Verwendung und Aufbau der Bluetooth-API für die Android-Entwicklung

Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt

Tobias Hahn

Version 1

Inhaltsverzeichnis

1. Allgemeines

Um in Verbindung mit der Bluetooth-API Android-Anwendungen entwickeln zu können werden allgemeine Kenntnisse über die Java und Android-Entwicklung benötigt. Außerdem sind Kenntnisse über die Programmierung des ROBO TX Controllers nötig.

Dieses Dokument bietet eine Einführung in die Verwendung der API und deren Aufbau. Außerdem wird ein kurzer Abriss über die Verwendung von Bluetooth in Android gegeben.

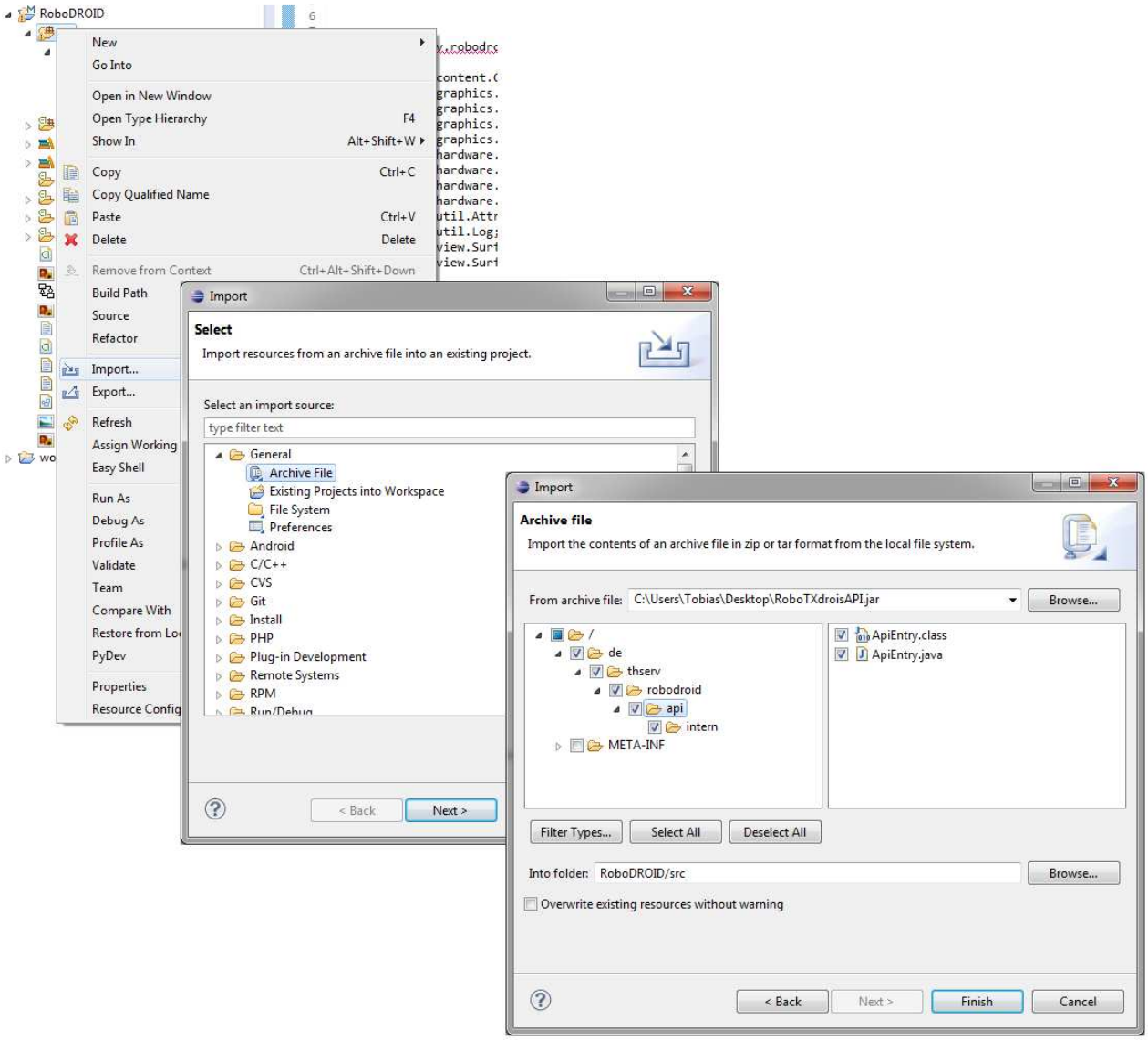
Die API verfügt über zwei Modi. Zum einen über den Onlinemodus und zum anderen über den Downloadmodus. Im Onlinemodus übernimmt die Android-Anwendung die Steuerung der ROBO TX Controller und es wird kein zusätzliches Programm auf dem Controller benötigt. Im Downloadmodus hingegen besteht keine aktive Verbindung zum ROBO TX Controller und es wird ein C-Programm benötigt, welches die Nachrichten vom Smartphone interpretiert und die Steuerung übernimmt.

