Erstellen einer Bluetooth-Verbindung

Patrick Schwane

26. September 2013

Inhaltsverzeichnis

1 Einführung

Diese Anleitung beschreibt ein einfaches Programm mit dem man sich Kurznachrichten über Bluetooth zwischen zwei Android-Handys zuschicken kann.

Ziel es ist, zu zeigen, wie man eine Bluetooth-Verbindung zwischen zwei Geräten mit Android-Betriebssystem, innerhalb einer App herstellt und darüber Daten austauscht.

Als Voraussetzung benötigt man Kenntnisse in Java. Im geringen Umfang werden auch XML-Dateien verändert. Die Veränderungen sind jedoch eher klein, so dass sie auch ohne Kenntnisse in XML verstanden werden sollten.

Die beschriebene Beispiel-App wurde mit dem SDK-ADT Bundle for Windows in der 64 bit Version erstellt, einer speziellen Eclipse-Umgebung für die Entwicklung von Android-Apps, herunterzuladen unter: http://developer.android.com/sdk/index.html.

Natürlich funktioniert die Anleitung auch mit jeder anderen Eclipse-Version mit installiertem ADT (Android Development Tool). Bei der Verwendung einer anderen Version, kann es aber möglicherweise bei den Screenshots zu geringfügigen Abweichungen kommen.

2 Vorschau

Zunächst wird in der Anleitung beschrieben, wie man ein neues Android-Projekt in Eclipse anlegt. Darauf folgt eine kurze Beschreibung, wie man der App erlaubt, den Bluetooth-Adapter zu benutzen. Danach beginnt die eigentliche Programmierarbeit. Die Bluetooth spezifischen Elemente werden dabei immer wieder von Programmierungen begleitet, die der Gestaltung des Layouts dienen. In diesem Zusammenhand wird dabei die Bedeutung der xml-Dateien für das Programmieren einer App verdeutlicht. Ziel ist es, am Ende ein funktionsfähiges Programm zu haben, mit dem sich Textnachrichten zwischen zwei Handys austauschen lassen. Dies gilt gleichzeitig als Test, ob die App funktioniert. Wichtig ist dabei, die für Bluetooth wesentlichen Elemente hervorzuheben. Daher werden diese ausführlich beschrieben, während die Layouts und die Funktionen des Textaustauschs auf einem Minimum gehalten werden, da sie nur als Beispiel dienen.

3 Anlegen eines neuen Projekts

Zunächst wird unter File > New > Android Application Project ein neues Projekt angelegt:

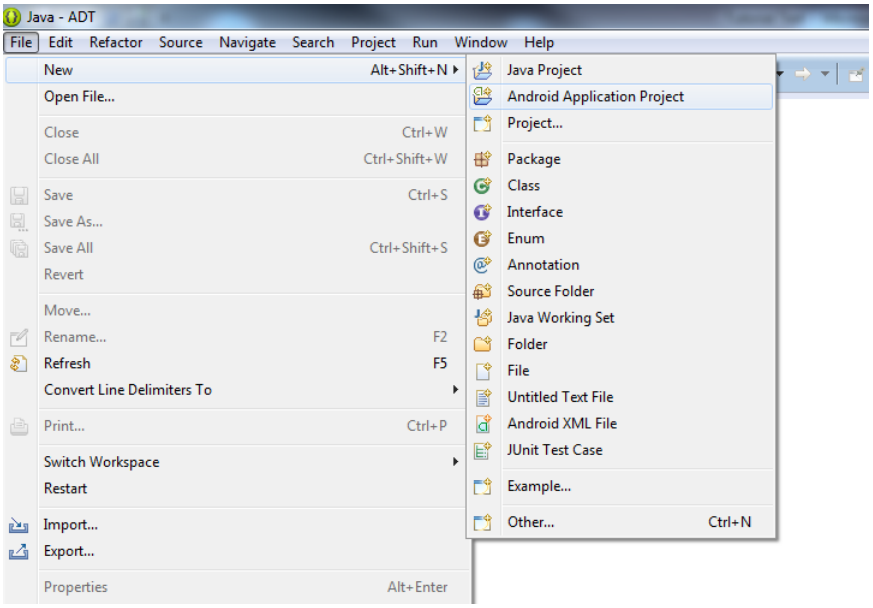

Abbildung 3.1: Anlegen eines neuen Projekts

Daraufhin öffnet sich ein Fenster, in dem der Name der App, des Projekts und des Pakets eingegeben werden können. Außerdem wird hier eingestellt, welche Android-Version ein Handy mindestens besitzen muss. Dies ist wichtig, da ältere Geräte nicht alle Befehle der neusten Android-Versionen kennen. Mit dieser Einstellung verhindert man, einen unbekannten Befehl zu verwenden, da Eclipse sonst darauf hinweist. Die Einstellung des Target SDK funktioniert ähnlich. Hier wird eingegeben, mit welcher Version die App auf jeden Fall noch laufen soll, daher kann hjer eine m öglichst hohe Version(hier API 17)verwendete werden . Da Android abwärts kompatibel ist, sollte es eigentlich auf jeder neueren Version laufen. Mit dieser Angabe wird nur noch einmal zusätzlich sichergestellt, dass kein Befehl bis zur verwendeten Version verändert wurde. Mit welcher Version kompiliert wird, ist ebenfalls nicht so wichtig.

Diese Anleitung ist für API 8 geschrieben, daher wird das Theme auf None gesetzt, da API 8 die anderen Themes nicht unterstützt. Der Name der App kann beliebig anderes gewählt werden. In diesem Fall muss jedoch darauf geachtet werden, dass entsprechende Stellen im Quellcode auch anders aussehen werden.

Für diese Anleitung wurden folgende Einstellungen verwendet:

Auf den folgenden Bildschirmen wird festgelegt, welche Einstellungen konfiguriert werden

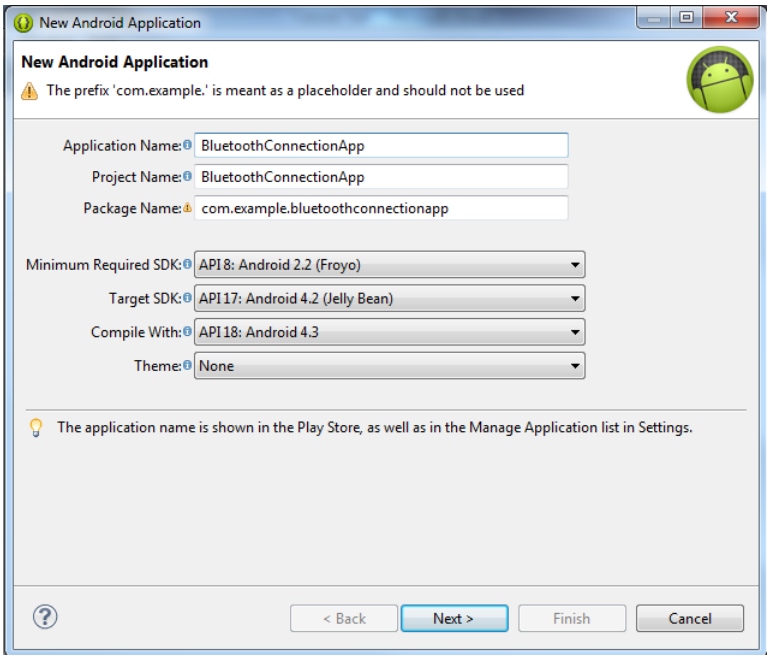

Abbildung 3.2: Name des Projekts festlegen und API-Versionen einstellen

sollen, welches Icon die App bekommen soll und wie das Grundlayout aussehen soll. Da dies alles für das Ziel eine Bluetooth-Verbindung herzustellen unwichtig ist, wurden hier jeweils die Standardeinstellungen übernommen:

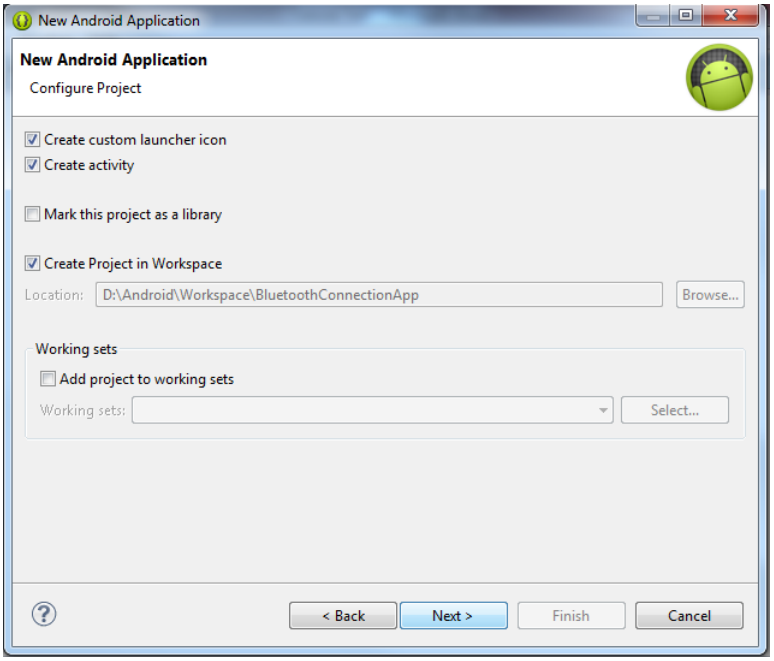

Abbildung 3.3: Projekt anlegen - Workspace festlegen

Auf dem nächsten Bildschirm werden dann der Name der Hauptaktivität und des Layouts festgelegt. Auch hier wurden für diese App die Standardeinstellungen gewählt. Diese Dateien sind sehr wichtig! In ihnen werden die meisten Veränderungen vorgenommen und

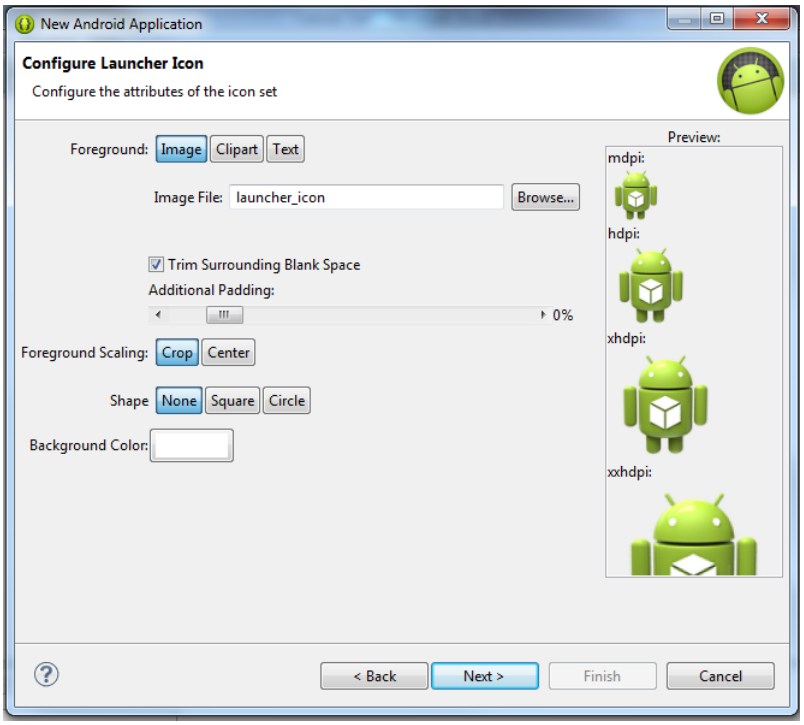

Abbildung 3.4: Projekt anlegen - Icon festlegen

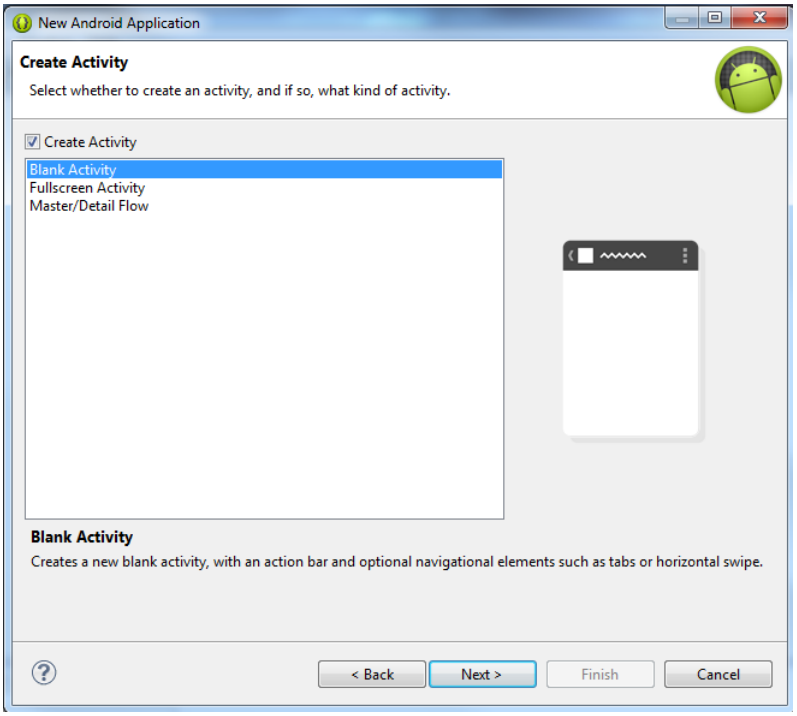

Abbildung 3.5: Projekt anlegen - Activity erzeugen

dabei immer wieder auf diese Dateien verwiesen. Es empfiehlt sich also, mit denselben Namen zu arbeiten, wenn ein lästiges Umdenken verhindert werden soll:

Nach Bestätigung des Finish-Buttons ist das Projekt angelegt.

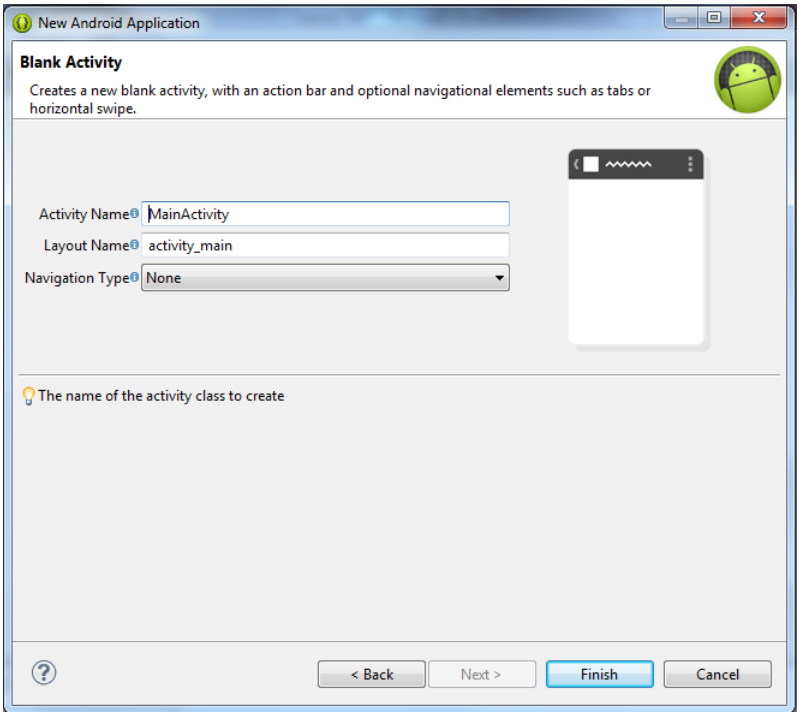

Abbildung 3.6: Projekt anlegen - Dateinamen festlegen

4 Erlaubnis einholen

Damit eine App Bluetooth benutzen kann, muss das Android-Betriebssystem zunächst eine Erlaubnis dafür geben. Diese muss nur einmal, bei der Installation, erfragt werden. Zur Anfrage der Erlaubnis (engl. permission), wird diese in die XML-Datei Android-Manifest.xml eingetragen, die sich direkt im Hauptordner des Projekts befindet:

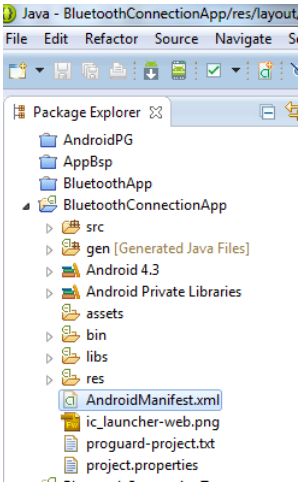

Abbildung 4.1: Die Datei AndroidManifest

Die Anfrage um Erlaubnis ist nach folgender Syntax aufgebaut:

```
1 <uses-permission android:name = "string" />
```

Listing 4.1: Syntax für die Erlaubnis

(mehr Infos: http://developer.android.com/guide/topics/manifest/uses-permission-element.html)

Für die Bluetooth-Kommunikation ist zunächst die Erlaubnis Bluetooth benutzen zu dürfen wichtig. Die Erlaubnis „BLUETOOTH“ erlaubt der App, Verbindungen bei anderen Geräten anzufragen oder zu bestätigen, so wie, bei bestehender Verbindung, Daten auszutauschen.

(mehr Infos: http://developer.android.com/reference/android/Manifest.permission.html#BLUETOOT

Mit dieser Erlaubnis kann prinzipiell schon eine Kommunikation stattfinden. Da jedoch häufig noch Änderungen bei den Einstellungen von Bluetooth vorgenommen werden sollen, wie das Einschalten von Bluetooth, das Sichtbarmachen des eigenen Geräts, die Suche nach anderen Geräten und die Kopplung mit anderen Geräten, während die App bereits läuft, ist es sinnvoll diese Einstellungen auch aus der App heraus steuern zu können. So muss die App nicht verlassen werden um diese Einstellungen vorzunehmen. Die App benötigt dafür die Erlaubnis „BLUETOOTH_ADMIN“.

(mehr Infos: http://developer.android.com/reference/android/Manifest.permission.html#BLUETOOTH

Diese beiden Erlaubnisse fügt man nach oben beschriebener Syntax in die „AndroidManifest.xml“ ein:

```
1 <manifest . . . >
2     . . .
3   <uses-permission android:name =
      "android.permission.BLUETOOTH" />
4   <uses-permission android:name =
      "android.permission.BLUETOOTH_ADMIN" />
5     . . .
6 </manifest>
```

Listing 4.2: AndroidManifest.xml

Damit sind alle Grundvoraussetzungen geschaffen. Die Punkte (. . .) sollen nur verdeutlichen, dass hier weiterer Code steht. Da dieser jedoch nicht verändert wird, wird darauf verzichtet ihn hier anzugeben. In den folgenden Abschnitten wird dies immer wieder so gehandhabt, ohne nochmal explizit darauf einzugehen. Teilweise werden die Punkte auch weggelassen, wenn bereits bekannte Klassen verändert werden und es zu keinen Verwechslungen kommen kann.

5 Die MainActivity

Im Ordner src/com.examples.bluetoothconnectionapp befindet sich die Datei MainActivity.java. Dies ist die Aktivität die beim Programmstart als erstes aufgerufen wird.

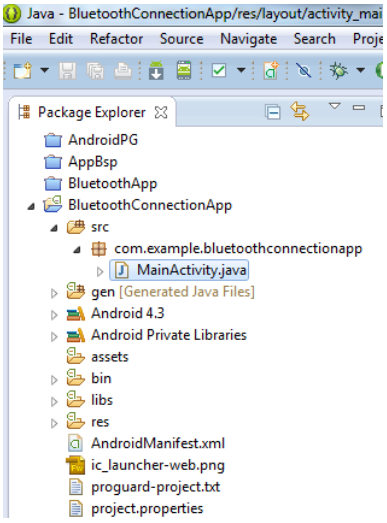

Abbildung 5.1: Die Datei MainActivity

Sie besitzt bei ihrer Erzeugung zwei Methoden, die Methode onCreate und die Methode onCreateOptionsMenu, die noch überschrieben werden müssen.

In der Methode onCreate wird eine Verbindung zu dem Layout, welches in die xml-Datei acitivity_main.xml ausgelagert wird, erstellt. Die Methode onCreateOptionsMenu, wird in dieser Anleitung nicht verwendet und kann gelöscht werden.

```
1 public class MainActivity extends Activity {
2
3         @Override
4         protected void onCreate(Bundle savedInstanceState) {
5                 super.onCreate(savedInstanceState);
6                 setContentView(R.layout.activity_main);
7         }
8
9 }
```

Listing 5.1: MainActivity.java

6 Die Layoutdateien

Die Datei activity_main.xml befindet sich im Ordner res/layout. Die Datei strings.xml im Ordner res/values.

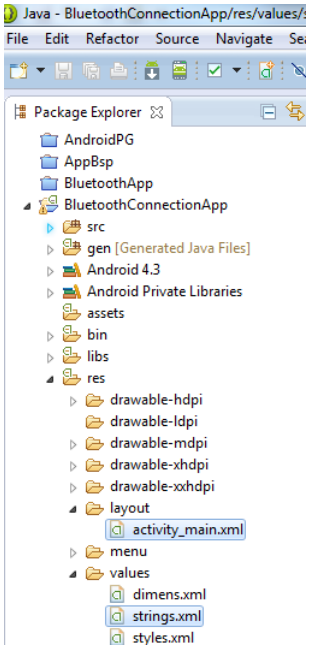

Abbildung 6.1: Die Dateien activitymain.xml und strings.xml

Die acitivity_main.xml beinhaltet das Layout der App. In diese werden später Die Buttons, Listenelemente, etc. eingetragen. Zurzeit befindet sich dort ein einziges Textfeld, das später nicht mehr benötigt wird und gelöscht werden kann. Man sieht an diesem Textfeld jedoch, dass auch dieses Textfeld weiter auf die Datei strings.xml verweist, mit dem Befehl @string/. Der dahinterstehenden Variable wird dort ein Wert zugeordnet.

```
1  <RelativeLayout
2      xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:paddingBottom="@dimen/activity_vertical_margin"
7      android:paddingLeft="@dimen/activity_horizontal_margin"
8      android:paddingRight="@dimen/activity_horizontal_margin"
9      android:paddingTop="@dimen/activity_vertical_margin"
10     tools:context=".MainActivity" >
11
12     <TextView
```

```
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="@string/hello_world" />
16
17 </RelativeLayout>
```

Listing 6.1: activity_main.xml

In der Datei strings.xml werden aber auch Variablen von anderen Dateien verwaltet, wie man sieht, wird hier beispielsweise schon der Name der App gespeichert, der dann zum Beispiel in der Titelleiste ausgegeben werden kann:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3
4     <string name="app_name">BluetoothConnectionApp</string>
5     <string name="action_settings">Settings</string>
6     <string name="hello_world">Hello world!</string>
7
8 </resources>
```

Listing 6.2: strings.xml

Das TextView und der hello_world-String können hier gelöscht werden, da sie für die App nicht benötigt werden.

Die Strings können etwas verschönert werden, in dem man Leerzeichen einfügt, oder ihre Namen ändert:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3
4     <string name="app_name">Bluetooth Connection App</string>
5     <string name="action_settings">Einstellungen</string>
6
7 </resources>
```

Listing 6.3: strings.xml

7 Besitzt das Gerät einen Bluetooth-Adapter?

Die erste Voraussetzung, um eine Bluetooth-Kommunikation aufbauen zu können, ist natürlich, dass das verwendete Gerät überhaupt Bluetooth unterstützt. Sollte die App jedoch auf einem Gerät ohne Bluetooth-Adapter installiert sein, soll ein Abstürzen der App verhindert werden und stattdessen die App ordnungsgemäß beendet werden. In anderen Apps, könnte es sinnvoll sein, dass vielleicht nur einige Funktionen der App abgeschaltet werden, wenn kein Bluetooth-Adapter vorhanden ist. Da diese Anleitung jedoch keine Benutzersoftware sondern lediglich ein Beispiel erstellt, wird auf eine gesonderte Behandlung verzichtet und sich damit begnügt die App ordnungsgemäß zu beenden.

