**Praesibert – Netzwerkschicht**

**Konzept:**

Die Netzwerkschicht der Praesibert-Applikation besteht grundsätzlich aus zwei Typen von Socket-Klassen. Einer *ClientSocket* Klasse für die Nutzerseite und einer *ServerSocket* Klasse für die Server-Applikation. Zusätzlich benutzt die *ServerSocket* Klasse eine weitere Klasse *ConnectedClient*, von der für jeden neu verbundenen Client ein neues Objekt erzeugt wird.

Beide Sockets sind asynchron implementiert und können mittels Signalsund Slots Daten aufnehmen und ausgeben. Als Kommunikationsprotokoll wird TCP verwendet.  
Sowohl *ServerSocket* als auch *ClientSocket* besitzen jeweils zwei Arten von Sockets für die Datenübertragung. Einen Socket zum Senden und Empfangen von Kommandos und einen Socket zum Übermitteln von Datenpaketen. So soll verhindert werden, dass Kommandos durch zu lange Datenpakete zu spät oder gar nicht empfangen werden.  
Für jede der Übertragungsarten wird ein eigener Port verwendet.

**Die ClientSocket Klasse:**

Mithilfe der *ClientSocket* Klasse lassen sich Kommando- und Daten-Socket für einen Client erstellen.  
Sie initialisiert zwei Arten von Sockets (Kommandos und Daten) vom Typ *QTcpSocket*.

Herstellen einer Verbindung:

Um sich mit einem Server zu verbinden kann die Funktion *connectToServer()* verwendet werden.  
Als Parameter werden sowohl die IP-Addresse als auch die beiden Ports für Daten und Kommandos im *QString* Format übergeben.  
Die Methode stellt dann eine Verbindung zu dem Server an der gewünschten Adresse her.  
Gewöhnlich verbindet sich der Kommando-Socket zuerst, allerdings ist der Server so konstruiert, dass es unabhängig ist, welcher Socket sich zuerst verbindet.  
Bei einem Fehler liefert die Methode den Wert **false** zurück.  
Wird die Verbindung erfolgreich hergestellt, so liefert die Methode den Wert **true** zurück.

Senden von Daten:

Über die Funktionen *sendCmd()* und *sendData()* lassen sich Kommandos oder Daten im *QByteArray* Format an den Server schicken, mit dem der Client verbunden ist.  
Die Funktion liefert als Rückgabewert die Anzahl der versendeten Daten in Byte.

Empfangen von Daten:

Die Klasse stellt die Signale *receivedCmd()* und *receivedData()* zur Verfügung.  
Je nachdem welche Art von Daten empfangen wurden, wird eines der Signale ausgelöst.  
Als Übergabeparameter werden die zuletzt am Socket empfangenen Daten in *QByteArray* Format übergeben.

Verbindung zum Server trennen:

Über den Slot *disconnectFromServer()* kann die aktuelle Verbindung zum Server geschlossen werden.

Verbindung zum Server verlieren:

Die Klasse stellt das Signal *lostConnection()* zur Verfügung.  
Sollte die Verbindung eines Sockets zum Server abbrechen oder beendet werden, werden beide Verbindungen geschlossen und dieses Signal ausgelöst.

**Die ServerSocket Klasse:**

Mithilfe der *ServerSocket* Klasse lässt sich ein Socket für die Server Applikation erstellen.  
Sie initialisiert zwei Arten von Servern (Kommandos und Daten) vom Typ *QTcpServer*.

Für jeden Client, welcher sich mit dem Server verbindet, wird ein Object der Klasse *ConnectedClient* erstellt. Diese Klasse enthält wiederum zwei Sockets über welchen der Server mit dem jeweiligen Client kommunizieren kann.

Auf Verbindungen von Clients warten:

Über den Slot *beginListening*() wird der Server initialisiert und für eingehende Verbindungen vorbeireitet. Als Parameter müssen die beiden Ports für Kommandos und Daten im *QString* Format übergeben werden.  
Nachdem die Routine abgearbeitet ist, wird von der Funktion das Signal *newIP()* ausgelöst, welches die aktuelle IP des Servers im *QString* Format zur Verfügung stellt.  
Konnte keine korrekte IP ermittelt werden, so wird *Localhost* als neue IP ausgegeben.

Neue Clients verbinden sich mit dem Server:

Die Klasse stellt den Slot *handleNewConnection()* zur Verfügung, welcher mit den Signalen *newConnection()* der beiden Server-Sockets verbunden ist.  
Verbindet sich ein neuer Client mit dem Server, überprüft die Routine welcher der beiden Server-Sockets das Signal ausgelöst hat.  
Dementsprechend wird ein neues *QTcpSocket*-Objekt, welches die jeweilige Verbindung zum Client darstellt, erzeugt und in ein neu erstelltes Objekt der Klasse *ConnectedClient* übergeben. Ist unter der IP-Adresse des Client schon der jeweils andere Socket verfügbar, wird das *QTcpSocket* Objekt an das *ConnectedClient* Objekt mit dieser IP-Adresse übergeben.