Der Onlinemodus kann ab der Firmware 1.24 verwendet werden, um den Downloadmodus nutzen zu können wird auf dem ROBO TX Controller die Firmware mit der Version 1.32 verwendet werden.

Weiterführende Informationen zu der Entwicklung für den ROBO TX Controller bieten auch die durch die Firma fischertechnik zur Verfügung gestellten Dokumente.

2. Importieren der API

Die API wird einfach in das Projekt der Android-Anwendung eingebunden. Die Abbildung zeigt die Abfolge zum Importieren der API.



Hier ist zu beachten, dass vorher der Ordner ausgewählt werden muss, in welchen die API importiert werden soll. Im nächsten Schritt selektiert man die Form ›Archive File‹, in welcher die zu importierenden Daten zur Verfügung stehen. Als letzten Schritt wählt man die zu importierenden Daten aus. Hier kann der Ordner ›META-INF‹ Vernachlässigt werden, denn dieser enthält lediglich Meta-Informationen für das Package und ist für das Verwenden der API nicht notwendig.

3. Ermitteln der MAC-Adresse

Um in einer Android-Applikation die Android eigenen Bluetooth-APIs nutzen zu können, müssen im Manifest die benötigten Berechtigungen gesetzt werden.

```
<manifest xmlns:android=http://schemas.android.com/apk/res/android …>
    …
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    …
</manifest>
```

Bedeutung der verwendeten Berechtigungen

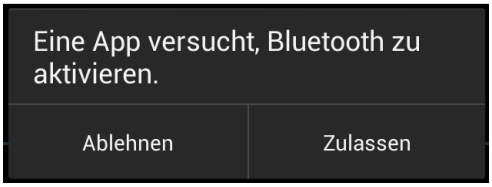
- BLUETOOTH: Diese erlaubt es, die grundlegendsten Funktionen der Bluetooth-Kommunikation zu nutzen. Darunter fallen das Anfragen einer Verbindung, das Akzeptieren einer eingehenden Verbindung und die Datenübertragung.
- BLUETOOTH_ADMIN: Durch diese ist es möglich, eine Suche nach anderen Bluetooth-Geräten zu starten oder die Bluetooth-Einstellungen zu verändern.

Auch wenn man davon ausgehen kann, dass eigentlich jedes Smartphone Bluetooth besitzt, sollte man dennoch überprüfen, ob dies auch der Fall ist.

```
BluetoothAdapter blueAdapter = BluetoothAdapter.getDefaultAdapter();
if (blueAdapter == null) {
      // Kein Bluetooth verfügbar.
}
```

Der oben stehende Code überprüft, ob das Smartphone über Bluetooth verfügt. Durch die Methode getDafaultAdapter(), bekommt man vom System den BluetoothAdapter. Dieser stellt die Repräsentation des eigenen Bluetooth-Gerätes dar. Wenn man als Rückgabewert ein „NULL“ erhält, besitzt das Gerät kein Bluetooth.

