock();
00222         retval = myAliveToggle;
00223         myAliveToggle = false;
00224
00225         if (!ReleaseMutex(myMutexHandle))
00226             cout << "Mutex konnte nicht wieder freigegeben werden! (requestAlive())" << endl;
00227     }
00228
00229     request.lng = myAliveID;
00230
00231     //  Anfrage-Paket versenden
00232     mySocket->sendData((const char*) &request, sizeof(long_1), myServerAddr);
00233
00234     return retval;
```

```
00235 }
00236
00237 bool Client::answerAlive(const long aliveID)
00238 {
00239     bool retval = false;
00240     int bytesSend = 0;
00241
00242     // Anfrage-Paket erstellen
00243     long_1 request;
00244     request.id = ALIVE_ANSWER;
00245     request.lng = aliveID;
00246
00247     //  Anfrage-Paket versenden
00248     bytesSend = mySocket->sendData((const char*) &request, sizeof(long_1), myServerAddr);
00249
00250     //  Überprüfen ob alle Daten gesendet wurden
00251     if (bytesSend == sizeof(long_1))
00252         retval = true;
00253     return retval;
00254 }
00255
00256 bool Client::requestAxisPos(const Gamer gamer)
00257 {
00258     bool retval = false;
00259     int bytesSend = 0;
00260
00261     //  Anfrage-Paket erstellen
00262     requestPacket request;
00263     request.id = REQUEST;
00264     request.request.value = 0;
00265     switch (gamer)
00266     {
00267     case HUMAN:
00268         request.request.data.humanAxisPos = 1;
00269         break;
00270     case COMPUTER:
00271         request.request.data.computerAxisPos = 1;
00272         break;
00273     case BOTH:
00274         request.request.data.bothAxisPos = 1;
00275         break;
00276     }
00277
00278     //  Anfrage-Paket versenden
00279     bytesSend = mySocket->sendData((const char*) &request, sizeof(requestPacket), myServerAddr
      );
00280
00281     //  Überprüfen ob alle Daten gesendet wurden
00282     if (bytesSend == sizeof(requestPacket))
00283         retval = true;
00284     return retval;
00285 }
00286
00287 bool Client::requestScore()
00288 {
00289     bool retval = false;
00290     int bytesSend = 0;
00291
00292     //  Anfrage-Paket erstellen
00293     requestPacket request;
00294     request.id = REQUEST;
00295     request.request.value = 0;
00296     request.request.data.score = 1;
00297
00298     //  Anfrage-Paket versenden
00299     bytesSend = mySocket->sendData((const char*) &request, sizeof(requestPacket), myServerAddr
      );
00300
00301     //  Überprüfen ob alle Daten gesendet wurden
00302     if (bytesSend == sizeof(requestPacket))
00303         retval = true;
00304     return retval;
00305 }
00306
00307 bool Client::requestBallPos()
00308 {
00309     bool retval = false;
00310     int bytesSend = 0;
00311
00312     //  Anfrage-Paket erstellen
00313     requestPacket request;
00314     request.id = REQUEST;
00315     request.request.value = 0;
00316     request.request.data.ballPos = 1;
00317
00318     //  Anfrage-Paket versenden
00319     bytesSend = mySocket->sendData((const char*) &request, sizeof(requestPacket), myServerAddr
```

```
    );
00320
00321     //  Überprüfen ob alle Daten gesendet wurden
00322     if (bytesSend == sizeof(requestPacket))
00323         retval = true;
00324     return retval;
00325 }
00326
00327 bool Client::subscribeAxisPos(const Gamer gamer, const bool subs)
00328 {
00329     bool retval = false;
00330     int bytesSend = 0;
00331
00332     //  Abonnieren-Paket erstellen
00333     subscribePacket subscribe;
00334     subscribe.id = SUBSCRIBE;
00335     subscribe.initiate = subs;
00336     subscribe.subscribe.value = 0;
00337     switch (gamer)
00338     {
00339     case HUMAN:
00340         subscribe.subscribe.data.humanAxisPos = 1;
00341         break;
00342     case COMPUTER:
00343         subscribe.subscribe.data.computerAxisPos = 1;
00344         break;
00345     case BOTH:
00346         subscribe.subscribe.data.bothAxisPos = 1;
00347         break;
00348     }
00349
00350     //  Abonnieren-Paket versenden
00351     bytesSend = mySocket->sendData((const char*) &subscribe, sizeof(
    subscribePacket), myServerAddr);
00352
00353     //  Überprüfen ob alle Daten gesendet wurden
00354     if (bytesSend == sizeof(subscribe))
00355         retval = true;
00356     return retval;
00357 }
00358
00359 bool Client::subscribeScore(const bool subs)
00360 {
00361     bool retval = false;
00362     int bytesSend = 0;
00363
00364     //  Abonnieren-Paket erstellen
00365     subscribePacket subscribe;
00366     subscribe.id = SUBSCRIBE;
00367     subscribe.initiate = subs;
00368     subscribe.subscribe.value = 0;
00369     subscribe.subscribe.data.score = 1;
00370
00371     //  Abonnieren-Paket versenden
00372     bytesSend = mySocket->sendData((const char*) &subscribe, sizeof(
    subscribePacket), myServerAddr);
00373
00374     //  Überprüfen ob alle Daten gesendet wurden
00375     if (bytesSend == sizeof(subscribe))
00376         retval = true;
00377     return retval;
00378 }
00379
00380 bool Client::subscribeBallPos(const bool subs)
00381 {
00382     bool retval = false;
00383     int bytesSend = 0;
00384
00385     //  Abonnieren-Paket erstellen
00386     subscribePacket subscribe;
00387     subscribe.id = SUBSCRIBE;
00388     subscribe.initiate = subs;
00389     subscribe.subscribe.value = 0;
00390     subscribe.subscribe.data.ballPos = 1;
00391
00392     //  Abonnieren-Paket versenden
00393     bytesSend = mySocket->sendData((const char*) &subscribe, sizeof(
    subscribePacket), myServerAddr);
00394
00395     //  Überprüfen ob alle Daten gesendet wurden
00396     if (bytesSend == sizeof(subscribe))
00397         retval = true;
00398     return retval;
00399 }
00400
00401 bool Client::sendAxisPos(const float axisPos[2][4], const float axisAngle[2][4])
00402 {
```

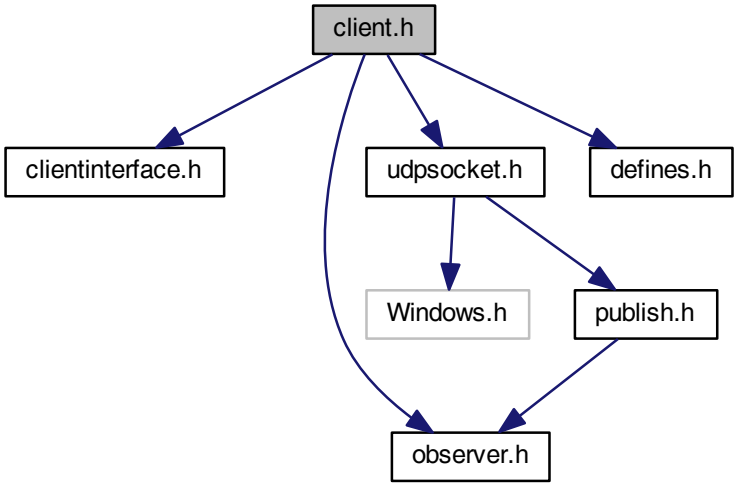
```
00403     bool retval = false;
00404     int bytesSend = 0;
00405
00406     //  Spielstangenpositions-Paket erstellen
00407     float_2_2_4 newAxisPos;
00408     newAxisPos.id = SET_AXISPOS;
00409     for (unsigned int value = VALUE; value <= VELOCITY; value ++)
00410     {
00411         for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00412         {
00413             newAxisPos.flt[POSITION][value][axis] = axisPos[value][axis];
00414             newAxisPos.flt[ANGLE][value][axis] = axisAngle[value][axis];
00415         }
00416     }
00417
00418     //  Spielstangenpositions-Paket versenden
00419     bytesSend = mySocket->sendData((const char*) &newAxisPos, sizeof(float_2_2_4), myServerAddr)
    ;
00420
00421     //  Überprüfen ob alle Daten gesendet wurden
00422     if (bytesSend == sizeof(float_2_2_4))
00423         retval = true;
00424     return retval;
00425 }
00426
00427 bool Client::sendScore(const unsigned short computer, const unsigned short human)
00428 {
00429     bool retval = false;
00430     int bytesSend = 0;
00431
00432     //  Spielstands-Paket erstellen
00433     unsignedShort_2 newScore;
00434     newScore.id = SET_SCORE;
00435     newScore.us[COMPUTER] = computer;
00436     newScore.us[HUMAN] = human;
00437
00438     //  Spielstands-Paket versenden
00439     bytesSend = mySocket->sendData((const char*) &newScore, sizeof(
    unsignedShort_2), myServerAddr);
00440
00441     //  Überprüfen ob alle Daten gesendet wurden
00442     if (bytesSend == sizeof(unsignedShort_2))
00443         retval = true;
00444     return retval;
00445 }
```

7.3 client.h-Dateireferenz

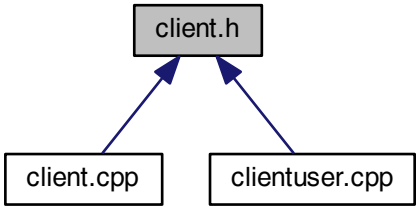
Enthält die Definition der Klasse Client.