Um zu prüfen, ob die App einen Bluetooth-Adapter besitzt, erzeugt man zunächst eine Repräsentation dieses Adapters in Form eines Objekt mBluetoothAdapter, der Klasse BluetoothAdapter. Dieses wird in der MainActivity deklariert und zunächst mit dem Wert null initialisiert.

(Infos zur Klasse Bluetooth Adapter: http://developer.android.com/reference/android/bluetooth/Blue

In der Methode onCreate wird mit der Methode getDefaultAdapter() der lokale Bluetooth Adapter ausgelesen und diesem Objekt zugeordnet. Ist kein Bluetooth Adapter vorhanden, gibt diese Methode null zurück. Daher wird im nächsten Schritt geprüft, ob mBluetoothAdaper=null ist. Ist dies der Fall, wird ein kurzes Popup geöffnet, was darauf hinweist, dass kein Adapter vorhanden ist und die App geschlossen.

(mehr Infos: http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getI

Beim erstellen, wird Eclipse zunäst noch Fehler anzeigen. Dies liegt daran, dass noch nicht alle Klassen importiert sind die verwendet werden. Dieses Problem öst man, indem man mit der Maus zu dem entsprechenden Fehler geht und per Mausklick die Importierfunktion auswählt. Diese Situation wird noch öfter vorkommen, jedoch nicht jedes Mal explizit erwähnt werden. Sollte also bei der Übernahme des weiteren Quelltextes ein Fehler auftauchen, so sollte jedes Mal geprüeft werden, ob dieser Fehler möglicherweise durch den Fehlenden import der Klasse zustande kommt.

```
1 public class MainActivity extends Activity {
2
3
4     //Hier werden die Repraesentanten der Bluetooth-Geraete
        deklariert:
5     // Lokaler Bluetooth Adapter
6     private BluetoothAdapter mBluetoothAdapter = null;
7
8
9         @Override
```

```
10          protected void onCreate(Bundle savedInstanceState) {
11                  super.onCreate(savedInstanceState);
12                  setContentView(R.layout.activity_main);
13
14
15                  // Auslesen des lokalen Bluetooth-Adapters
16          mBluetoothAdapter =
               BluetoothAdapter.getDefaultAdapter();
17
18          // Pruefen, ob ads Geraet Bluetooth besitzt
19          if (mBluetoothAdapter == null) {
20              Toast.makeText(this, "Bluetooth is not
                   available", Toast.LENGTH_LONG).show();
21              finish();
22              return;
23          }
24          }
25  }
```

Listing 7.1: MainActivity.java

8 Ist Bluetooth eingeschaltet?

Nachdem nun sichergestellt ist, dass ein Bluetooth-Adapter vorhanden ist, kann geprüft werden, ob dieser eingeschaltet ist. Dies geschieht in der Methode onStart(), die man sich automatisch zum Überschreiben generieren lassen kann (Rechte Maustaste > Source > Overwrite/Implement Methods...). Falls Eclipse an dieser Stelle fragt, an welcher Stelle die neue Methode eingefügt werden soll, so spielt es keine Rolle, welche Stelle ausgewählt wird.

Über die Methode isEnabled(), bekommt man einen Wahrheitswert zurück, der true liefert, falls Bluetooth eingeschaltet ist.

(mehr Infos: http://developer.android.com/reference/android/ bluetooth/BluetoothAdapter.html#isEnabled())

Ist der Adapter jedoch nicht eingeschaltet, besteht dank der Erlaubnis BLUETOOTH_ADMIN die Möglichkeit, den Adapter von dieser Stelle aus einschalten zu lassen. Dafür wird ein Intent erzeugt, eine Anfrage ans Betriebssystem, ob der Bluetooth-Adapter eingeschaltet werden soll. Über „BluetoothAdapter.ACTION_REQUEST_ENABLE“ wird die Systemabfrage, ob Bluetooth aktiviert werden soll erstellt und über die Methode „startActivityForResult()“ ins System abgesetzt. Die Methode „startActivityForResult()“ benötigt den Parameter „REQUEST_ENABLE_BT“. Dies ist eine ganzzahlige positive Zahl, durch die der „start-Methode“ eine Callback-Methode zugeordnet wird, die aktiviert wird, sobald das Betriebssystem die Entscheidung des Anwenders zurückliefert. Sie wird in der MainActivity als Konstante deklariert.

(mehr Infos: http://developer.android.com/reference/android/ bluetooth/BluetoothAdapter.html#ACTION_REQUEST_ENABLE

und: http://developer.android.com/reference/ android/app/Activity.html#startActivityForResult(android.content.Intent, int))

```
1 public class MainActivity extends Activity {
2
3 //Hier werden die Repraesentanten der Bluetooth-Geraete
     deklariert:
4         // Lokaler Bluetooth Adapter
5         private BluetoothAdapter mBluetoothAdapter = null;
6
7 // Codes fuer Intent Requests als Konstanten
8         private static final int REQUEST_ENABLE_BT = 1;
```

Listing 8.1: MainActivity.java

```
1 @Override
2 protected void onStart() {
3 // TODO Auto-generated method stub
```

```
4         super.onStart();
5
6         // Pruefen, ob der Adapter ausgeschaltet ist
7 // falls ja, geht es weiter in onActivityForResult
8         if (!mBluetoothAdapter.isEnabled()) {
9                 Intent enableIntent = new
                     Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
10                startActivityForResult(enableIntent,
                     REQUEST_ENABLE_BT);
11 // Wenn der Adapter eingeschaltet ist, wird die Methode
     Setup aufgerufen
12        }
13 // Wenn der Adapter eingeschaltet ist, wird die Methode
     Setup aufgerufen
14       else {
15        // Muss noch implementiert werden!
16            }
17 }
```

Listing 8.2: MainActivity.java

9 Verarbeiten der Antwort, ob Bluetooth eingeschaltet werden soll

Die Antwort des Betriebssystems wird zur obigen Anfrage, wird von einer Methode onActivityResult verarbeitet, die sich ebenfalls zum Überschreiben automatisch generieren lässt ((Rechte Maustaste > Source > Overwrite/Implement Methods...).

Diese Methode empfängt zwei Integer-Werte und ein Objekt vom Typ Intent. Da diese Methode Anfragen von verschiedenen Methoden verarbeiten kann, sind die Integer-Werte von großer Bedeutung. Über diese Werte, wird der Methode mittgeteilt, wer die Anfrage ins System gesetzt hat und wer die Antwort gegeben hat. Dadurch kann die Methode die Anfragen zuordnen und verarbeiten.

(mehr Infos: http://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, android.content.Intent)

Damit der Code leserlicher ist, werden die wenig aussagekräftigen Zahlen in besser verständliche Konstanten gespeichert, wie hier im Beispiel REQUEST_ENABLE_BT = 1.

Die Methode geht nun über switch-case-Struktur alle mögliche Request-Codes durch und prüft damit, welcher Code ausgeführt werden soll. Zunächst besteht hier nur die Möglichkeit für den Request-Code REQUEST_ENABLE_BT. Später werden jedoch weitere hinzukommen.

Kommt ein Ereignis mit dem Request-Code REQUEST_ENABLE_BT an, so wird der Result-Code ausgelesen und geprüft, ob die Anfrage bestätigt wurde, also ob Bluetooth eingeschaltet wurde.

