Entwicklerdokumentation

Service-Interface für ein Formula-Student-Fahrzeug

Technische Universität Ilmenau
Softwareprojekt SS 2013
Gruppe 19

Christian Boxdörfer
Thomas Golda
Daniel Häger
David Kudlek
Tom Porzig
Tino Tausch
Tobias Zehner
Sebastian Zehnter

19.06.2013

betreut durch

Dr. Heinz-Dietrich Wuttke, TU Ilmenau
Oliver Dittrich, fachlicher Betreuer Team StarCraft e.V.

Inhaltsverzeichnis

1 Hierarchie-Verzeichnis

1.1 Klassenhierarchie

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

2 Klassen-Verzeichnis

2.1 Auflistung der Klassen

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

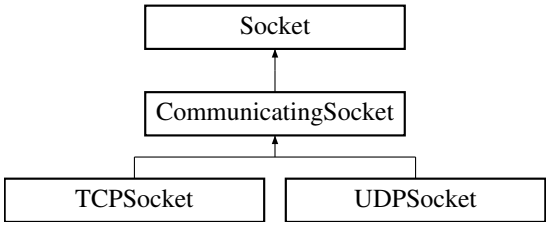
3 Klassen-Dokumentation

Wir bedienen uns teilweise an der PracticalSocket-Bibliothek um die Socket-Kommunikaton unter C++ zu vereinfachen. Der Quellcode in `PracticalSocket.h` und `PracticalSocket.cpp` stammt bis auf ein paar kleinere Änderungen also nicht von uns. Der Quelltext wurde hier entnommen: http://cs.ecs.baylor.edu/~donahoo/practical/CSockets/practical/

3.1 CommunicatingSocket Klassenreferenz

`#include <PracticalSocket.h>`

Klassendiagramm für CommunicatingSocket:



Öffentliche Methoden

- void connect (const string &foreignAddress, unsigned short foreignPort) throw (SocketException)
- void send (const void ∗buffer, int bufferLen) throw (SocketException)
- int recv (void ∗buffer, int bufferLen) throw (SocketException)
- string getForeignAddress () throw (SocketException)
- unsigned short getForeignPort () throw (SocketException)

Geschützte Methoden

- **CommunicatingSocket** (int type, int protocol) throw (SocketException)
- **CommunicatingSocket** (int newConnSD)

Weitere Geerbte Elemente

3.1.1 Ausführliche Beschreibung

Socket der in der Lage ist sich zu verbinden, zu senden und zu empfangen.

3.1.2 Dokumentation der Elementfunktionen

3.1.2.1 void CommunicatingSocket::connect (const string & *foreignAddress,* unsigned short *foreignPort*) throw SocketException)

Baut eine Verbindung über Sockets zu einem bestimmten Teilnehmer.

Parameter

foreign-Address	Adresse des Teilnehmers (IP Adresse oder Name).
foreignPort	Portnummer des Teilnehmers.

Ausnahmebehandlung

SocketException	wird geworfen wenn die Verbindung fehlschlägt.

3.1.2.2 string CommunicatingSocket::getForeignAddress ( ) throw SocketException)

Holt die Zieladresse.

Rückgabe

Zieladresse.

Ausnahmebehandlung

SocketException	wird geworfen falls das Holen fehlschlägt.

3.1.2.3 unsigned short CommunicatingSocket::getForeignPort ( ) throw SocketException)

Holt den Zielport. Call connect() before calling recv()

Rückgabe

Zielportnummer.

Ausnahmebehandlung

SocketException	wird geworfen falls das Holen fehlschlägt.

3.1.2.4 int CommunicatingSocket::recv (void * *buffer,* int *bufferLen*) throw SocketException)

Ließt eine Nachricht in den übergebenen Buffer ein. connect() muss vorher aufgerufen werden.

Parameter

buffer	Speicher in den die Nachricht geschrieben werden soll.
bufferLen	Maximale Anzahl an Bytes die empfangen werden sollen.

Rückgabe

Gibt die Anzahl der gelesenen Bytes zurück. 0 wenn EOF und -1 im Fehlerfall.

Ausnahmebehandlung

SocketException	wird geworfen falls keine Daten emfpangen werden können.