```
#include "clientinterface.h"
#include "observer.h"
#include "udpsocket.h"
#include "defines.h"
```

Include-Abhängigkeitsdiagramm für client.h:



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



Klassen

- class Client

  ein UDP-Client

7.3.1 Ausführliche Beschreibung

Enthält die Definition der Klasse Client.

Autor

Scharel Clemens

Datum

07.12.2012

Version

0.1 Testphase

Definiert in Datei client.h.

7.4 client.h

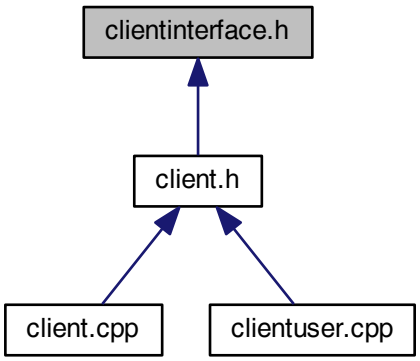
```
00001
00010 #ifndef CLIENT_H
00011 #define CLIENT_H
00012
00013 #include "clientinterface.h"
00014 #include "observer.h"
00015 #include "udpsocket.h"
00016 #include "defines.h"
00017
00018 class ClientUserInterface;
00019
00026 class Client : public Observer, public ClientInterface
00027 {
00028 public:
00038     Client(const unsigned short localPort, const string serverHost, const unsigned short serverPort);
00039
00045     ~Client();
00046
00054     bool start(ClientUserInterface* const user);
00055
00063     void update(Publish* const pub);
00064
00074     void update(Publish* const pub, const int* const iValues, const unsigned int iValuesSize);
00075
00087     void update(Publish* const pub, const float* const fValues, const unsigned int fValuesSize
      , const int* const iValues = NULL, const unsigned int iValuesSize = 0);
00088
00089     /*************************************************************************
00090     *   Hier müssen die virtuellen Memberfunktionen der Basisklasse          *
00091     *   ClientInterface definiert werden.                                    *
00092     *   z.B.: zum Anfragen/Abonnieren von bestimmten Daten, ...              *
00093     *************************************************************************/
00094
00101     bool requestAlive(void);
00102
00108     bool requestAxisPos(const Gamer gamer = BOTH);
00109
00114     bool requestScore();
00115
00120     bool requestBallPos();
00121
00129     bool subscribeAxisPos(const Gamer gamer = BOTH, const bool subs = true);
00130
00137     bool subscribeScore(const bool subs = true);
00138
00144     bool subscribeBallPos(const bool subs = true);
00145
00156     bool sendAxisPos(const float axisPos[2][4], const float axisAngle[2][4]);
00157
00164     bool sendScore(const unsigned short computer, const unsigned short human);
00165
00166 private:
00170     UDPsocket* mySocket;
00171
00175     ClientUserInterface* myUser;
00176
00180     const char* myRecvData;
00181
00185     sockaddr_in myServerAddr;
00186
00192     HANDLE myMutexHandle;
00193
00199     long myAliveID;
00200
00208     bool myAliveToggle;
00209
00215     bool answerAlive(const long aliveID);
00216 };
```