Achtung: Im folgenden Quelltext befinden sich bereits Methodenaufrufe, von Methoden, die noch nicht implementiert wurden. Diese werden erst im Anschluss implementiert, so dass im Quellcode Fehler angezeigt werden, bis diese Implementierung nachgeholt wurde. Solange kann die App nich getestet werden.

```
1  @Override
2  protected void onActivityResult(int requestCode, int
       resultCode, Intent data) {
3          // TODO Auto-generated method stub
4          super.onActivityResult(requestCode, resultCode,
             data);
5
6          switch (requestCode) {
7          case REQUEST_ENABLE_BT:
8          // Pruefen ob Bluetooth eingeschaltet wurde
9                  if (resultCode == Activity.RESULT_OK) {
10                 // Bluetooth wurde eingeschaltet
```

```
11                  setup();
12          }
13 else {
14                  // Bluetooth wurde nicht eingeschaltet
15 Toast.makeText(this, R.string.bt_not_enabled_leaving,
      Toast.LENGTH_SHORT).show();
16                  finish();
17                  }
18          }
19 }
```

Listing 9.1: MainActivity.java

Falls Bluetooth aktiviert wurde, wird die Methode setup() aufgerufen, die noch implementiert werden muss. Falls die Anfrage vom Benutzer abgelehnt wurde, wird eine Textnachricht ausgegeben und die Anwendung beendet, da ohne Bluetooth nicht kommuniziert werden kann. Der Texte wird hier nicht direkt in den Toast geschrieben, sondern wie bereits oben erwähnt werden alles Texte in einer Datei strings.xml im Ordner res/layout gespeichert. Über R wird auf diesen String verwiesen. Es muss also der entsprechende String in die string.xml eingefügt werden. Da noch viele Strings eingefügt werden sollen, wird über den Kommentar <– Bluetooth Connection–> ein wenig Ordnung geschaffen, da hier hin nun alle Strings geschrieben werden sollen, die zur Bluetooth Connection gehören.

```
1 <resources>
2
3     <string name="app_name">BluetoothConnectionApp</string>
4     <string name="action_settings">Einstellungen</string>
5
6         <!--  Bluetooth Connection -->
7     <string name="bt_not_enabled_leaving">Bluetooth ist
         deaktiviert. Beende  App.</string>
```

Listing 9.2: strings.xml

10 Das Layout

Interessanter ist natürlich der Fall, in dem die App nicht beendet wird, sondern weiter eine Verbindung aufgebaut werden soll. Es muss also die Methode setup() implementiert werden. Diese soll zunächst einmal eine Bedieneroberfläche für die App bereitstellen. Dazu werden verschiedene Elemente erzeugt und das dazugehörige Layout in der acitivity_main.xml hinterlegt.

Zur Gestaltung des Layouts muss überlegt werden, was die App hinterher können soll. Zum einen soll sie Geräte suchen können. Diese Suche soll über einen Button angeregt werden. Nun könnte man weitere Elemente einfügen, die die Suche verarbeiten. Für eine bessere Übersicht, wird die Suche jedoch nach Betätigung des Suchen-Buttons in einer neuen Activity verarbeitet.

Damit andere Geräte eine Verbindung erfragen können, muss das eigene Gerät sichtbar sein. Um das Gerät sichtbar zu machen, wird ebenfalls ein Button eingefügt.

Da später Texte gelesen und empfangen werden können, wird eine Textausgabe, die hier in Form einer Liste verwirklicht wird, benötigt. Zum Senden muss es ein Textfeld geben, in dem man Texte eingeben kann. Durch einen Sende-Button werden diese dann versandt. Wie dies im Einzelnen funktioniert, wird später gezeigt. An dieser Stelle sollen aber alle Elemente bereits ohne Inhalt bereits eingerichtet werden.

An dieser Stelle wird es etwas unübersichtlich, da viele Elemente gleichzeitig eingefügt werden, jedoch passiert nichts Kompliziertes. Zunächst einmal, werden alle Elemente in der MainActivity deklariert.

```
1  public class MainActivity extends Activity {
2          . . .
3          //Hier werden die Repraesentanten der
               Bluetooth-Geraete deklariert:
4          // Lokaler Bluetooth Adapter
5          private BluetoothAdapter mBluetoothAdapter = null;
6          // Adapter fuer die Nachrichten
7          private ArrayAdapter<String> mMessagesArrayAdapter;
8
9
10         // Layout-Elemente
11         private Button buttonScanDevices;
12         private Button buttonEnsureDiscoverable;
13         private ListView listViewMessages;
14         private EditText editTextMessage;
15         private Button buttonSendMessage;
```

Listing 10.1: MainActivity.java

Anschließend werden sie in der Methode setup , zunächst noch leer erzeugt.

```
1  private void setup() {
2          // TODO Auto-generated method stub
3
4          // Scan-Button mit Listener
5          buttonScanDevices = (Button)
              findViewById(R.id.buttonScanDevices);
6          buttonScanDevices.setOnClickListener(new
              OnClickListener() {
7            public void onClick(View v) {
8
9            }
10         });
11
12         // Sichtbar-Button mit Listener
13                 buttonEnsureDiscoverable = (Button)
                      findViewById(R.id.buttonEnsureDiscoverable);
14         buttonEnsureDiscoverable.setOnClickListener(new
              OnClickListener() {
15           public void onClick(View v) {
16
17           }
18         });
19
20         // Array-Adapter fuer die Nachrichten
21                 mMessagesArrayAdapter = new
                      ArrayAdapter<String>(this,R.layout.message);
22         listViewMessages = (ListView)
              findViewById(R.id.listViewMessages);
23         listViewMessages.setAdapter(mMessagesArrayAdapter);
24
25         // Textfeld zum Schreiben
26         editTextMessage = (EditText)
              findViewById(R.id.editTextMessage);
27
28         //Sende-Button mit Listener
29         buttonSendMessage = (Button)
              findViewById(R.id.buttonSendMessage);
30         buttonSendMessage.setOnClickListener(new
              OnClickListener() {
31           public void onClick(View v) {
32
33           }
34         });
```

Listing 10.2: MainActivity.java

Über R wird auf das Layout in der activity_main.xml und der strings.xml verwiesen. Diese müssen also auch einen Inhalt für entsprechende Verweise bekommen. Da diese Anleitung

sich mit Bluetooth beschäftigt, wird nicht näher darauf eingegangen, wie die xml-Elemente gestaltet sind. Sie werden hier lediglich angegeben, da sie notwendig sind, um weiter arbeiten zu können.

```
1  <RelativeLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      tools:context=".MainActivity" >
10
11     <EditText
12         android:id="@+id/editTextMessage"
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:layout_alignParentBottom="true"
16         android:layout_alignParentLeft="true"
17         android:ems="10" >
18
19         <requestFocus />
20     </EditText>
21
22     <Button
23         android:id="@+id/buttonSendMessage"
24         style="?android:attr/buttonStyleSmall"
25         android:layout_width="wrap_content"
26         android:layout_height="wrap_content"
27         android:layout_alignBottom="@+id/editTextMessage"
28         android:layout_toRightOf="@+id/editTextMessage"
29         android:text="@string/buttonSendMessage" />
30
31     <ListView
32         android:id="@+id/listViewMessages"
33         android:layout_width="match_parent"
34         android:layout_height="wrap_content"
35         android:layout_above="@+id/buttonSendMessage"
36         android:layout_alignLeft="@+id/editTextMessage"
37         android:layout_alignRight="@+id/buttonSendMessage"
38         android:layout_below="@+id/buttonScanDevices" >
39
40     </ListView>
41
42     <Button
43         android:id="@+id/buttonScanDevices"
44         style="?android:attr/buttonStyleSmall"
```

```
45          android:layout_width="wrap_content"
46          android:layout_height="wrap_content"
47          android:layout_alignLeft="@+id/listViewMessages"
48          android:layout_alignParentTop="true"
49          android:text="@string/buttonScanDevices" />
50
51      <Button
52          android:id="@+id/buttonEnsureDiscoverable"
53          style="?android:attr/buttonStyleSmall"
54          android:layout_width="wrap_content"
55          android:layout_height="wrap_content"
56          android:layout_above="@+id/listViewMessages"
57          android:layout_alignRight="@+id/listViewMessages"
58          android:text="@string/buttonEnsureDiscoverable" />
59
60
61
62  </RelativeLayout>
```