3.1.2.5 void CommunicatingSocket::send (const void * *buffer*, int *bufferLen*) throw SocketException)

Schreibt den übergebenen Buffer in den Socket. Vor send() muss collect() aufgerufen werden.

Parameter

buffer	Speicher in den geschrieben werden soll.
bufferLen	Anzahl der Bytes die geschrieben werden sollen.

Ausnahmebehandlung

SocketException	wird geworfen wenn keine Daten gesendet werden können.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- PracticalSocket.h
- PracticalSocket.cpp

3.2 Data Klassenreferenz

`#include <Data.h>`

Öffentliche Methoden

- Data (double value, unsigned int datatype, unsigned int position)
- double getValue ()
- unsigned int getDatatype ()
- unsigned int getPosition ()

3.2.1 Ausführliche Beschreibung

Datenstruktur die einen Fahrzeugwert und die dazugehörigen Daten speichert.

3.2.2 Beschreibung der Konstruktoren und Destruktoren

3.2.2.1 Data::Data (double *value,* unsigned int *datatype,* unsigned int *position*)

Erzeugt eine Datenstruktur zur Speicherung von Fahrzeugdaten.

Parameter

value	Wert des Datensatzes.
datatype	Datentyp des Datensatzes.
position	Position des Datensatzes in den ursprünglichen Daten.

3.2.3 Dokumentation der Elementfunktionen

3.2.3.1 unsigned int Data::getDatatype ( )

Rückgabe

Gibt den Datentyp des Datensatzes zurück.

3.2.3.2 unsigned int Data::getPosition ( )

Rückgabe

Gibt die Position des Datensatzes in den ursprünlichen Daten zurück.

3.2.3.3 double Data::getValue ( )

Rückgabe

Gibt den Wert des Datensatzes zurück.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Data.h
- Data.cpp

3.3 DatatypeDaemon Klassenreferenz

Klasse zum Aufbereiten der Daten.

`#include <DatatypeDaemon.h>`

Öffentliche Methoden

- DatatypeDaemon ()
- string parseNextValue ()

- int getPosActualPacket ()
- int getTime ()

3.3.1 Ausführliche Beschreibung

Klasse zum Aufbereiten der Daten.

In dieser Klasse werden Die Daten nicht nur in das richtige Format umgewandelt, sondern auch durch eine Zehnerpotenz geteilt, sodass sie wieder ihrem Wert vor der Übertragung entsprechen. Es werden weiterhin Zusatzinformationen angeboten, die es sowohl ermöglichen, die Paketnummer und damit die Tabelle, in die eingefügt werden soll, zu ermitteln, als auch den Zeitstempel der Daten einzufügen. Diese Klasse dient also als Schnittstelle für DBPacketInsert zum Decoder.

Siehe auch

Data

3.3.2 Beschreibung der Konstruktoren und Destruktoren

3.3.2.1 DatatypeDaemon::DatatypeDaemon ( )

Konstruktor der Klasse.

3.3.3 Dokumentation der Elementfunktionen

3.3.3.1 int DatatypeDaemon::getPosActualPacket ( )

Diese Funktion liefert die Position des jetzigen Datenwerts im Übertragungsvektor der MikroAutoBoxII zurück.

Dies ist für die Bestimmung des Übertragungspakets vom Embedded PC wichtig, da darauf basierend die richtige Datenbankanfrage in DBPacketInsert aufgebaut wird.

Rückgabe

Aus Data extrahierter und aufbereiteter Wert im String-Format.

3.3.3.2 int DatatypeDaemon::getTime ( )

Diese Memberfunktion gibt den Zeitstempel des aktuellen Pakets zurück.

Rückgabe

Zeitstempel des Pakets.

3.3.3.3 string DatatypeDaemon::parseNextValue ( )

Diese Funktion wird von DBPacketInsert aufgerufen.

Sie holt sich ein Data Objekt vom Decoder und extrahiert daraus die Nutzdaten und die Position dieser Daten im Paket. Die Position wird dabei in einer Membervariablen zwischengespeichert. Nachdem der Datentyp bestimmt wurde, wird eine Divison durch eine Zehnerpotenz entsprechend dem jetzt erst extrahierten Wert aus Data durchgeführt, um den Wert auf seine ursprüngliche Größe zurückzubringen. Dies wird jedoch nur ausgeführt, wenn der Wert vor der Übertragung ein Gleitkommawert war. Abschließend wird der Wert in einen String umgewandelt, der dann an den Aufrufer zurückgegeben wird.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- DatatypeDaemon.h
- DatatypeDaemon.cpp

3.4 dbData Klassenreferenz

Klasse zum Auslesen der Zugangsdaten für die Datenbank.