```
00217
00218 #endif // CLIENT_H
```

7.5 clientinterface.h-Dateireferenz

Enthält das Interface zur Kommunikation mit dem UDP-Client.

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



Klassen

- interface ClientInterface

  Interface zur Kommunikation mit dem UDP-Client.

7.5.1 Ausführliche Beschreibung

Enthält das Interface zur Kommunikation mit dem UDP-Client.

Autor

Scharel Clemens

Datum

08.02.2012

Version

0.1 Testphase

Definiert in Datei clientinterface.h.

7.6 clientinterface.h

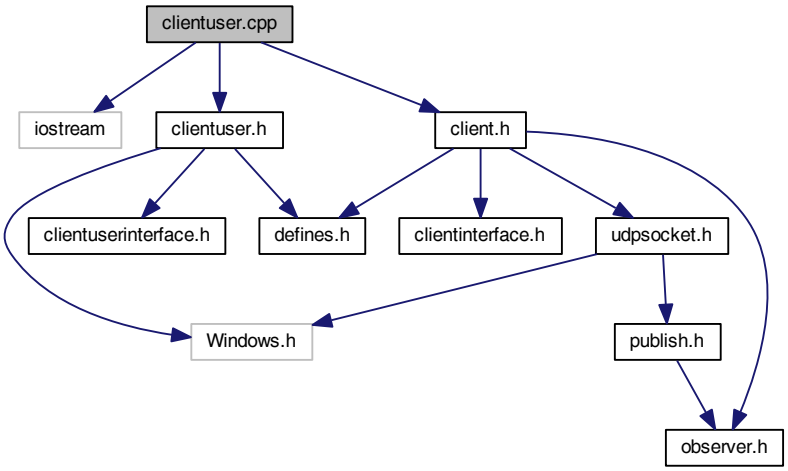
```
00001
00010 #ifndef CLIENTINTERFACE_H
00011 #define CLIENTINTERFACE_H
00012
00013 class ClientUserInterface;
00014 enum Gamer;
00015
00020 class ClientInterface
00021 {
00022 public:
00026     virtual ~ClientInterface(void) {};
00027
00033     virtual bool start(ClientUserInterface* const user) = 0;
00034
00035     /*************************************************************************
00036     *   Hier müssen die rein virtuellen Memberfunktionen definiert werden,     *
00037     *   welche vom Client implementiert werden müssen.                         *
00038     *   Diese Funktionen werden vom Benutzer des Clients aufgerufen.           *
00039     *   z.B.: zum Anfragen/Abonnieren von bestimmten Daten, ...                *
00040     *************************************************************************/
00041
00048     virtual bool requestAlive(void) = 0;
00049
00055     virtual bool requestAxisPos(const Gamer gamer) = 0;
00056
00061     virtual bool requestScore() = 0;
00062
00067     virtual bool requestBallPos() = 0;
00068
00076     virtual bool subscribeAxisPos(const Gamer gamer, const bool subs = true) = 0;
00077
00084     virtual bool subscribeScore(const bool subs = true) = 0;
00085
00091     virtual bool subscribeBallPos(const bool subs = true) = 0;
00092
00103     virtual bool sendAxisPos(const float axisPos[2][4], const float axisAngle[2][4]) = 0;
00104
00111     virtual bool sendScore(const unsigned short computer, const unsigned short human) = 0;
00112
00113 private:
00119     virtual bool answerAlive(const long aliveID) = 0;
00120 };
00121
00122 #endif  // CLIENTINTERFACE_H
```

7.7 clientuser.cpp-Dateireferenz

Enthält die Implementierung der Klasse ClientUser.

```
#include <iostream>
#include "clientuser.h"
#include "client.h"
```

Include-Abhängigkeitsdiagramm für clientuser.cpp:



7.7.1 Ausführliche Beschreibung

Enthält die Implementierung der Klasse ClientUser.

Autor

Scharel Clemens

Datum

07.12.2012

Version

0.1 Testphase

Definiert in Datei clientuser.cpp.

7.8 clientuser.cpp

```
00001
00010 #include <iostream>
00011 using namespace std;
00012
00013 #include "clientuser.h"
00014 #include "client.h"
00015
00016 ClientUser::ClientUser()
00017 {
00018     //  Startwerte festlegen
00019     myComputerAxisPos[VALUE][AXIS_ONE] = myHumanAxisPos[AXIS_ONE] =
      TRAVERSE_DISTANCE_1 / 2;
00020     myComputerAxisPos[VALUE][AXIS_TWO] = myHumanAxisPos[AXIS_TWO] =
      TRAVERSE_DISTANCE_2 / 2;
00021     myComputerAxisPos[VALUE][AXIS_THREE] = myHumanAxisPos[
      AXIS_THREE] = TRAVERSE_DISTANCE_3 / 2;
00022     myComputerAxisPos[VALUE][AXIS_FOUR] = myHumanAxisPos[AXIS_FOUR] =
      TRAVERSE_DISTANCE_4 / 2;
00023
```

```
00024     for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00025     {
00026         myComputerAxisPos[VELOCITY][axis] = MAX_VELOCITY_TRANS;
00027         myComputerAxisAngle[VELOCITY][axis] = MAX_VELOCITY_ROT;
00028         myHumanAxisAngle[axis] = 0.f;
00029         myComputerAxisAngle[VALUE][axis] = 0.f;
00030     }
00031     myScore[HUMAN] = myScore[COMPUTER] = 0;
00032
00033     myBallPos.bCoordinateOk = false;
00034     myBallPos.timestamp = 0;
00035     myBallPos.ulFrameCnt = 0;
00036     myBallPos.x = FIELD_X_MAX / 2.f;
00037     myBallPos.y = FIELD_Y_MAX / 2.f;
00038
00039     myMutexHandle = CreateMutex(NULL, FALSE, NULL);
00040     if (!myMutexHandle)
00041         cout << "Fehler beim erzeugen des Mutex: " << GetLastError() << endl;
00042     myClient = new Client(CLIENT_PORT, SERVER_HOST,
      SERVER_PORT);
00043 }
00044
00045 ClientUser::~ClientUser()
00046 {
00047     if (myClient)
00048         delete myClient;
00049     myClient = NULL;
00050     CloseHandle(myMutexHandle);
00051 }
00052
00053 bool ClientUser::start()
00054 {
00055     bool retval = false;
00056     if (myClient->start(this))
00057         retval = true;
00058     return retval;
00059 }
00060
00061 bool ClientUser::run()
00062 {
00063     bool retval = false;
00064     bool end = false;
00065     char input;
00066
00067     do
00068     {
00069         cout
00070             << "[A] Daten ausgeben    |    [U] Daten abfragen" << endl
00071             << "[E] Daten eingeben    |    [R] Daten abonnieren" << endl
00072             << "[K] Alive anfragen    |    [S] Daten senden" << endl
00073             << "                      |" << endl
00074             << "[Q] Beenden           |" << endl
00075             << "------------------------------------------------------------" << endl
00076             << "Option waehlen: ";
00077         cin >> input;
00078         cout << endl << endl;
00079         switch (input)
00080         {
00081         case 'a':
00082         case 'A':
00083             cout << "Spielstangenpositionen:" << endl;
00084             cout << "Mensch:" << endl;
00085             for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00086                 cout << axis << ": " << myHumanAxisPos[axis] << "mm \t" << myHumanAxisAngle[axis] << char(2
      48) << endl;
00087             cout << "Computer:" << endl;
00088             for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00089                 cout
00090                 << axis << ": "
00091                 << myComputerAxisPos[VALUE][axis] << "mm (" << myComputerAxisPos[
      VELOCITY][axis] << "mm/s) \t"
00092                 << myComputerAxisAngle[VALUE][axis] << char(248) << " (" << myComputerAxisAngle[
      VELOCITY][axis] << char(248) << "/s)" << endl;
00093
00094             cout << endl;
00095
00096             cout << "Ballposition:" << endl;
00097             cout
00098                 << "X: " << myBallPos.x << endl
00099                 << "Y: " << myBallPos.y << endl;
00100
00101             cout << endl;
00102
00103             cout << "Spielstand:" << endl;
00104             for (unsigned int gamer = COMPUTER; gamer <= HUMAN; gamer++)
00105             {
00106                 if (gamer == HUMAN)
```