Wenn das Gerät über Bluetooth verfügt, muss sichergestellt werden, dass es auch zum Zeitpunkt der Nutzung aktiviert ist. Die Aktivierung muss jedoch nochmals explizit durch den Nutzer bestätigt werden.



Sollte Bluetooth noch nicht aktiv sein, wird ein Intent mit der ACTION_REQUEST_ENABLE Aktion erzeugt, welche per startActivityForResult()-Methode ins System abgesetzt wird.

```
if (!blueAdapter.isEnabled()) {
      Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
      startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

Dadurch das öffnet sich ein Dialog, wie in der Abbildung, durch welchen der Nutzer gefragt wird, ob Bluetooth aktiviert werden soll, oder nicht.

Durch startActivityForResult() wird nach Abarbeitung des Intents bzw. nach der Antwort des Nutzers, die Callback-Methode onActivityResult() aufgerufen. Durch Auslesen des ResultCodes, kann abgelesen werden, ob die Aktivierung erfolgreich war.

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
	if (resultCode == Activity.RESULT_OK) {
		// Bluetooth ist aktiv.
	} else {
		// Bluetooth wurde nicht aktiviert.
	}
}
```

Um eine Verbindung zu einem anderen Bluetooth-Gerät herzustellen, ist es zunächst notwendig, dass nach allen in der Nähe befindlichen Geräten gesucht wird und von diesen Informationen abgerufen werden. Diese Informationen beinhalten den Gerätenamen, dessen Klasse und seine MAC-Adresse. Anhand dieser Informationen ist es möglich, sich mit einem anderen Gerät zu „paaren“ (vom engl. Begriff pairing), das heißt, diese Geräte tauschen zum Zweck der Authentifizierung und Verschlüsselung einen Schlüssel aus. Alle mit dem eigenen Gerät gepaarten Geräte werden gespeichert und können durch die Bluetooth-API jederzeit abgerufen werden. So kann man anhand der MAC-Adresse jederzeit eine Verbindung zu einem Gerät herstellen, ohne eine neue Suche anstoßen zu müssen. Voraussetzung dafür ist natürlich, dass sich das gewünschte Gerät in Reichweite befindet.

Soll eine Verbindung zu einem bereits bekannten Gerät aufgebaut werden, so muss keine neue Suche nach anderen Geräten gestartet, da man alle Informationen auf dem eigenen Gerät finden kann.

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
	// wenn gepaarte Geräte existieren
	if (pairedDevices.size() > 0) {
		// Durchgehen der gepaarten Geräte
		for (BluetoothDevice device : pairedDevices) {
			// einem Array die Adresse und den Namen der Geräte
			hinzufügen
			mArrayAdapter.add(device.getName() + "\n" +
			device.getAddress());
		}
	}
```

Durch die Methode getBondedDevices(), wird die Liste aller gespeicherten und gepaarten Geräte abgerufen. Wenn man eine Verbindung zu einem dem eigenem Gerät noch unbekanntem Gerät, herstellen will, muss man zunächst nach diesem suchen. Dies erreicht man durch den Aufruf der Methode startDiscovery(). Der Aufruf dieser Methode löst eine asynchrone Suche aus.

Nach Beendigung dieser Suche wird die Aktion ACTION_FOUND in das System ausgesendet, das Ergebnis diesem zugänglich gemacht und kann anschließend abgefragt werden. Zu diesem Zweck benötigt man einen BroadcastReceiver, der auf das Ergebnis der Suche reagiert.

```
// BroadcastReceiver für ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
	public void onReceive(Context context, Intent intent) {
		String action = intent.getAction();
		// wenn durch die Suche ein Gerät gefunden wurde
		if (BluetoothDevice.ACTION_FOUND.equals(action)) {
			// das Bluetooth-Gerät aus dem Intent holen
			BluetoothDevice device =
```

```
                        intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                    // Hinzufügen des Namens und der Adresse in ein Array
                    mArrayAdapter.add(device.getName() + "\n" +
                    device.getAddress());
                }
        }
};

// den BroadcastReceiver im System registrieren
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter);
```

Der Aufbau zu einem gefundenen Gerät erfolgt durch die API. Diese muss dafür lediglich die MAC-Adresse des gewünschten Gerätes übergeben bekommen.

4. Verwendung der API

4.1. Allgemeine Verwendung

Da nach dem Importieren bereits alle für die weitere Verwendung benötigten Schritte erledigt sind, kann jetzt die API in den Quellcode importiert werden, um sie dort zu verwenden.