`#include <dbData.h>`

Öffentliche Methoden

- dbData ()
- string getHost ()

  Memberfunktion, die die Hostadresse bereitstellt.
- string getUser ()

  Memberfunktion, die den Benutzernamen für die Datenbank bereitstellt.
- string getPW ()

  Memberfunktion, die das Passwort der Datenbank bereitstellt.
- string getDB ()

  Memberfunktion, die den Namen der Datenbank bereitstellt.

3.4.1 Ausführliche Beschreibung

Klasse zum Auslesen der Zugangsdaten für die Datenbank.

Diese Klasse liest die Zugangsdaten aus einer Textdatei (dbconfig.txt) aus und stellt sie über Memberfunktionen bereit.

Siehe auch

Insert

3.4.2 Beschreibung der Konstruktoren und Destruktoren

3.4.2.1 dbData::dbData ( )

Konstruktor der Klasse. Hier werden die Zugangsdaten aus der Textdatei in die entsprechenden lokalen Variablen eingelesen.

3.4.3 Dokumentation der Elementfunktionen

3.4.3.1 string dbData::getDB ( )

Memberfunktion, die den Namen der Datenbank bereitstellt.

Rückgabe

Der Name der Datenbank.

3.4.3.2 string dbData::getHost ( )

Memberfunktion, die die Hostadresse bereitstellt.

Wird von Insert aufgerufen.

Rückgabe

Die Hostadresse.

3.4.3.3 string dbData::getPW ( )

Memberfunktion, die das Passwort der Datenbank bereitstellt.

Rückgabe

Das Passwort für die Datenbank und den Nutzer.

3.4.3.4 string dbData::getUser ( )

Memberfunktion, die den Benutzernamen für die Datenbank bereitstellt.

Rückgabe

Der Nutzername.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- dbData.h
- dbData.cpp

3.5 DBPacketInsert Klassenreferenz

```
#include <DBPacketInsert.h>
```

Öffentliche Methoden

- DBPacketInsert ()

  Konstruktor der Klasse.
- void db_insert ()

3.5.1 Ausführliche Beschreibung

Klasse zum Aufbau eines SQL-Strings.

Diese Klasse baut einen SQL-String mit Hilfe anderer Klassen auf, mit dem schließlich die Daten in die Datenbank geschrieben werden.

Siehe auch

DatatypeDaemon
Insert

3.5.2 Dokumentation der Elementfunktionen

3.5.2.1 void DBPacketInsert::db_insert ( )

Memberfunktion zum Aufruf aus anderen Klassen.

Diese Funktion löst den Aufruf von Unterfunktionen aus, die dann den SQL-String modular aufbauen und schließlich an Insert übergeben. Gleichzeitig bietet sie die Schnittstelle zu den anderen Teilen des Programms.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- DBPacketInsert.h
- DBPacketInsert.cpp

3.6 Decoder Klassenreferenz

Öffentliche Methoden

- Decoder (char ∗buffer, const int bufferlen)
- Decoder (char ∗buffer, const int bufferlen, char ∗vecLayout, const int vecLayoutlen, char ∗vecDatatypes, const int vecDatatypeslen, char ∗vecComma, const int vecCommalen)
- Data getNextData ()
- unsigned int getPackageNum ()
- unsigned int getPackagePos (char ∗vecLayout, const int vecLayoutlen)

3.6.1 Beschreibung der Konstruktoren und Destruktoren

3.6.1.1 Decoder::Decoder (char * *buffer,* const int *bufferlen*)

Erzeugt einen Dekoder der zum dekodieren der Paketinformation dient.

Parameter

buffer	Speicher der die Paketinformationen enthält. [Layout,Datentypen,Kommasetzung]
bufferlen	Länge von *buffer*.

3.6.1.2 Decoder::Decoder (char * *buffer,* const int *bufferlen,* char * *vecLayout,* const int *vecLayoutlen,* char * *vecDatatypes,* const int *vecDatatypeslen,* char * *vecComma,* const int *vecCommalen*)

Erzeugt einen Dekoder der ein Datenpaket anhand der übergebenen Informationen dekodiert.