```
00107                   cout << "Mensch:   ";
00108               if (gamer == COMPUTER)
00109                   cout << "Computer: ";
00110               cout << myScore[gamer] << endl;
00111           }
00112           break;
00113
00114       case 'e':
00115       case 'E':
00116           cout
00117               << "[P] Spielstangenpositionen (Computer)" << endl
00118               << "[S] Spielstand" << endl
00119               << "------------------------------------------------------------" << endl
00120               << "Option waehlen: ";
00121               cin >> input;
00122               cout << endl << endl;
00123               switch (input)
00124               {
00125               case 'p':
00126               case 'P':
00127                   int axisToSet;
00128                   float newAxisPos[2][2];
00129
00130                   cout
00131                       << "[1] Torward    |   [3] Mittelfeld" << endl
00132                       << "[2] Abwehr     |   [4] Sturm" << endl
00133                       << "------------------------------------------------------------" << endl
00134                       << "Option waehlen: ";
00135                   cin >> input;
00136                   cout << endl << endl;
00137                   switch (input)
00138                   {
00139                   case '1':
00140                       axisToSet = AXIS_ONE;
00141                       break;
00142                   case '2':
00143                       axisToSet = AXIS_TWO;
00144                       break;
00145                   case '3':
00146                       axisToSet = AXIS_THREE;
00147                       break;
00148                   case '4':
00149                       axisToSet = AXIS_FOUR;
00150                       break;
00151                   default:
00152                       axisToSet = -1;
00153                       break;
00154                   }
00155
00156                   if (axisToSet >= 0)
00157                   {
00158                       cout << "Linearposition eingeben: ";
00159                       cin >> newAxisPos[POSITION][VALUE];
00160                       cout << "Lineargeschwindigkeit eingeben: ";
00161                       cin >> newAxisPos[POSITION][VELOCITY];
00162                       cout << "Rotationswinkel eingeben: ";
00163                       cin >> newAxisPos[ANGLE][VALUE];
00164                       cout << "Rotationsgeschwindigkeit eingeben: ";
00165                       cin >> newAxisPos[ANGLE][VELOCITY];
00166
00167                       if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00168                       {
00169                           myComputerAxisPos[VALUE][axisToSet] = newAxisPos[
    POSITION][VALUE];
00170                           myComputerAxisPos[VELOCITY][axisToSet] = newAxisPos[
    POSITION][VELOCITY];
00171                           myComputerAxisAngle[VALUE][axisToSet] = newAxisPos[
    ANGLE][VALUE];
00172                           myComputerAxisAngle[VELOCITY][axisToSet] = newAxisPos[
    ANGLE][VELOCITY];
00173                           ReleaseMutex(myMutexHandle);
00174                       }
00175                   }
00176                   break;
00177               case 's':
00178               case 'S':
00179                   int scoreToSet;
00180                   unsigned short score;
00181
00182                   cout
00183                       << "[H] Mensch" << endl
00184                       << "[C] Computer" << endl
00185                       << "------------------------------------------------------------" << endl
00186                       << "Option waehlen: ";
00187                   cin >> input;
00188                   cout << endl << endl;
00189                   switch (input)
```

```
00190                   {
00191                   case 'h':
00192                   case 'H':
00193                       scoreToSet = HUMAN;
00194                       break;
00195                   case 'c':
00196                   case 'C':
00197                       scoreToSet = COMPUTER;
00198                       break;
00199                   default:
00200                       scoreToSet = -1;
00201                       break;
00202                   }
00203
00204                   if (scoreToSet >= 0)
00205                   {
00206                       cout << "Spielstand eingeben: ";
00207                       cin >> score;
00208
00209                       if (WaitForSingleObject(myMutexHandle, 10) == WAIT_OBJECT_0)
00210                       {
00211                           myScore[scoreToSet] = score;
00212                           ReleaseMutex(myMutexHandle);
00213                       }
00214                   }
00215                   break;
00216               }
00217           break;
00218
00219       case 'u':
00220       case 'U':
00221           cout
00222               << "[P] Spielstangenpositionen" << endl
00223               << "[B] Ballposition" << endl
00224               << "[S] Spielstand" << endl
00225               << "------------------------------------------------------------" << endl
00226               << "Option waehlen: ";
00227           cin >> input;
00228           cout << endl << endl;
00229           switch (input)
00230           {
00231           case 'p':
00232           case 'P':
00233               cout
00234                   << "[H] Mensch" << endl
00235                   << "[C] Computer" << endl
00236                   << "[B] Mensch & Computer" << endl
00237                   << "------------------------------------------------------------" << endl
00238                   << "Option waehlen: ";
00239               cin >> input;
00240               cout << endl << endl;
00241               switch (input)
00242               {
00243               case 'h':
00244               case 'H':
00245                   cout << "Es wird die Spielstangenposition des Menschen abgefragt!" << endl;
00246                   if (!myClient->requestAxisPos(HUMAN))
00247                       cout << "Fehler beim Abfragen der Spielstangenpositionen des Menschen!" << endl;
00248                   break;
00249               case 'c':
00250               case 'C':
00251                   cout << "Es wird die Spielstangenposition des Computer abgefragt!" << endl;
00252                   if (!myClient->requestAxisPos(COMPUTER))
00253                       cout << "Fehler beim Abfragen der Spielstangenpositionen des Computers!" << endl;
00254                   break;
00255               case 'b':
00256               case 'B':
00257                   if (!myClient->requestAxisPos(BOTH))
00258                       cout << "Fehler beim Abfragen der Spielstangenpositionen beider Spieler!" << endl;
00259                   break;
00260               }
00261               break;
00262
00263           case 'b':
00264           case 'B':
00265               if (!myClient->requestBallPos())
00266                   cout << "Fehler beim Abfragen der Ballposition!" << endl;
00267               break;
00268
00269           case 's':
00270           case 'S':
00271               if (!myClient->requestScore())
00272                   cout << "Fehler beim Abfragen des Spielstandes!" << endl;
00273               break;
00274           }
00275           break;
00276
```