```
import de.thserv.robodroid.api.*;
```

Die API ist als Singleton implementiert deshalb muss hier nicht extra eine Instanz erzeugt werden. Dies geschieht beim ersten Verwenden automatisch.

Es kann direkt mit dem Verwenden der API begonnen werden. Der folgende Quellcode verdeutlicht, dass ohne weitere Vorarbeiten eine MAC-Adresse an die API übergeben werden kann. Direkt danach kann versucht werden eine Verbindung zum ROBO TX Controller aufzubauen.

```
ApiEntry.getInstance().setMacAdress(this.getIntent().getExtras().getString("01:23:4
5:67:89:AB"));

ApiEntry.getInstance().open();
```

Der Status der Verbindung, bzw. des Verbindungsversuchs kann wie folgt beschrieben abgefragt werden. Sollte die Verbindung erfolgreich hergestellt sein, kann die TransferArea gestartet werden.

```
if (ApiEntry.getInstance().getConnectionStatus().equals("INVALID") == true) {
      // Kein ROBO TX Controller
} else if (ApiEntry.getInstance().getConnectionStatus().equals("NOT CONNECTED") ==
true) {
      // Verbindung felgeschlagen
} else if (ApiEntry.getInstance().getConnectionStatus().equals("CONNECTED") == true)
{
      // Verbindung Hergestellt
      ApiEntry.getInstance().startTransferArea();
}
```

Nachdem die TransferArea gestartet wurde, kann damit begonnen werden, steuernde Funktionen aufzurufen. Die Funktionen sind wie die der ftMscLib.dll strukturiert.

```
// SetOutPwmValues(int channel, short duty)