Parameter

buffer	Speicher des Datenpakets.
buffernlen	Länge von *buffer*.
vecLayout	Aufteilung des ursprünglichen Datenstroms die aus den Paketinformationen dekodiert wurden. Dient zur Ermittlung der konkreten Datensätze.
vecLayoutlen	Länge von *vecLayout*.
vecDatatypes	Beinhaltet die Informationen zu den Datentypen der jeweiligen Datensätze.
vecDatatypeslen	Länge von *vecDatatypes*.
vecComma	Beinhaltet die Kommasetzung sämtlicher Datensätze.
vecCommalen	Länge von *vecComma*.

3.6.2 Dokumentation der Elementfunktionen

3.6.2.1 Data Decoder::getNextData ( )

Holt den nächsten Datensatz aus den empfangenen Daten.

Rückgabe

Gibt ein Datenobjekt Data zurück das sämtlich Informationen über den Datensatz enthält.

3.6.2.2 unsigned int Decoder::getPackageNum ( )

Holt die Paketnummer des akutell bearbeiteten Pakets.

Rückgabe

Paketnummer das aktuellen Pakets

3.6.2.3 unsigned int Decoder::getPackagePos (char * *vecLayout,* const int *vecLayoutlen*)

Holt die Position des aktuellen Pakets im ursprünglichen Datensatz.

Parameter

vecLayout	Aufteilung des ursprünglichen Datenstroms die aus den Paketinformationen dekodiert wurden.
vecLayoutlen	Länge von *vecLayout*.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Encoding.h
- Encoding.cpp

3.7 Encoder Klassenreferenz

`#include <Encoding.h>`

Öffentliche Methoden

- Encoder (const char ∗buffer, const int bufferlen, const char ∗vecLayout, const int vecLayoutlen, const char ∗vecDatatypes, const int vecDatatypeslen)
- int getPackage (char ∗package, size_t len, unsigned short packageNumber)
- int getNextPackage (char ∗package, size_t len)
- int getPackageSize (unsigned short packageNumber)
- unsigned int getPackageSum ()

3.7.1 Ausführliche Beschreibung

Service der aus einem kompletten Satz Fahrzeugdaten mehrere Pakete erzeugt und komprimiert. Die Komprimierung ist noch nicht implementiert.

3.7.2 Beschreibung der Konstruktoren und Destruktoren

3.7.2.1 Encoder::Encoder (const char ∗ *buffer,* const int *bufferlen,* const char ∗ *vecLayout,* const int *vecLayoutlen,* const char ∗ *vecDatatypes,* const int *vecDatatypeslen*)

Erzeugt einen Encoder.

Parameter

buffer	Die zu bearbeitenden Daten. Dabei muss es sich um einen Datenstrom handeln in dem jeweils 2 Byte einen Fahrzeugwert entsprechen.
bufferlen	Die Länge der zu bearbeitenden Daten.

vecLayout	Gibt an wie die Daten geteilt werden sollen. [Anfangsbyte Paket 1, Anfangsbyte Paket 2, ..., Anfangsbyte Paket n]
vecLayoutlen	Die Länge von *vecLayout*.
vecDatatypes	Gibt an um welchen Datentyp es sich jeweils handelt.
vecDatatypeslen	Die Länge von *vecDatatypes*.

3.7.3 Dokumentation der Elementfunktionen

3.7.3.1 int Encoder::getNextPackage (char * *package*, size_t *len*)

Holt das jeweils nächste Paket. (1,2,...,n,1,2,...)

Parameter

package	Speicher in den das Paket geschrieben werden soll.
len	Länge von *package*.

Rückgabe

Die Länge des Pakets oder -1 falls *len* zu klein.

3.7.3.2 int Encoder::getPackage (char * *package*, size_t *len*, unsigned short *packageNumber*)

Holt ein Paket mit einer speziellen Paketnummer.

Parameter

package	Speicher in den das Paket geschrieben werden soll.
len	Länge von *package*.
packageNumber	Paketnummer des gewünschten Pakets.

Rückgabe

Die Länge des Pakets oder -1 falls Paket mit *packageNumber* nicht vorhanden oder *len* zu klein.