```
00277         case 'r':
00278         case 'R':
00279             cout
00280                 << "[B] Ballposition    |   [N] Ballposition kuendigen" << endl
00281                 << "[S] Spielstand      |   [D] Spielstand kuendigen" << endl
00282                 << "------------------------------------------------------------" << endl
00283                 << "Option waehlen: ";
00284             cin >> input;
00285             cout << endl << endl;
00286             switch (input)
00287             {
00288             case 'b':
00289             case 'B':
00290                 if (!myClient->subscribeBallPos())
00291                     cout << "Fehler beim Abonnieren der Ballposition!" << endl;
00292                 break;
00293             case 's':
00294             case 'S':
00295                 if (!myClient->subscribeScore())
00296                     cout << "Fehler beim Abonnieren des Spielstandes!" << endl;
00297                 break;
00298             case 'n':
00299             case 'N':
00300                 if (!myClient->subscribeBallPos(false))
00301                     cout << "Fehler beim Kuendigen der Ballposition!" << endl;
00302                 break;
00303             case 'd':
00304             case 'D':
00305                 if (!myClient->subscribeScore(false))
00306                     cout << "Fehler beim Kuendigen des Spielstandes!" << endl;
00307                 break;
00308             }
00309             break;
00310 
00311         case 's':
00312         case 'S':
00313             cout
00314                 << "[P] Spielstangenpositionen" << endl
00315                 << "[S] Spielstand" << endl
00316                 << "------------------------------------------------------------" << endl
00317                 << "Option waehlen: ";
00318             cin >> input;
00319             cout << endl << endl;
00320             switch (input)
00321             {
00322             case 'p':
00323             case 'P':
00324                 if (!myClient->sendAxisPos(myComputerAxisPos, myComputerAxisAngle))
00325                     cout << "Fehler beim Senden der Spielstangenpositionen des Computers!" << endl;
00326                 break;
00327 
00328             case 's':
00329             case 'S':
00330                 if (!myClient->sendScore(myScore[COMPUTER], myScore[
    HUMAN]))
00331                     cout << "Fehler beim Senden des Spielstandes!" << endl;
00332                 break;
00333             }
00334             break;
00335         case 'k':
00336         case 'K':
00337             if (myClient->requestAlive())
00338                 cout << "Server hat auf die letzte Anfrage geantwortet." << endl;
00339             else
00340                 cout << "Server hat nicht auf die letzte Anfrage geantwortet!" << endl;
00341             break;
00342         case 'q':
00343         case 'Q':
00344             end = true;
00345             break;
00346         default:
00347             cout << "Ungueltiges Zeichen!" << endl;
00348         }
00349 
00350         cout << endl << endl << endl;
00351     } while(!end);
00352 
00353     retval = true;
00354     return retval;
00355 }
00356 
00357 bool ClientUser::setHumanAxisPos(const float axisPos[4], const float axisAngle[4
    ])
00358 {
00359     bool retval = false;
00360     DWORD waitResult;
00361 
```

```
00362     //  Sicheren Datenzugriff auf myAxisPos und myAxisAngel da diese Funktion von einem anderen Thread
      aufgerufen wird
00363     waitResult = WaitForSingleObject(myMutexHandle, 10);
00364     if (waitResult == WAIT_OBJECT_0)
00365     {
00366         //  Spielstangenpositionen übernehmen
00367         for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00368         {
00369             myHumanAxisPos[axis] = axisPos[axis];
00370             myHumanAxisAngle[axis] = axisAngle[axis];
00371         }
00372 
00373         //  Mutex wieder freigeben
00374         if (!ReleaseMutex(myMutexHandle))
00375         {
00376             cout << "Mutex konnte nicht wieder freigegeben werden! (setHumanAxisPos())" << endl;
00377         }
00378         retval = true;
00379     }
00380     else
00381         cout << "Fehler beim Warten auf Mutex! (setHumanAxisPos())" << endl;
00382     return retval;
00383 
00384     /*
00385     cout << endl << "Menschliche Spielstangenposition:" << endl;
00386     for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00387         cout << axis + 1 << ": " << axisPos[axis] << "mm, " << axisAngle[axis] << static_cast<unsigned
      char>(248) << endl;
00388     */
00389 }
00390 
00391 bool ClientUser::setComputerAxisPos(const float axisPos[4], const float
      axisAngle[4])
00392 {
00393     bool retval = false;
00394     DWORD waitResult;
00395 
00396     //  Sicheren Datenzugriff auf myAxisPos und myAxisAngel da diese Funktion von einem anderen Thread
      aufgerufen wird
00397     waitResult = WaitForSingleObject(myMutexHandle, 10);
00398     if (waitResult == WAIT_OBJECT_0)
00399     {
00400         //  Spielstangenpositionen übernehmen
00401         for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00402         {
00403             myComputerAxisPos[VALUE][axis] = axisPos[axis];
00404             myComputerAxisAngle[VALUE][axis] = axisAngle[axis];
00405         }
00406 
00407         //  Mutex wieder freigeben
00408         if (!ReleaseMutex(myMutexHandle))
00409         {
00410             cout << "Mutex konnte nicht wieder freigegeben werden! (setComputerAxisPos())" << endl;
00411         }
00412         retval = true;
00413     }
00414     else
00415         cout << "Fehler beim Warten auf Mutex! (setComputerAxisPos())" << endl;
00416     return retval;
00417 
00418     /*
00419     cout << endl << "Computer Spielstangenposition:" << endl;
00420     for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00421         cout << axis + 1 << ": " << axisPos[axis] << "mm, " << axisAngle[axis] << static_cast<unsigned
      char>(248) << endl;
00422     */
00423 }
00424 
00425 bool ClientUser::setScore(const unsigned short human, const unsigned short computer)
00426 {
00427     bool retval = false;
00428     DWORD waitResult;
00429 
00430     //  Sicheren Datenzugriff auf myAxisPos und myAxisAngel da diese Funktion von einem anderen Thread
      aufgerufen wird
00431     waitResult = WaitForSingleObject(myMutexHandle, 10);
00432     if (waitResult == WAIT_OBJECT_0)
00433     {
00434         //  Spielstangenpositionen übernehmen
00435         for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00436         {
00437             myScore[HUMAN] = human;
00438             myScore[COMPUTER] = computer;
00439         }
00440 
00441         //  Mutex wieder freigeben
00442         if (!ReleaseMutex(myMutexHandle))
```

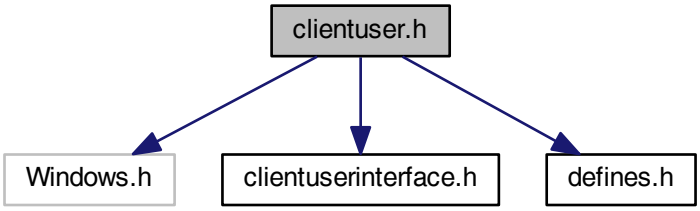
```
00443         {
00444             cout << "Mutex konnte nicht wieder freigegeben werden! (setScore())" << endl;
00445         }
00446         retval = true;
00447     }
00448     else
00449         cout << "Fehler beim Warten auf Mutex! (setScore())" << endl;
00450     return retval;
00451 
00452     /*
00453     cout << endl << "Spielstand:" << endl;
00454     cout << computer << ":" << human << " (Computer:Mensch)" << endl;
00455     */
00456 }
00457 
00458 bool ClientUser::setBallPos(const ballCoordinate_t ballPos)
00459 {
00460     bool retval = false;
00461     DWORD waitResult;
00462 
00463     //  Sicheren Datenzugriff auf myAxisPos und myAxisAngel da diese Funktion von einem anderen Thread
     aufgerufen wird
00464     waitResult = WaitForSingleObject(myMutexHandle, 10);
00465     if (waitResult == WAIT_OBJECT_0)
00466     {
00467         //  Spielstangenpositionen übernehmen
00468         for (unsigned int axis = AXIS_ONE; axis <= AXIS_FOUR; axis++)
00469         {
00470             myBallPos = ballPos;
00471         }
00472 
00473         //  Mutex wieder freigeben
00474         if (!ReleaseMutex(myMutexHandle))
00475         {
00476             cout << "Mutex konnte nicht wieder freigegeben werden! (setBallPos())" << endl;
00477         }
00478         retval = true;
00479     }
00480     else
00481         cout << "Fehler beim Warten auf Mutex! (setBallPos())" << endl;
00482     return retval;
00483 
00484     /*
00485     cout << endl << "Ballposition:" << endl;
00486     cout << endl
00487         << "Frame: " << ballPos.ulFrameCnt << endl
00488         << "X: " << ballPos.x << endl
00489         << "Y: " << ballPos.y << endl
00490         << "OK: " << boolalpha << ballPos.bCoordinateOk << endl;
00491     */
00492 }
```

7.9 clientuser.h-Dateireferenz

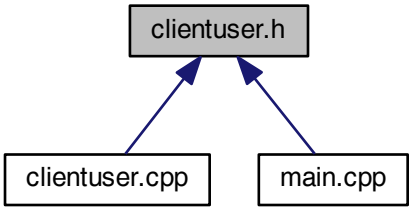
Enthält die Definition der Klasse ClientUser.

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

Include-Abhängigkeitsdiagramm für clientuser.h:



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



Klassen

- class ClientUser

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

7.9.1 Ausführliche Beschreibung

Enthält die Definition der Klasse ClientUser.

Autor

Scharel Clemens

Datum

07.12.2012

Version

0.1 Testphase

Definiert in Datei clientuser.h.

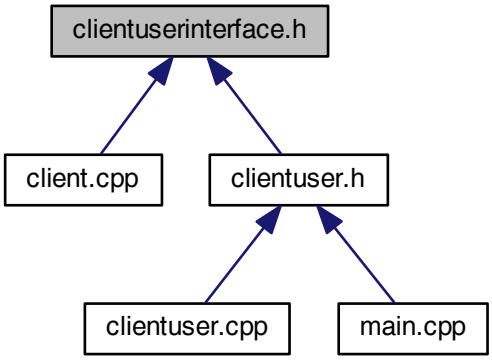
7.10 clientuser.h