ApiEntry.getInstance().SetOutPwmValues(5, (short)512);
```

4.2. Funktionen der API

4.2.1. changeToDownloadMode()

Diese Funktion aktiviert den Downloadmodus. Ab diesem Zeitpunkt wird die weitere Kommunikation mit dem ROBO TX Controller im Downloadmodus ausgeführt. Ein Aufruf dieser Funktion sollte vor dem Herstellen einer Verbindung erfolgen.

4.2.2. changeToOnlineMode()

Diese Funktion aktiviert den Onlinemodus. Ab diesem Zeitpunkt wird die weitere Kommunikation mit dem ROBO TX Controller im Onlinemodus ausgeführt. Ein Aufruf dieser Funktion sollte vor dem Herstellen einer Verbindung erfolgen.

4.2.3. isOnlineMode()

Diese Funktion liefert den boolschen Wert, welcher zeigt ob sich die API im Onlinemodus befindet.

4.2.4. setMacAdress(String mac)

Diese Funktion übergibt der API die MAC-Adresse, zu welcher eine Verbindung aufgebaut werden soll.

Aufruf:	String mac	MAC-Adresse Beispiel: `"01:23:45:67:89:AB"`

4.2.5. getConnectionStatus()

Diese Funktion gibt den aktuellen Verbindungsstatus zurück.

Return:	String	`"INVALID"`
		`"NOT CONNECTED`
		`"CONNECTED"`

4.2.6. open()

Diese Funktion versucht eine Verbindung zum ROBO TX Controller herzustellen.

4.2.7. close()

Diese Funktion beendet eine aktive Verbindung zu einem ROBO TX Controller.

4.2.8. startTransferArea()

Diese Funktion aktiviert die TransferArea in der Library für den Online-Betrieb. Der Kommunikations-Thread wird gestartet und führt die I/O-Befehle im „Online-Modus“ durch. Dabei liest dieser die aktuellen Werte aus der Output-Struktur der TransferArea (Konfiguration- und Outputwerte) und sendet diese zum ROBO TX Controller. Als Antwort auf einen I/O-Request sendet der Controller die aktuellen

Werte und der Kommunikations-Thread aktualisiert diese dann in der Input-Struktur der TransferArea. Diese Funktion sollte erst nach dem erfolgreichen Herstellen einer Verbindung aufgerufen werden.

4.2.9. stopTransferArea()

Diese Funktion deaktiviert die Kommunikation der TransferArea mit dem ROBOTX Controller. Der Kommunikations-Thread wird gestoppt. Es findet keine Kommunikation mehr mit dem ROBO TX Controller statt.

4.2.10. doMovement(int left, int right)

Diese Funktion setzt in der TransferArea die Geschwindigkeiten für Motoren. Diese Funktion ist nur sinnvoll bei einem fahrenden Modell, welches die Motoren auf M1 und M2 angeschlossen hat.

Aufruf:	int left	Geschwindigkeit des Linken Motors
	Int right	Geschwindigkeit des Rechten Motors

4.2.11. StartCounterReset(int cnt_index)

Diese Funktion startet das Rücksetzen des Zählereingangs auf dem ROBO TX Controller auf 0. Dies ist ein asynchroner Vorgang und ist daher nicht direkt nach Rückkehr des Funktionsaufrufs beendet.

Aufruf:	int	Counterindex (0 – 3) der Counter 1 - 4

4.2.12. SetOutMotorValues(int motorId, int duty_p, int duty_m)