3.7.3.3 unsigned int Encoder::getPackageSize (unsigned short *packageNumber*)

Gibt die Paketgröße eines speziellen Pakets zurück.

Parameter

packageNumber	Paketnummer dessen Größe gesucht ist.

Rückgabe

Größe des Pakets oder -1 falls Paket mit *packageNumber* nicht vorhanden.

3.7.3.4 unsigned int Encoder::getPackageSum ( )

Gibt die Anzahl der Pakete zurück.

Rückgabe

Anzahl der Pakete.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Encoding.h
- Encoding.cpp

3.8 Insert Klassenreferenz

Klasse zum Aufbauen einer Verbindung zur Datenbank und Einfügen von Daten.

`#include <Insert.h>`

Öffentliche Methoden

- Insert ()
- int insertIntoDB (string ∗anw)

3.8.1 Ausführliche Beschreibung

Klasse zum Aufbauen einer Verbindung zur Datenbank und Einfügen von Daten.

In dieser Klasse wird die eigentliche Kommunikation mit der MySQL-Datenbank realisiert. Hierzu wird der MySQL Connector/C++ benutzt. Die Zugangsdaten werden dafür mit Hilfe einer anderen Klasse aus einer Textdatei ausgelesen.

3.8.2 Beschreibung der Konstruktoren und Destruktoren

3.8.2.1 Insert::Insert ( )

Konstruktor der Klasse. Hier werden die Verbindungsdaten ermittelt.

3.8.3 Dokumentation der Elementfunktionen

3.8.3.1 int Insert::insertIntoDB (string ∗ *anw*)

Verbindung aufbauen und MySQL-String an die Datenbank schicken. Diese führt ihn dann aus.

Parameter

anw	ist ein Zeiger auf einen in DBPacketInsert aufgebauten SQL-String, der an diese Funktion übergeben wird.

Rückgabe

Integerwert, der aber nicht verwendet wird.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Insert.h
- Insert.cpp

3.9 Location Klassenreferenz

`#include <Location.h>`

Öffentliche Methoden

- Location (std::string address, short port)
- std::string getAddress ()
- int getPort ()

3.9.1 Ausführliche Beschreibung

Datenstruktur die Netzwerkdaten bestimmter Teilnehmer speichert.

3.9.2 Beschreibung der Konstruktoren und Destruktoren

3.9.2.1 Location::Location (std::string *address,* short *port*)

Erzeugt einen Teilnehmer.

Parameter

address	IP-Adresse des Teilnehmers.
port	Port-Nummer des Teilnehmers.

3.9.3 Dokumentation der Elementfunktionen

3.9.3.1 std::string Location::getAddress ( )

Rückgabe

Gibt die Adresse zurück.

3.9.3.2 int Location::getPort ( )

Rückgabe

Gibt die Portnummer zurück.

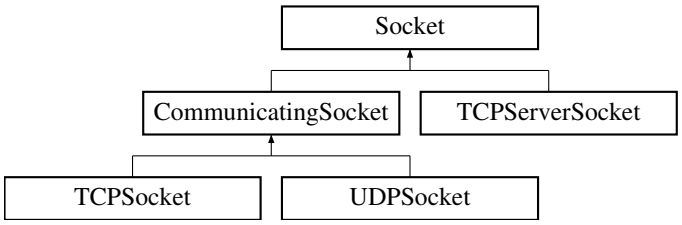
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Location.h
- Location.cpp

3.10 Socket Klassenreferenz

`#include <PracticalSocket.h>`

Klassendiagramm für Socket:



Öffentliche Methoden

- ~Socket ()
- string getLocalAddress () throw (SocketException)
- unsigned short getLocalPort () throw (SocketException)
- void setLocalPort (unsigned short localPort) throw (SocketException)
- void setLocalAddressAndPort (const string &localAddress, unsigned short localPort=0) throw (SocketException)

Öffentliche, statische Methoden

- static unsigned short resolveService (const string &service, const string &protocol="tcp")

Geschützte Methoden

- **Socket** (int type, int protocol) throw (SocketException)
- **Socket** (int sockDesc)

Geschützte Attribute

- int **sockDesc**

3.10.1 Ausführliche Beschreibung

Basisklasse, welche den Endpunkt der grundlegenden Kommunikation darstellt.