```
00001
00010 #ifndef CLIENTUSER_H
00011 #define CLIENTUSER_H
00012
00013 #include <Windows.h>
00014
00015 #include "clientuserinterface.h"
00016 #include "defines.h"
00017
00018 class ClientInterface;
00019
00026 class ClientUser : public ClientUserInterface
00027 {
00028 public:
00034     ClientUser();
00035
00041     ~ClientUser();
00042
00047     bool start();
00048
00054     bool run();
00055
00056     /*****************************************************************************
00057     *   Hier müssen die virtuellen Memberfunktionen der Basisklasse             *
00058     *   ClientUserInterface definiert werden.                                   *
00059     *   z.B.: zum Übergeben von empfangenen Daten, ...                          *
00060     *****************************************************************************/
00061
00070     bool setHumanAxisPos(const float axisPos[4], const float axisAngle[4]);
00071
00080     bool setComputerAxisPos(const float axisPos[4], const float axisAngle[4]);
00081
00090     bool setScore(const unsigned short human, const unsigned short computer);
00091
00099     bool setBallPos(const ballCoordinate_t ballPos);
00100
00101 private:
00105     ClientInterface* myClient;
00106
00113     HANDLE myMutexHandle;
00114
00121     float myHumanAxisPos[4];
00122
00129     float myHumanAxisAngle[4];
00130
00138     float myComputerAxisPos[2][4];
00139
00147     float myComputerAxisAngle[2][4];
00148
00154     unsigned myScore[2];
00155
00159     ballCoordinate_t myBallPos;
00160 };
00161
00162 #endif //CLIENTUSER_H
```

7.11 clientuserinterface.h-Dateireferenz

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

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



Klassen

- interface ClientUserInterface

  Interface zur Kommunikation mit dem Benutzer des UDP-Clients.

7.11.1 Ausführliche Beschreibung

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

Autor

Scharel Clemens

Datum

07.12.2012

Version

0.1 Testphase

Definiert in Datei clientuserinterface.h.

7.12 clientuserinterface.h

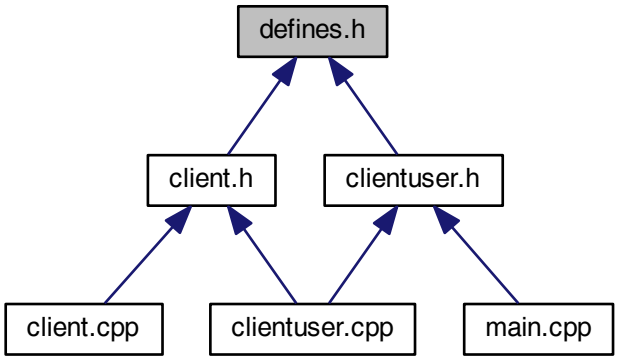
```
00001
00010 #ifndef CLIENTUSERINTERFACE_H
00011 #define CLIENTUSERINTERFACE_H
00012
00013 struct ballCoordinate_t;
00014
00019 class ClientUserInterface
00020 {
00021 public:
00025     virtual ~ClientUserInterface(void) {};
00026
00031     virtual bool start() = 0;
```