Diese Funktion setzt in der TransferArea für einen Motor die Duty-Werte für die beiden Motor-ausgänge M+ und M-.

Aufruf:	int motorId	Index des anzusteuernden Motors (0 - 3)
	int duty_p	Duty-Wert für den Motorausgang M+
	int duty_m	Duty-Wert für den Motorausgang M-

4.2.13. SetOutPwmValues(int channel, short duty)

Diese Funktion setzt in der TransferArea für einen PWM-Ausgang den angegebenen Duty-Wert.

Aufruf:	int channel	Index des PWM-Ausgang (0-7)
	short duty	Duty-Wert für den PWM-Ausgang

4.2.14. StartMotorExCmd(int mIdx, int duty, int mDirection, int sIdx, int sDirection, int pulseCnt)

Diese Funktion aktiviert den intelligenten Motor-Modus zur Motorsynchronisation. Der Motor fährt die angestrebte Position anhand der mitgeteilten Zählerinformation an.

Aufruf:	int mIdx	Motorindex (0 - 3) vom Master (-motor)
	int duty	Duty-Wert für Master/Slave-Motor
	int mDirection	Richtung für Master-Motor (0= CW, 1= CCW)

	int sIdx	Motorindex (0 – 3) vom Slave (-motor)
	int sDirection	Richtung für Slave-Motor (0= CW, 1= CCW)
	int pulseCnt	Anzahl der Zählimpulse zum Anfahren einer Position, relativ zur Ausgangsposition

4.2.15. StopAllMotorExCmd()

Diese Funktion stoppt unverzüglich den vorher aktivierten intelligenten Motor-Modus zur Motorsynchronisation für alle Motoren des ROBO TX Controllers durch Zurücksetzen der Duty-Werte auf 0, sowie die der Distance-Werte in der Output-Struktur.

4.2.16. StopMotorExCmd(int mtrIdx)

Diese Funktion stoppt unverzüglich den vorher aktivierten intelligenten Motor-Modus zur Motorsynchronisation für den angegebenen Motor durch Zurücksetzen der Duty-Werte sowie des Distance-Wertes in der Output-Struktur auf 0.

Aufruf:	int mtrIdx	Motorindex (0 - 3)

4.2.17. SetRoboTxMessage(String msg)

Diese Funktion generiert einen darstellbaren Text auf das Display des ROBO TX Controllers. Der Text kann auf dem Display nur dargestellt werden, wenn die TransferArea aktiviert ist. Die maximale Textlänge beträgt 98 Zeichen (ASCII). Bei der Übergabe eines leeren Texts (= 0), wird der aktuell angezeigte Text auf dem Display gelöscht und es erscheint wieder die Standardausgabe des ROBO TX Controllers. Alternativ kann über die beiden Taster auf dem Controller die angezeigten und gepufferten Meldungen gelöscht werden.

Aufruf:	String msg	Text, der auf dem Display dargestellt wird, max. 98 Zeichen

4.2.18. SetFtUniConfig(int ioId, int mode, boolean digital)

Diese Funktion konfiguriert einen Universaleingang(Kombi-Eingang) zur Messung von analogen und digitalen Spannungs- und Widerstandswerten und zur analogen Abstandsmessung.

Aufruf:	int ioId	Index des Universaleingangs (0 - 7)
	int mode	Art der Messung 0= Spannung (mV), 1= Widerstand (5 kΩ) 3= Ultraschallsensor (Abstandsmessung)
	boolean digital	Kennung ob der Wert digital zurückgegeben wird (FALSE= analog, TRUE= digital), bei der Abstandsmessung nur analog