3.10.2 Beschreibung der Konstruktoren und Destruktoren

3.10.2.1 Socket::∼Socket ( )

Schließt und deallociert diesen Socket.

3.10.3 Dokumentation der Elementfunktionen

3.10.3.1 string Socket::getLocalAddress ( ) throw SocketException)

Holt die lokale Adresse.

Rückgabe

local Adresse des Sockets.

Ausnahmebehandlung

Wirft	SocketException falls das Holen fehlschlägt.

3.10.3.2 unsigned short Socket::getLocalPort ( ) throw SocketException)

Holt den lokalen Port.

Rückgabe

Lokalen Port des Sockets.

Ausnahmebehandlung

SocketException	wird geworfen falls das Holen fehlschlägt.

3.10.3.3 unsigned short Socket::resolveService (const string & *service*, const string & *protocol* = "tcp") [static]

Ermittelt den spezifizierten Service für das angegebene Protokoll.

Parameter

service	Zu ermittelnde Service (e.g., "http").
protocol	Protokoll des zu ermittelnden Service. Standard ist "tcp".

3.10.3.4 void Socket::setLocalAddressAndPort (const string & *localAddress,* unsigned short *localPort* = 0) throw SocketException)

Setzt den lokalen Port und die lokale Adresse. Wenn kein Port angegeben wird, wird ein zufälliger gewählt.

Parameter

localAddress	Lokale Adresse.
localPort	Lokaler Port.

Ausnahmebehandlung

SocketException	wird geworfen falls das Setzen des lokalen Ports oder der Adresse fehlschlägt.

3.10.3.5 void Socket::setLocalPort (unsigned short *localPort*) throw SocketException)

Setzt den lokalen Port zu einem gewählten Port und die lokale Adresse zu einer beliebigen Schnittstelle.

Parameter

localPort	Lokaler Port

Ausnahmebehandlung

SocketException	wird geworfen wenn das Setzen des lokalen Ports fehlschlägt.

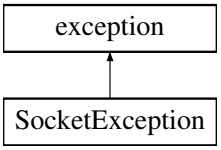
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- PracticalSocket.h
- PracticalSocket.cpp

3.11 SocketException Klassenreferenz

`#include <PracticalSocket.h>`

Klassendiagramm für SocketException:



Öffentliche Methoden

- SocketException (const string &message, bool inclSysMsg=false) throw ()
- ~SocketException () throw ()
- const char * what () const throw ()

3.11.1 Ausführliche Beschreibung

Zeigt ein Problem mit der Ausführung eines Socketaufrufs an.

3.11.2 Beschreibung der Konstruktoren und Destruktoren

3.11.2.1 SocketException::SocketException (const string & *message*, bool *inclSysMsg* = `false`) throw ( )

Erzeugt eine "SocketException" mit einem erklärenden Hinweistext.

Parameter

message	Nachricht die den Fehlern beschreibt
incSysMsg	true falls eine Systemnachricht (von strerror(errno)) sollte zu der für den User bereitgestellten Nachricht hinzugefügt werden.

3.11.2.2 SocketException::~SocketException ( ) throw ( )

Nur bereitgestellt um zu gewährleisten, dass keine Exceptions zurückgeworfen / zurückgegeben werden.

3.11.3 Dokumentation der Elementfunktionen

3.11.3.1 const char * SocketException::what ( ) const throw ( )

Holt die Exception-Nachricht.

Rückgabe

Exception-Nachricht.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- PracticalSocket.h
- PracticalSocket.cpp

3.12 T_nuex Strukturreferenz

Öffentliche Attribute

- short int **testen** [401]

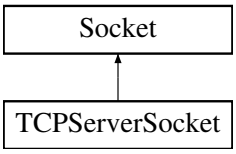
Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- mab.cpp

3.13 TCPServerSocket Klassenreferenz

`#include <PracticalSocket.h>`

Klassendiagramm für TCPServerSocket:



Öffentliche Methoden

- TCPServerSocket (unsigned short localPort, int queueLen=5) throw (SocketException)
- TCPServerSocket (const string &localAddress, unsigned short localPort, int queueLen=5) throw (SocketException)
- TCPSocket ∗ accept () throw (SocketException)

Weitere Geerbte Elemente

3.13.1 Ausführliche Beschreibung

TCP-Socket Klasse für Server.