```
00032
00033     /*************************************************************************
00034     *   Hier müssen die rein virtuellen Memberfunktionen definiert werden,     *
00035     *   welche vom Benutzer des Clients implementiert werden müssen.           *
00036     *   Diese Funktionen werden vom Client aufgerufen.                         *
00037     *   z.B.: zum Übergeben von empfangenen Daten, ...                         *
00038     *************************************************************************/
00039
00048     virtual bool setHumanAxisPos(const float axisPos[4], const float axisAngle[4]) = 0;
00049
00058     virtual bool setComputerAxisPos(const float axisPos[4], const float axisAngle[4]) = 0
      ;
00059
00068     virtual bool setScore(const unsigned short human, const unsigned short computer) = 0;
00069
00077     virtual bool setBallPos(const ballCoordinate_t ballPos) = 0;
00078 };
00079
00080 #endif  // CLIENTUSERINTERFACE_H
```

7.13 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.

7.13.1 Ausführliche Beschreibung

Enthält die defines der verschiedenen Strukturen.

Autor

Scharel Clemens

Datum

11.12.2012

Version

0.1 Testphase

Definiert in Datei defines.h.

7.13.2 Makro-Dokumentation

7.13.2.1 #define SERVER_HOST "127.0.0.1"

Host des Servers.

Definiert in Zeile 21 der Datei defines.h.

Wird benutzt von ClientUser::ClientUser().

7.13.3 Dokumentation der benutzerdefinierten Typen

7.13.3.1 typedef struct ballCoordinate_t ballCoordinate_t

Struktur der Ballposition.

7.13.3.2 typedef struct ballPacket ballPacket

Paket mit der Ballposition.

Beinhaltet die Paket-ID und eine Struktur mit der Ballposition

7.13.3.3 typedef struct bool_1 bool_1

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein bool-Wert.

7.13.3.4 typedef struct bool_2 bool_2

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und zwei bool-Werte.

7.13.3.5 typedef struct float_1 float_1

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein float-Wert.

7.13.3.6 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.

7.13.3.7 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.

7.13.3.8 typedef struct float_4 float_4

Anfrage-Paket für bestimmte Daten.

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

7.13.3.9 typedef struct long_1 long_1

Anfrage-Paket für bestimmte Daten.

Beinhaltet die Paket-ID und ein long-Wert.

7.13.3.10 typedef struct requestPacket requestPacket

Anfrage-Paket für bestimmte Daten.

7.13.3.11 typedef struct subscribePacket subscribePacket

Abonnieren-Paket für bestimmte Daten.

7.13.3.12 typedef union ubf16_request ubf16_request

Bitfeld, bei dem jedes Bit bestimmte Daten anfordert.

7.13.3.13 typedef struct unsignedShort_2 unsignedShort_2

Anfrage-Paket für bestimmte Daten.

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

7.13.4 Dokumentation der Aufzählungstypen

7.13.4.1 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.

7.13.4.2 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.

7.13.4.3 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.

7.13.4.4 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.

7.13.4.5 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.

7.13.4.6 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.

7.13.5 Variablen-Dokumentation

7.13.5.1 const unsigned short CLIENT_PORT = 60000

Port des Clients.

Definiert in Zeile 16 der Datei defines.h.

Wird benutzt von ClientUser::ClientUser().

7.13.5.2 const float FIELD_X_MAX = 1200

Länge des Spielfeldes in mm.

Definiert in Zeile 36 der Datei defines.h.

Wird benutzt von ClientUser::ClientUser().

7.13.5.3 const float FIELD_Y_MAX = 690

Tiefe des Spielfeldes in mm.

Definiert in Zeile 41 der Datei defines.h.

Wird benutzt von ClientUser::ClientUser().

7.13.5.4 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 ClientUser::ClientUser().

7.13.5.5 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 ClientUser::ClientUser().

7.13.5.6 const unsigned int RECV_BUFFER_SIZE = 256

Größe des Empfangsspeichers von UDPsocket.

Definiert in Zeile 31 der Datei defines.h.

Wird benutzt von Client::Client().

7.13.5.7 const unsigned short SERVER_PORT = 50000

Port des Servers.

Definiert in Zeile 26 der Datei defines.h.

Wird benutzt von ClientUser::ClientUser().

7.13.5.8 const int TRAVERSE_DISTANCE_1 = 241

Verfahrweg des Torwards.

Definiert in Zeile 46 der Datei defines.h.

Wird benutzt von ClientUser::ClientUser().

7.13.5.9 const int TRAVERSE_DISTANCE_2 = 377

Verfahrweg des Abwehr.

Definiert in Zeile 51 der Datei defines.h.

Wird benutzt von ClientUser::ClientUser().

7.13.5.10 const int TRAVERSE_DISTANCE_3 = 233

Verfahrweg des Mittelfeld.

Definiert in Zeile 56 der Datei defines.h.

Wird benutzt von ClientUser::ClientUser().

7.13.5.11 const int TRAVERSE_DISTANCE_4 = 120

Verfahrweg des Angriff.

Definiert in Zeile 61 der Datei defines.h.

Wird benutzt von ClientUser::ClientUser().

7.14 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
```

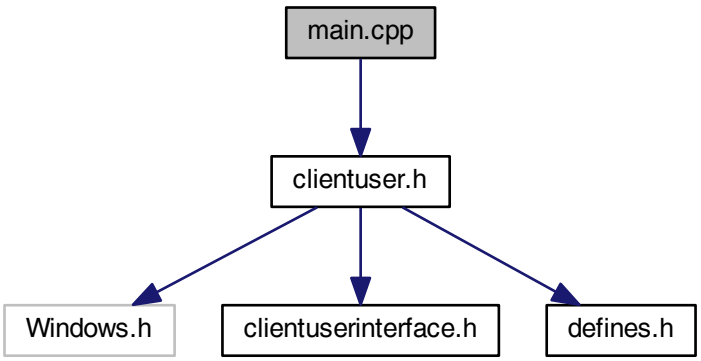
```
00196         unsigned int score : 1;                // liefert GET_SCORE
00197         unsigned int ballPos : 1;              // liefert GET_BALLPOS
00198         unsigned int reserved_05 : 1;             // reserviert
00199         unsigned int reserved_06 : 1;             // reserviert
00200         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
```

7.15 main.cpp-Dateireferenz

Enthält die Main-Funktion.

`#include "clientuser.h"`
Include-Abhängigkeitsdiagramm für main.cpp:



Funktionen

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

7.15.1 Ausführliche Beschreibung

Enthält die Main-Funktion.

Autor

Scharel Clemens

Datum

08.01.2013

Version

0.9 Testphase

Definiert in Datei main.cpp.

7.15.2 Dokumentation der Funktionen

7.15.2.1 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

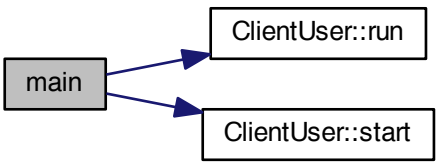
0 wenn die Applikation sachgemäß beendet wird

Legt die Klasse ClientUser an und führt sie aus.