4.2.19. SetFtCntConfig(int cntId, int mode)

Diese Funktion konfiguriert einen Zählereingang (Counter), wie der Zustand des Zählers zu interpretieren ist.

Aufruf:	int cntId	Counter-Index (0 - 3)
	Int mode	0= Mode NORMAL, 1= Mode INVERTED

4.2.20. SetFtMotorConfig(int motorId, boolean status)

Diese Funktion aktiviert oder deaktiviert die Motorausgänge. Analog dazu werden damit die PWM-Ausgänge entsprechend deaktiviert oder aktiviert.

Aufruf:	int motorId	Motorindex (0 - 3)
	Boolean status	TRUE = Motor an (Motorausgänge aktiviert)
		FALSE = Motor aus (PWM-Ausgang aktiviert)

4.2.21. GetInIOValue(int ioId)

Diese Funktion liest aus der TransferArea den aktuellen Wert eines Universaleingangs und stellt diesen einer Applikation zur Verfügung. Die Werte der Universaleingänge kommen als Antwort auf einen IO-Request vom ROBO TX Controller und werden in der TransferArea aktualisiert.

Aufruf:	int ioId	Index Universaleingang (0 - 7)
Return	int	Wert des Eingangs

4.2.22. GetInIOOverrun(int ioId)

Diese Funktion liest aus der TransferArea den aktuellen Wert eines Universaleingangs und stellt diesen einer Applikation zur Verfügung. Die Werte der Universaleingänge kommen als Antwort auf einen IO-Request vom ROBO TX Controller und werden in der TransferArea aktualisiert.

Aufruf:	int ioId	Index Universaleingang (0 - 7)
Return	int	Overrun-Meldung
		FALSE: kein Überlauf (Overrun)
		TRUE : Überlauf

4.2.23. GetInCounterValue(int cntId)

Diese Funktion liest aus der TransferArea den aktuellen Wert eines Zählereingangs (Counter) und stellt diesen einer Applikation zur Verfügung.

Aufruf:	int cntId	Index Zähler (Counter) (0 - 3)
Return:	int	Aktueller Zählerstand

4.2.24. GetInCounterState(int cntId)

Diese Funktion liest aus der TransferArea den aktuellen Status eines Zählereingangs (Counter) und stellt diesen einer Applikation zur Verfügung.

Aufruf:	int cntId	Index Zähler (Counter) (0 - 3)
Return:	int	Eingestellter Mode des Zählers
		TRUE : Mode "INVERTED"

FALSE: Mode "NORMAL"

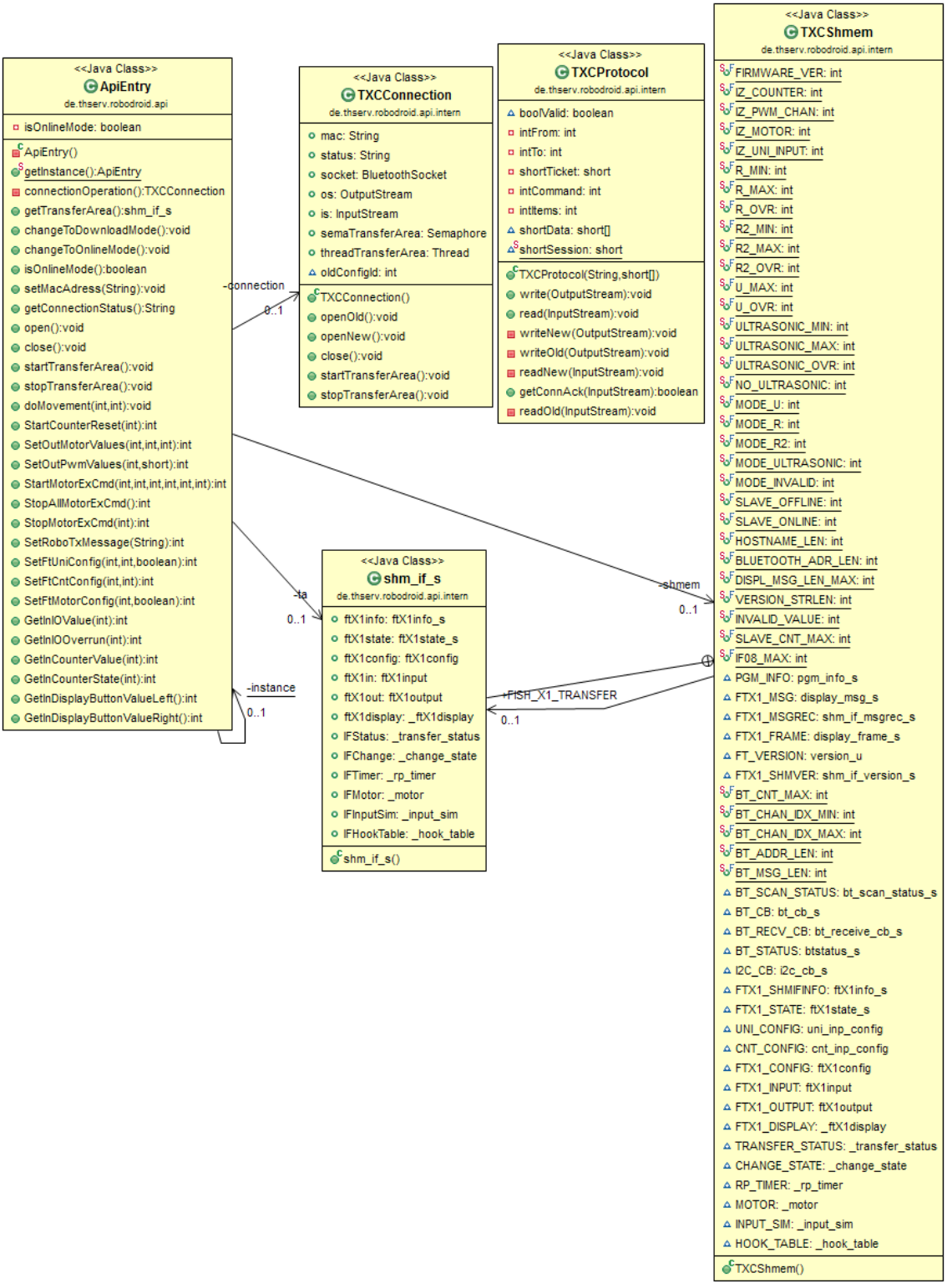
4.2.25. GetInDisplayButtonValueLeft()

Diese Funktion liest den aktuellen Status des Linken Taster auf dem Display. Angezeigt wird die Zeit, wie lange ein Taster gedrückt bleibt.

4.2.26. GetInDisplayButtonValueRight()

Diese Funktion liest den aktuellen Status des Rechten Taster auf dem Display. Angezeigt wird die Zeit, wie lange ein Taster gedrückt bleibt.

5. Interner Aufbau als Klassendiagramm



6. Aufbau des C-Programms im Downloadmodus

Das C-Programm für den Downloadmodus besteht im Wesentlichen aus 4 Bestandteilen. Die Methoden PrgInit und PrgTic sollten aus der Programmierung für den ROBO TX Controller bekannt sein. Neu ist hier nur, das eine Callback Funktion eingerichtet wird, welche aufgerufen wird, wenn eine neue Nachricht eintrifft.