Listing 10.3: activity_main.xml

```
1                   . . .
2          <!-- MainActivity -->
3      <string name="buttonScanDevices">Geraetsuche</string>
4      <string name="buttonEnsureDiscoverable">Sichtbar
          werden</string>
5      <string name="buttonSendMessage">Senden</string>
```

Listing 10.4: strings.xml

Für die Messages muss eine ganz neue xml-Datei angelegt werden: res/layout/message.xml. Diese sieht zwar anfangs sehr leer aus, aber sie beinhaltet später die Nachrichten.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TextView
      xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="wrap_content"
5      android:textSize="18sp"
6      android:padding="5dp"
7  />
```

Listing 10.5: message.xml

Die App sollte beim Starten danach etwa so aussehen:

Abbildung 10.1: So sieht die App aus

11 Der Verbindungsaufbau

Die Setup-Methode wird nicht nur genutzt, um das Layout zu gestalten, sondern auch, auch um die Verbindungsherstellung vorzubereiten. Die Verbindungsherstellung wird in eine eigene Service-Klasse ausgelagert, die Klasse BluetoothConnectionService. Die setup-Methode erzeugt lediglich ein Objekt dieser Klasse. Dieses Objekt muss in der MainActivity also zunächst wieder deklariert werden. Ebenso initialisiert die setup-Methode den einen Buffer, um Strings rauszuschicken, der ebenfalls deklariert werden muss.

```
1  public class MainActivity extends Activity {
2
3          . . .
4          //Hier werden die Repraesentanten der
               Bluetooth-Geraete deklariert:
5          // Lokaler Bluetooth Adapter
6          private BluetoothAdapter mBluetoothAdapter = null;
7          // Adapter fuer die Nachrichten
8          private ArrayAdapter<String> mMessagesArrayAdapter;
9          // Der Connection Service
10         private BluetoothConnectionService
               mBluetoothConnectionService = null;
11         // String buffer fuer ausgehende Nachrichten
12         private StringBuffer mOutStringBuffer;
```

Listing 11.1: MainActivity.java

Damit die Service-Klasse Informationen an die MainAcitivity zurückgeben kann, wird ihr ein Handler übergeben, der ebenfalls noch implementiert werden muss. Die Setup-Klasse wird also wie folgt vervollständigt:

```
1  private void setup() {
2                  . . .
3  // Objekt der Service-Klasse initialisieren
4  mBluetoothConnectionService = new
       BluetoothConnectionService(this, mHandler);
5
6  // Buffer fuer ausgehende Nachrichten leer initialisieren
7  mOutStringBuffer = new StringBuffer("");
8  }
```

Listing 11.2: MainActivity.java

Damit ist die setup-Methode vollständig und der Rest wird von der Service-Klasse und dem Handler übernommen. Die setup-Methode muss immer aufgerufen werden, wenn ein Bluetooth-Adapter eingeschaltet ist, um die Verbindung vorzubereiten. Bislang wird sie

aber nur aufgerufen, wenn Bluetooth von der App selber eingeschaltet wurde. Damit die setup-Methode auch aufgerufen wird, wenn Bluetooth bereits zu Beginn der Anwendung aktiv ist, muss sie also auch in der onStart-Methode aufgerufen, bei Erfüllung dieser Bedingung aufgerufen werden. Daher wird die else-Bedingung der onStart-Methode mit dem Aufruf gefüllt. Um zu verhindern, dass bereits eine Verbindung der App besteht, geschieht dies jedoch nur, nachdem geprüft wurde, dass dies nicht der Fall ist.

```
1  @Override
2  protected void onStart() {
3                   . . .
4          if (
5          }
6  // Wenn der Adapter eingeschaltet ist, wird die Methode
      Setup aufgerufen
7          else {
8          if (mBluetoothConnectionService == null) {
9                   setup();
10             }
11         }
12 }
```

Listing 11.3: MainActivity.java

Der Handler wird zunächst erzeugt, jedoch erst später implementiert, wenn klar ist, welche Nachtrichten er verarbeiten soll.

```
1  public class MainActivity extends Activity {
2
3          . . .
4  // Handler zur Verarbeitung der Informationen von
      BluetoothConnectionService
5          private final Handler mHandler = new Handler() {
6
7          };
```

Listing 11.4: MainActivity.java

12 Die Klasse BluetoothConnectionService

Wie beschrieben, wird der Verbindungsaufbau in einer Service-Klasse ausgelagert. Dazu muss zunächst eine neue Klasse erzeugt werden. Dies geschieht über File > New > Class. Als Name wird der verwendete Name BluetoothConnectionService eingegeben. Alle anderen Einstellungen werden übernommen.

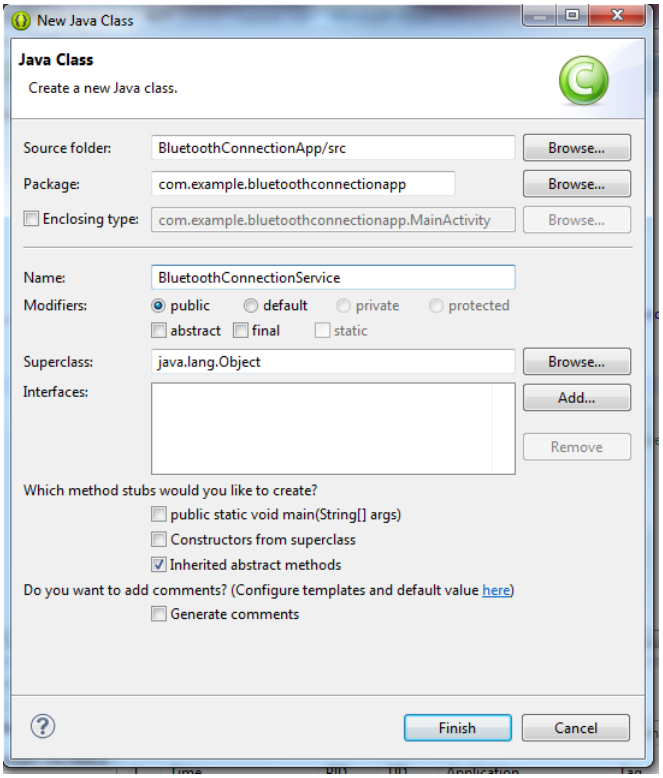

Abbildung 12.1: Erstellen einer neuen Klasse

Diese Klasse benötigt zunächst einen Konstruktor, der die Verbindung vorbereitet. Diesem Konstruktor wird auch der Handler übergeben. Bevor der Konstruktor aufgerufen wird, werden einige Variablen und Konstanten definiert.

Damit diese Klasse auch den eigenen Bluetooth-Adapter kennt, wird in ihr ebenfalls ein Objekt der Klasse BluetoothAdapter erzeugt und über getDefaultAdapter festgelegt. Der Handler wird deklariert und bei der Übergabe eingelesen. Hinzu kommt eine weitere Variable, über die festgehalten wird, was die Service-Methode gerade macht. Dieser Status wird als Zahl codiert, ähnlich wie bei der onActivityResult-Methode. Diese Zahlen werden wieder als aussagekräftige Konstanten gespeichert, um eine bessere Lesbarkeit zu erreichen. Zu Beginn wird noch nichts gemacht, aber die Zustände, dass gelauscht werden soll,

ob ein anderes Gerät eine Verbindung aufbauen will, dass selbst eine Verbindung initiiert werden soll und dass eine Verbindung existiert, sind denkbar.