Definiert in Zeile 133 der Datei main.cpp.

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

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



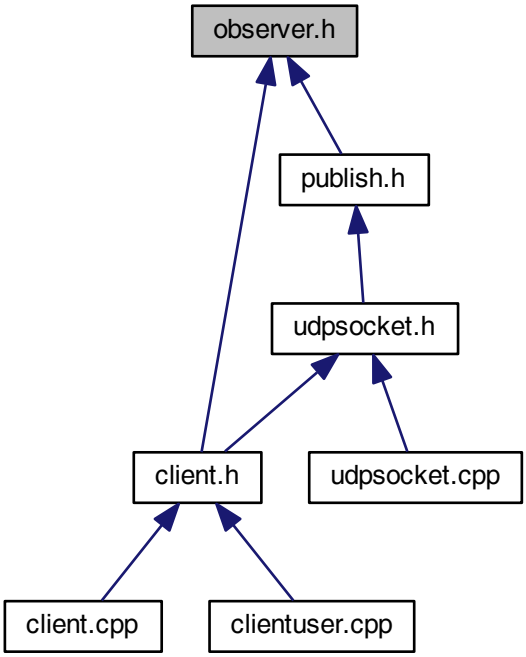
7.16 main.cpp

```
00001
00121 #include "clientuser.h"
00122
00133 int main(int argc, char** argv)
00134 {
00135     ClientUser* m_pClientUser = new ClientUser();
00136
00137     if (m_pClientUser->start())
00138         m_pClientUser->run();
00139
00140     if (m_pClientUser)
00141         delete m_pClientUser;
00142     m_pClientUser = NULL;
00143
00144     return EXIT_SUCCESS;
00145 }
```

7.17 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.

7.17.1 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.

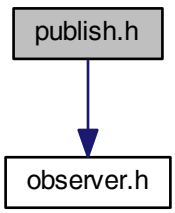
7.18 observer.h

```
00001
00010 #ifndef OBSERVER_H
00011 #define OBSERVER_H
00012
00013 class Publish;
00014
00020 class Observer
00021 {
00022 public:
00026     virtual ~Observer() {};
00027
00035     virtual void update(Publish* const pub) = 0;
00036
00043     virtual void update(Publish* const pub, const int* const iValues, const unsigned int
      iValuesSize) = 0;
00044
00053     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;
00054
00055 protected:
00059     Observer() {}
00060 };
00061
00062 #endif
```

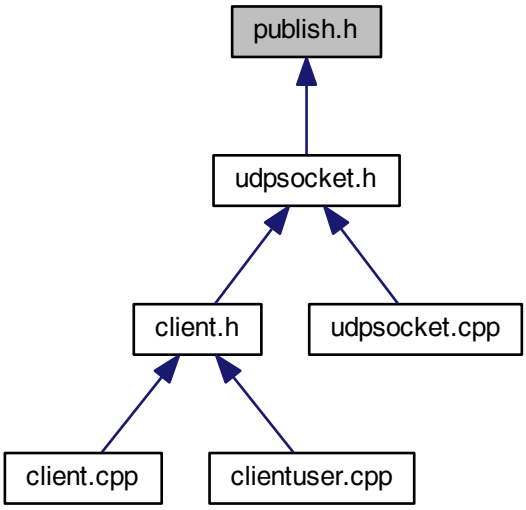
7.19 publish.h-Dateireferenz

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

7.19.1 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.

7.19.2 Makro-Dokumentation

7.19.2.1 #define NULL 0

Definiert in Zeile 15 der Datei publish.h.

Wird benutzt von Publish::attach(), Client::Client(), ClientUser::ClientUser(), Publish::detach(), UDPsocket::getAddrFromString(), UDPsocket::getRecvDataInstance(), main(), Publish::Publish(), UDPsocket::recvData(), UDPsocket::start(), Client::~Client(), ClientUser::~ClientUser() und UDPsocket::~UDPsocket().

7.19.3 Dokumentation der benutzerdefinierten Typen

7.19.3.1 typedef struct ObsListKnot ObsListKnot_t

7.20 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, ...
```

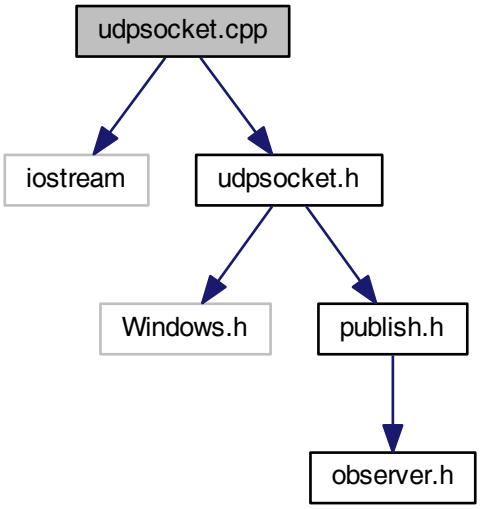
```
00100                 else if(knot == lastObs)
00101                 {
00102                     //  dann wird das zweitletzte Element zum Letzten
00103                     if (knot->prev)
00104                         knot->prev->next = NULL;
00105                     lastObs = knot->prev;
00106                 }
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
```

7.21 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.

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.

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
```

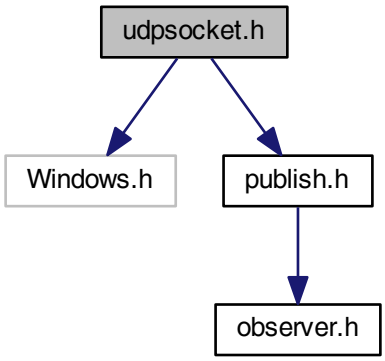
```
00212         iResult = recvfrom(mySock, myBuffer, myBufferSize, 0, (SOCKADDR*)&
      mySenderAddr, &senderAddrSize);
00213         //iResult = 12;
00214         if (iResult == SOCKET_ERROR)
00215             cout << "Fehler beim Empfangen eines Pakets: " << WSAGetLastError() << "!" << endl;
00216         //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

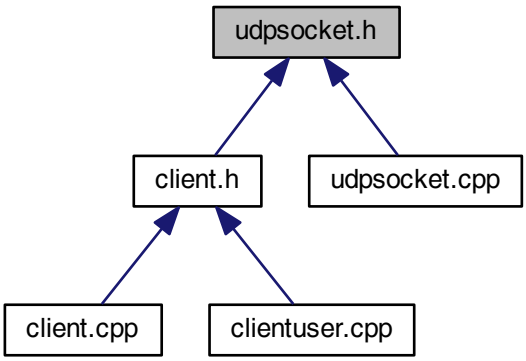
Enthält die Definition der Klasse UDPsocket.

```
#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.

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.

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