```
void PrgInit(TA * p_ta_array,// pointer to the array of transfer areas
             int ta_count)// number of transfer areas in array (equal to TA_COUNT)

{
       TA * p_ta = &p_ta_array[TA_LOCAL];
       // get BT address of registered external message device
       p_ta->hook_table.GetExtBtDevAddr(bt_addr);
       // Start listen to the controller with bt_address via Bluetooth channel
       // BT_CHANNEL
       stage = START_LISTEN;
       command = CMD_START_LISTEN;
       command_status = -1;
       p_ta->hook_table.BtStartListen(BT_CHANNEL, bt_address, BtCallback);
}

int PrgTic(TA * p_ta_array,// pointer to the array of transfer areas
             int ta_count )// number of transfer areas in array (equal to TA_COUNT)
{
       int rc = 0x7FFF; // return code: 0x7FFF - program should be further called
                        // by the firmware;
                        // 0 - program should be normally stopped by the firmware;
                        // any other value is considered by the firmware as an
                        // error code and the program is stopped.
       TA * p_ta = &p_ta_array[TA_LOCAL];

       // Programm welches immer ausgeführt wird auch wenn keine Nachrichten
       // empfangen werden

       return rc;
}
```

Die Callback Funktion BtReceiveCallback erhält die empfangene Nachricht. Hier kann die Nachricht ausgewertet werden. Der Übersicht halber wird die empfangene Nachricht an die Funktion HandleRequest übergeben und dort bearbeitet.

```
static void BtReceiveCallback(TA * p_ta_array, BT_RECV_CB * p_data)
{
       if (p_data->status == BT_MSG_INDICATION) {
              TA * p_ta = &p_ta_array[TA_LOCAL];
              UCHAR8 rxmsg[256];
              int rxlen = 0;
              if (p_data->msg_len <= 256) {
                     p_ta->hook_table.memcpy(rxmsg, p_data->msg, p_data->msg_len);
                     rxlen = p_data->msg_len;
                     rxmsg[p_data->msg_len] = 0;
                     HandleRequest(p_ta, rxmsg, rxlen);
              }
              if (rxlen > 0) {
```

```
                    command_status = -1;
                }
        } else {
                receive_command_status = p_data->status;
        }
}
```

Innerhalb der Funktion HandleRequest werden die Nachrichten ausgewertet. Hier wird anhand des ersten Bytes in der Nachricht die nötige Aktion ausgeführt.

```
void HandleRequest(TA * p_ta, UCHAR8 rxmsg[], int rxlen)
{
        int code = rxmsg[0];
        switch (code)
        {
                case ECHO_REQUEST: {
                        //ECHO_REQUEST empfangen
                }
                break;
                case REM_IO_REQUEST: {
                        //REM_IO_REQUEST empfangen
                }
                break;
                case MSG_WR_REQUEST: {
                        //MSG_WR_REQUEST empfangen
                }
                break;
        }
}
```

Eine Antwort kann über den Aufruf

```
p_ta->hook_table.BtSend(BT_CHANNEL, len, msg, BtCallback);
```

gesendet werden.

7. Download des C-Programmes

Um das kompilierte C-Programm auf den ROBO TX Controller zu laden, muss dieser an den PC angeschlossen sein. Mit der zur Verfügung stehenden GUI ist es möglich komfortabel das entwickelte Programm auf den ROBO TX Controller zu laden. Hier muss die vom Compiler erzeugte *.bin-Datei, der COM-Port an welchen der Controller angeschlossen ist und das Ziel darauf ausgewählt werden.