Für jedes der erstellten *ConnectedClient* Objekte wird ein eigener Thread angelegt.  
Wird der Thread gestoppt oder der Server geschlossen, wird der Thread über verschiedene Verbindungen von Signals und Slots terminiert.  
Alle Clients werden innerhalb der *ServerSocket* Klasse in einer Liste (*QList*) gespeichert.

Jeder Thread bzw. Socket erhält eine individuelle ID, über die jeder Socket eindeutig identifizierbar ist.

Nachrichten von Clients erhalten:

Jeder Client kann das Signal *newCmd()* oder *newData()* mit den Daten im *QByteArray* Format und der ID als Parameter aussenden. Diese werden in der Routine *handleNewConnection()* mit dem Signal *receivedCmdFromClient()* bzw. *receivedDataFromClient()* der *ServerSocket* Klasse verbunden, sodass diese bei neuen Nachrichten automatisch ausgesendet werden.

Nachrichten an Clients senden:

Über die Funktion *sendCmdToAll()* oder *sendDataToAll()* mit den Daten im *QByteArray* Format als Parameter lassen sich Kommandos oder Daten zu allen, mit dem Server verbundenen Clients, senden.

Über die Funktion *sendCmdToID()* oder *sendDataToID()* mit den Daten im *QByteArray* Format und der *clientID* als Parameter lassen sich Kommandos oder Daten zu einem bestimmten, mit dem Server verbundenen, Client senden.

Die Verbindung zu einem Client verlieren:

Jeder verbundene Client sendet das Signal *disconnected()* mit der ID als Parameter aus, wenn die Verbindung einer der Sockets unterbrochen worden sein sollte.  
Über die Routine des Slots *handleClientDisconnect()* entfernt der Server den Client aus seiner Liste und löst das Signal *clientDisconnect()* mit der Client ID als Parameter aus.

Den Server beenden:

Über die Funktion *closeServer()* lässt sich der Server stoppen.  
Mit dem Aufruf der Funktion wird jede Verbindung zu einem Client geschlossen.  
Anschließend wird der Server gestoppt und das Signal *stoppedServer()* ausgelöst.

**Die ConnectedClient Klasse:**

Die Klasse *ConnectedClient* wird genutzt, um für jeden Verbundenen Client ein eigenes Objekt mit einem *QTcpSocket* zu erzeugen.  
Ein Objekt dieser Klasse erhält kein *QObject* als *parent*, damit sich das Objekt in einen eigenen Thread schieben und ausführen lässt.  
Jedes Objekt der Klasse erhält über seinen Konstruktor eine *Client ID*, über die der Client eindeutig zugeordnet werden kann.

Neuer Client verbindet sich mit dem Server:

Nach dem Erzeugen des Objekts und Verschieben in einen eigenen Thread, wird durch den Aufruf von *start()* im Thread der Slot *process()* aufgerufen.  
In der Routine wird überprüft, welche Art von Socket bereits verfügbar ist und die Signals und Slots des jeweiligen Sockets mit den Handler-Routinen der *ConnectedClient* Klasse verbunden.

Neue Daten von Client sind verfügbar:

Wenn neue Kommandos oder Daten von einem Socket zur Verfügung stehen, wird von diesem Socket das Signal *readyRead()* ausgelöst.  
Über den Slot *handleCmdRead()* oder *handleDataRead()* werden die Daten gelesen und anschließend das Signal *newCmd()* bzw. *newData()* ausgelöst. Dieses haben die gelesenen Daten im *QByteArray* Format und die Client IDals Parameter.

Daten zum Client senden:

Über die Funktion *sendCmd()* bzw. *sendData()* mit den zu sendenden Daten im *QByteArray* als Parameter lassen sich Kommandos oder Daten zu dem Client senden.  
Die Funktion übergibt die Anzahl der gesendeten Daten in Byte als Rückgabeparameter.

Die Verbindung zum Client beenden:

Die Funktion *disconnectFromServer()* beendet die Verbindung zwischen Client und Server.  
Es wird anschließend das Signal *finished()* ausgelöst, um den Thread zu beenden in dem der Client läuft.

Die Verbindung zum Client verlieren:

Sollte die Verbindung zu einem der Sockets des Client unterbrochen werden, wird von dem jeweiligen Socket das Signal *disconnected()* ausgelöst.  
Über den Slot *handleDisconnect()* werden anschließend beide Sockets geschlossen und das Signal *disconnected()* mit der Client ID als Parameter und das Signal *finished()* ausgelöst.

**UML-Diagramm:**