```
1  public class BluetoothConnectionService {
2
3          private final BluetoothAdapter mAdapter;
4          private final Handler mHandler;
5          private int mState;
6
7          // Die Konstanten geben an, was der Service gerade
              macht.
8          public static final int STATE_NONE = 0;       // es
              wird nichts gemacht
9          eingehenden Verbindung
10         public static final int STATE_CONNECTING = 2; //
              Inititieren einer ausgehenden Verbindung
11         public static final int STATE_CONNECTED = 3;  //
              Verbunden
12
13
14         // Vorbereiten einer Bluetooth-Verbindung
15         public BluetoothConnectionService(Context context,
              Handler handler) {
16
17                 mAdapter =
                      BluetoothAdapter.getDefaultAdapter();
18         mState = STATE_NONE;
19         mHandler = handler;
20     }
21
22 }
```

Listing 12.1: BluetoothConnectionService.java

Diese Zustände können zur Kommunikation mit der MainActivity benutzt werden.

Dazu implementiert man zum einen eine Methode einen neuen Status setzt. Dazu wird zunächst der Status an diese Methode übergeben und in mState gespeichert. Anschließend wird der Status über die Methode obtainMessage dem Handler übergeben.

(mehr Infos: http://developer.android.com/reference/android/os/Handler.html#obtainMessage(int, int, int))

Die Methode wird als synchronized deklariert. Dies macht sie threadsicher. Durch dieses Schlüsselwort wird gewährleistet, dass die Methode komplett ausgeführt werden kann, ehe eine andere Methode Zugriff auf die Daten bekommt, so dass es zu keinen Konflikten zwischen konkurrierenden Methoden kommt.

```
1  public class BluetoothConnectionService {
2
3
4      // Neuen Status dem Handler mitteilen
```

```
5     private synchronized void setState(int state) {
6         mState = state;
7
8         // Der Status wird dem Handler uebergeben
9         mHandler.obtainMessage(MainActivity.MESSAGE_STATE_CHANGE,
              state, -1).sendToTarget();
10    }
```

Listing 12.2: BluetoothConnectionService.java

MESSAGE_STATE_CHANGE ist eine Konstante in der MainActivity, die noch deklariert werden muss. Sobald der Handler diese sieht, soll er reagieren. Über den Parameter state, wird dem Handler dann eine zusätzliche Information mitgegeben, welcher Status gesetzt ist. Folgende Zustände sind dafür angedacht: Lesen, Schreiben, der Geräte-Name und ein Zustand Toast, die der Handler verarbeiten können soll. Diese werden also wieder als Konstanten codiert. Sie müssen in der MainActivity deklariert werden.

```
1 public class MainActivity extends Activity {
2     // Nachrichtentypen
3     public static final int MESSAGE_STATE_CHANGE = 1;
4     public static final int MESSAGE_READ = 2;
5     public static final int MESSAGE_WRITE = 3;
6     public static final int MESSAGE_DEVICE_NAME = 4;
7     public static final int MESSAGE_TOAST = 5;
```

Listing 12.3: MainActivity.java

Im bereits ohne Inhalt implementierten Handler muss nun also eine Methode implementiert werden, die diese Nachrichten sortiert. Dafür gibt es die Methode handleMessage, die noch überschrieben werden muss. Eine erste Abfrage prüft, ob der Status verändert wurde. Ist dies der Fall, wird der Status des BluetoothConnectionService abgerufen und der Zustand im Titel ausgegeben.

```
1 private final Handler mHandler = new Handler() {
2
3   @Override
4   public void handleMessage(Message msg) {
5     // TODO Auto-generated method stub
6     super.handleMessage(msg);
7     switch (msg.what) {
8         case MESSAGE_STATE_CHANGE:
9                 switch (msg.arg1) {
10              case BluetoothConnectionService.STATE_CONNECTED:
11                setStatus(getString(R.string.title_connected_to,
                      mConnectedDeviceName));
12                mMessagesArrayAdapter.clear();
13                break;
14              case BluetoothConnectionService.STATE_CONNECTING:
15                setStatus(R.string.title_connecting);
16                break;
17              case BluetoothConnectionService.STATE_LISTEN:
18              case BluetoothConnectionService.STATE_NONE:
```

```
19                setStatus(R.string.title_not_connected);
20                break;
21            }
22        break;
23        }
24    }
```

Listing 12.4: MainActivity.java

Die anderen Fälle werden später implementiert. Die Ausgabetexte, müssen wieder in die string.xml geschrieben werden.

```
1         <!--  Bluetooth Connection -->
2     <string name="bt_not_enabled_leaving">Bluetooth ist
          deaktiviert. Beende App.</string>
3     <string name="title_connecting">verbinde...</string>
4     <string name="title_connected_to">verbunden</string>
5     <string name="title_not_connected">nicht
          verbunden</string>
```

Listing 12.5: string.xml

Die Methode setStatus muss ebenfalls noch implementiert werden. Über die kann später ausgegeben werden, in welchem Zustand sich die App gerade befindet.

mConnectedDeviceName wird in der ActivityMain deklariert und zunächst null gesetzt. Hierin wird später das ausgewählte Gerät gespeichert, dass die BluetoothServiceConnection übergibt.

```
1  public class MainActivity extends Activity {
2          . . .
3
4  // Name des ausgewaelten Geraetes
5  private String mConnectedDeviceName = null;
6  . . .
7  private void setStatus(String string) {
8                          // TODO Auto-generated method stub
9
10                 }
11
12         private void setStatus(int titleConnecting) {
13                         // TODO Auto-generated method stub
14
15                 }
```

Listing 12.6: MainActivity.java

13 Geräte suchen

Nachdem der Handler und die Service-Klasse nun eingerichtet sind, kann damit begonnen werden, die Buttons zu implementieren. Zunächst wird der Button zur Suche von Geräten implementiert.

```
1  private void setup() {
2
3          // Scan-Button mit Listener
4          buttonScanDevices = (Button)
              findViewById(R.id.buttonScanDevices);
5          buttonScanDevices.setOnClickListener(new
              OnClickListener() {
6              public void onClick(View v) {
7  Intent serverIntent = new Intent(MainActivity.this,
      DeviceListActivity.class);
8  startActivityForResult(serverIntent,
      REQUEST_CONNECT_DEVICE_SECURE);
9
10             }
11         });
```

Listing 13.1: MainActivity.java

Er soll eine zusätzliche Activity öffnen, die sich darum kümmert, die verfügbaren Geräte aufzulisten. Sie wird in einer separaten Klasse implementiert. Diese heißt DeviceListActivity und wird wieder über File > New > Class erzeugt. Sie erbt von der Klasse Acitivity.

Die Konstante REQUEST_CONNECTED_DEVICE_SECURE muss dafür noch deklariert werden.

```
1  // Codes fuer Intent Requests als Konstanten
2          private static final int REQUEST_ENABLE_BT = 1;
3          private static final int
              REQUEST_CONNECT_DEVICE_SECURE = 1;
```

Listing 13.2: MainActivity.java

In ihrer onCreate Methode soll wieder ein Layout gestaltet werden. Es soll in etwa so aussehen:

In einer Liste werden die gekoppelten Geräte angezeigt. Über den Suche-nach-Geräten-Button, kann man weitere unbekannte Geräte suchen.

Die Layoutdaten werden wieder in eine device_list.xml Datei ausgelagert und die Strings in der string.xml gespeichert.

Dies ergibt folgenden Quellcode:

Abbildung 13.1: Layout der Suche

Für die DeviceListActivity.java:

```
1  public class DeviceListActivity extends Activity {
2
3
4
5          @Override
6      protected void onCreate(Bundle savedInstanceState) {
7          super.onCreate(savedInstanceState);
8
9          // Window
10         requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
11         setContentView(R.layout.device_list);
12
13         // Bei Abbruch
14         setResult(Activity.RESULT_CANCELED);
15
16         // Button der die Suche nach Geraeten startet
17         Button scanButton = (Button)
              findViewById(R.id.button_scan);
18         scanButton.setOnClickListener(new OnClickListener() {
19             public void onClick(View v) {
20                 //Geraete suchen noch implementieren
21
22                 v.setVisibility(View.GONE);
```