3.13.2 Beschreibung der Konstruktoren und Destruktoren

3.13.2.1 TCPServerSocket::TCPServerSocket (unsigned short *localPort,* int *queueLen* = 5) throw SocketException)

Erzeugt einen TCP-Socket für die Nutzung mit dem Server, welcher zu einer beliebigen Schnittstelle auf dem vereinbarten Port Verbindungen zulässt.

Parameter

localPort	Lokaler Port des Server.
queueLen	Maximale Warteschlangenlänge für ausstehende Verbindungsanfragen. (default 5)

Ausnahmebehandlung

SocketException	wird geworfen falls es nicht möglich ist einen Socket zu erzeugen.

3.13.2.2 TCPServerSocket::TCPServerSocket (const string & *localAddress,* unsigned short *localPort,* int *queueLen* = 5) throw SocketException)

Erzeugt einen TCP-Socket für die Nutzung mit dem Server, welcher zu einer beliebigen Schnittstelle zu einer vereinbarten Adresse Verbindungen zulässt.

Parameter

localAddress	Lokales Interface (Adresse) des Server-Sockets.
localPort	Lokaler Port des Servers.
queueLen	Maximale Warteschlangenlänge für ausstehende Verbindungsanfragen. (default 5)

Ausnahmebehandlung

SocketException	wird geworfen falls es nicht möglich ist einen Socket zu erzeugen.

3.13.3 Dokumentation der Elementfunktionen

3.13.3.1 TCPSocket * TCPServerSocket::accept ( ) throw SocketException)

Blockiert solange bis eine neue Verbindung auf diesem Socket etabliert wurde oder ein Fehler auftritt.

Rückgabe

Neue Socketverbindung

Ausnahmebehandlung

SocketException	wird geworfen falls der Versuch eine neue Verbindung zu erzeugen fehlschlägt.

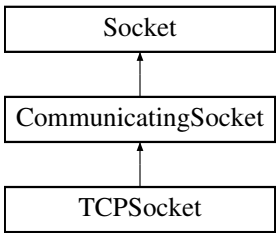
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- PracticalSocket.h
- PracticalSocket.cpp

3.14 TCPSocket Klassenreferenz

```
#include <PracticalSocket.h>
```

Klassendiagramm für TCPSocket:



Öffentliche Methoden

- TCPSocket () throw (SocketException)
- TCPSocket (const string &foreignAddress, unsigned short foreignPort) throw (SocketException)

Freundbeziehungen

- class **TCPServerSocket**

Weitere Geerbte Elemente

3.14.1 Ausführliche Beschreibung

TCP-Socket für die Kommunikation mit anderen TCP-Sockets.

3.14.2 Beschreibung der Konstruktoren und Destruktoren

3.14.2.1 TCPSocket::TCPSocket ( ) throw SocketException)

Erzeugt einen TCP-Socket mit keiner Verbindung.

Ausnahmebehandlung

SocketException	wird geworfen falls die Erzeugung fehlschlägt.

3.14.2.2 TCPSocket::TCPSocket (const string & *foreignAddress,* unsigned short *foreignPort*) throw SocketException)

Erzeugt einen TCP-Socket mit einer Verbindung zu einer bestimmten Adresse und einem bestimmten Port.

Parameter

foreign-Address	foreign address (IP address or name)
foreignPort	foreign port

Ausnahmebehandlung

SocketException	wird geworfen falls die Erzeugung fehlschlägt.

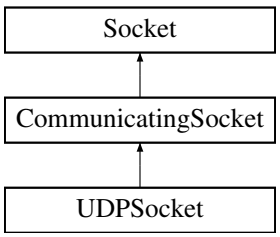
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- PracticalSocket.h
- PracticalSocket.cpp

3.15 UDPSocket Klassenreferenz

`#include <PracticalSocket.h>`

Klassendiagramm für UDPSocket:



Öffentliche Methoden

- UDPSocket () throw (SocketException)
- UDPSocket (unsigned short localPort) throw (SocketException)
- UDPSocket (const string &localAddress, unsigned short localPort) throw (SocketException)
- void disconnect () throw (SocketException)
- void sendTo (const void ∗buffer, int bufferLen, const string &foreignAddress, unsigned short foreignPort) throw (SocketException)
- int recvFrom (void ∗buffer, int bufferLen, string &sourceAddress, unsigned short &sourcePort) throw (SocketException)
- void setMulticastTTL (unsigned char multicastTTL) throw (SocketException)
- void joinGroup (const string &multicastGroup) throw (SocketException)
- void leaveGroup (const string &multicastGroup) throw (SocketException)

Weitere Geerbte Elemente

3.15.1 Ausführliche Beschreibung

UDP-Socket Klasse

3.15.2 Beschreibung der Konstruktoren und Destruktoren

3.15.2.1 UDPSocket::UDPSocket ( ) throw SocketException)

Erzeugt einen UDP-Socket.

Ausnahmebehandlung

SocketException	wird geworfen falls die Erzeugung fehlschlägt.

3.15.2.2 UDPSocket::UDPSocket (unsigned short *localPort*) throw SocketException)

Erzeugt einen UDP-Socket mit einem spezifischen Port.

Parameter

localPort	Lokaler Port

Ausnahmebehandlung

SocketException	wird geworfen falls die Erzeugung fehlschlägt.

3.15.2.3 UDPSocket::UDPSocket (const string & *localAddress,* unsigned short *localPort*) throw SocketException)

Erzeugt einen UDP-Socket mit einer spezifischen Adresse und einem gegebenen Port.

Parameter

localAddress	Lokale Adresse.
localPort	Lokaler Port.

Ausnahmebehandlung

SocketException	wird geworfen falls die Erzeugung fehlschlägt.

3.15.3 Dokumentation der Elementfunktionen

3.15.3.1 void UDPSocket::disconnect ( ) throw SocketException)

Setze Adresse und Port zurück.

Rückgabe

true falls kein Fehler auftrat.

Ausnahmebehandlung

SocketException	wird geworfen falls eine Trennung fehlschlägt.

3.15.3.2 void UDPSocket::joinGroup (const string & *multicastGroup*) throw SocketException)

Tritt der angegebenen Multicast-Gruppe bei.

Parameter

multicast-Group	Adresse der Mutlicast-Gruppe.

Ausnahmebehandlung

SocketException	wird geworfen falls das Beitreten fehlschlägt.

3.15.3.3 void UDPSocket::leaveGroup (const string & *multicastGroup*) throw SocketException)

Verlasse die angegebene Multicast-Gruppe.

Parameter

multicast-Group	Zu verlassende Multicast-Gruppe.

Ausnahmebehandlung

SocketException	wird geworfen falls das Verlassen fehlschlägt.

3.15.3.4 int UDPSocket::recvFrom (void * *buffer,* int *bufferLen,* string & *sourceAddress,* unsigned short & *sourcePort*) throw SocketException)

Empfängt eine Nachricht an einem UDP-Socket.

Parameter

buffer	Buffer in den gelesen werden soll.
bufferLen	Maximale Anzahl an Bytes die empfangen werden sollen.
source-Address	Adresse von der die Nachricht stammt.
sourcePort	Portnummer des Senders.

Rückgabe

Anzahl der Bytes die empfangen wurden, -1 falls ein Fehler auftrat.

Ausnahmebehandlung

SocketException	wird geworfen falls das Empfangen fehlschlägt.

3.15.3.5 void UDPSocket::sendTo (const void * *buffer*, int *bufferLen*, const string & *foreignAddress*, unsigned short *foreignPort*) throw SocketException)

Sendet einen Buffer als UDP-Datagramm an eine bestimmte Adresse und Portnummer.

Parameter

buffer	Buffer der gesendet werden soll.
bufferLen	Anzahl der Bytes die geschrieben werden sollen.
foreign-Address	Adresse an die versendet werden soll.
foreignPort	Portnummer an die versendet werden soll.

Rückgabe

true falls Versandt geklappt hat.

Ausnahmebehandlung

SocketException	wird geworfen falls das Senden fehlschlägt.

3.15.3.6 void UDPSocket::setMulticastTTL (unsigned char *multicastTTL*) throw SocketException)

Setzt Multicast TTL.

Parameter

multicastTTL	Multicast TTL.

Ausnahmebehandlung

SocketException	wird geworfen falls das Setzen fehlschlägt.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- PracticalSocket.h
- PracticalSocket.cpp

Index