```
            }
        });
        }
}
```

Listing 13.3: DeviceListActivity.java

Für die device_list.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >


    <TextView android:id="@+id/title_paired_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/title_paired_devices"
        android:visibility="gone"
        android:background="#666"
        android:textColor="#fff"
        android:paddingLeft="5dp"
    />
    <ListView android:id="@+id/paired_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:stackFromBottom="true"
        android:layout_weight="1"
    />
    <TextView android:id="@+id/title_new_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/title_other_devices"
        android:visibility="gone"
        android:background="#666"
        android:textColor="#fff"
        android:paddingLeft="5dp"
    />
    <ListView android:id="@+id/new_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:stackFromBottom="true"
        android:layout_weight="2"
    />
    <Button android:id="@+id/button_scan"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
41          android:text="@string/button_scan"
42      />
43
44  </LinearLayout>
```

Listing 13.4: device_list.xml

In der Klasse strings.xml müssen folgende Einträge hinzugefügt werden:

```
1           <!--  DeviceListActivity -->
2       <string name="scanning">Suche nach Geraeten...</string>
3       <string name="select_device">Waehle ein Geraet zum
           Verbinden aus</string>
4       <string name="none_paired">Keine Geraet ist
           gekoppelt</string>
5       <string name="none_found">Kein Geraet gefunden</string>
6       <string name="title_paired_devices">Gekoppelte
           Geraete</string>
7       <string name="title_other_devices">Weitere verfuegbare
           Geraete</string>
8       <string name="button_scan">Suche nach Geraeten</string>
```

Listing 13.5: strings.xml

Damit in der Liste die Geräte angezeigt werden, müssen dafür Array-Adapter implementiert werden und an die ListViews übergeben werden. Es soll eine Liste für die gekoppelten Geräte erstellt werden, die direkt gefüllt wird und eine für die gesuchten Geräte, die vor Betätigung des Buttons leer bleibt.

```
1   public class DeviceListActivity extends Activity {
2
3     // Deklarieren der Adapter
4     private BluetoothAdapter mBtAdapter;
5     private ArrayAdapter<String> mPairedDevicesArrayAdapter;
6     private ArrayAdapter<String> mNewDevicesArrayAdapter;
7     @Override
8     protected void onCreate(Bundle savedInstanceState) {
9           . . .
10      // Array-Adapter fuer gekoppelte Geraete
11      mPairedDevicesArrayAdapter = new
           ArrayAdapter<String>(this, R.layout.device_name);
12      // Array-Adapter fuer neu gefundene Geraete
13      mNewDevicesArrayAdapter = new ArrayAdapter<String>(this,
           R.layout.device_name);
14
15      // ListView fuer gekoppelte Geraete
16      ListView pairedListView = (ListView)
           findViewById(R.id.paired_devices);
17      pairedListView.setAdapter(mPairedDevicesArrayAdapter);
18
19
20      // ListView fuer neu gefundene Geraete
```

```
21      ListView newDevicesListView = (ListView)
           findViewById(R.id.new_devices);
22      newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
23    }
24  }
```

Listing 13.6: DeviceListActivity.java

Im Ordner res/layout muss wieder eine xml-Datei device_name.xml erstellt werden, die das Layout des Gerätenamens bestimmt.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TextView
      xmlns:android="http://schemas.android.com/apk/res/android"
3       android:layout_width="match_parent"
4       android:layout_height="wrap_content"
5       android:textSize="18sp"
6       android:padding="5dp"
7  />
```

Listing 13.7: device_name.java

14 Gekoppelte Geräte

Die gekoppelten Geräte sollen sofort in das ListView eingetragen werden. Dazu werden die gekoppelten Geräte über die Methode getBondedDevices ausgelesen und in ein Set gespeichert. Dieses Set wird dann durchlaufen und für jedes gekoppelte Gerät wird der Name und die Adresse als String in das Adapter-Array für die gekoppelten Geräte gespeichert. Sollte kein Gerät gekoppelte sei, wird ein String mit dem Hinweis, dass kein Gerät vorhanden ist, in das Array gespeichert.

```
1  public class DeviceListActivity extends Activity {
2
3          . . .
4    @Override
5    protected void onCreate(Bundle savedInstanceState) {
6
7        . . .
8
9    // Liste der gekoppleten Geraete
10   Set<BluetoothDevice> pairedDevices =
       mBtAdapter.getBondedDevices();
11
12   // Wenn gekoppelte Gerate vorhanden, in Adapter-Array
       speichern
13   if (pairedDevices.size() > 0) {
14     findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE)
15     for (BluetoothDevice device : pairedDevices) {
16       mPairedDevicesArrayAdapter.add(device.getName() + "\n"
           + device.getAddress());
17     }
18   }
19   else {
20     String noDevices =
         getResources().getText(R.string.none_paired).toString();
21     mPairedDevicesArrayAdapter.add(noDevices);
22   }
23       . . .
```

Listing 14.1: DeviceListActivity.java

15 Neue Geräte suchen

Die Suche nach neuen Geräten ist etwas komplizierter. Sie wird per Button ausgelöst. Dazu wird eine Methode doDiscovery implementiert, die vom Suchen-Button aufgerufen wird:

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3            . . .
4
5    // Button der die Suche nach Geraeten startet
6    Button scanButton = (Button)
       findViewById(R.id.button_scan);
7    scanButton.setOnClickListener(new OnClickListener() {
8      public void onClick(View v) {
9          //Geraete suchen noch implementieren
10                 doDiscovery();
11         v.setVisibility(View.GONE);
12     }
13   });
14 }
```

Listing 15.1: DeviceListActivity.java

Die Methode doDiscovery prüft zunächst, ob der Bluetooth-Adapter bereits eine Suche ausführt und bricht diese in diesem Fall ab, damit sich die beiden Suchen nicht stören. Danach wird eine neue Suche aufgerufen.

```
1  private void doDiscovery() {
2
3          // Stoppe eine laufende Suche, falls vorhanden
4          if (mBtAdapter.isDiscovering()) {
5                  mBtAdapter.cancelDiscovery();
6          }
7
8          // Starte suche
9          mBtAdapter.startDiscovery();
10 }
```

Listing 15.2: DeviceListActivity.java

Die Methode startet eine Suche über das Betriebssystem. Um die Rückmeldungen verarbeiten zu können, müssen Broadcast-Receiver registriert werden, die auf die Rückmeldungen des Betriebssystems warten und diese verarbeiten.

Ein Receiver lauscht, ob ein Gerät gefunden wurde:

```
protected void onCreate(Bundle savedInstanceState) {
        . . .
        // Registriere BroadcastReceiver Geraet gefunden
    IntentFilter filter = new
       IntentFilter(BluetoothDevice.ACTION_FOUND);
    this.registerReceiver(mReceiver, filter);
```

Listing 15.3: DeviceListActivity.java

Einer ob die Suche beendet wurde:

```
// Registriere BroadcastReceiver Suche beendet
filter = new
   IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
this.registerReceiver(mReceiver, filter);
```

Listing 15.4: DeviceListActivity.java

Der BroadcastReceiver implementiert die Methode onReceive, die die Rückmeldung des Betriebssystems verarbeitet. Sobald ein Gerät gefunden wird, wird über die Methode getParcelableExtra der Name und die MAC-Adresse ausgelesen. Daraufhin wird geprüft, ob das Gerät ein bereits gekoppeltes Gerät ist. Falls nicht, werden Name und MAC-Adresse als String in den Array-Adapter gespeichert.

```
// Broadcast Receiver
private final BroadcastReceiver mReceiver = new
   BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();

    // bei Rueckmeldung
    if (BluetoothDevice.ACTION_FOUND.equals(action)) {
      // BluetoothDevice auslesen
      BluetoothDevice device =
         intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
      // Pruefen, dass es nicht bereits in der Liste der
         gekoppelten Geraete ist
      if (device.getBondState() !=
         BluetoothDevice.BOND_BONDED) {
        mNewDevicesArrayAdapter.add(device.getName() + "\n"
          + device.getAddress());
      }
    }
  }
};
```

Listing 15.5: DeviceListActivity.java

16 Gerät für Verbindung auswählen

Damit sind beide Listen gefüllt, einmal die der gekoppelten und einmal die der neu gefundenen Geräte. Um diese benutzen zu können, muss jedoch eins ausgewählt werden. Dies geschieht über einen OnItemClickListener. Sobald auf ein Item in der Liste geklickt wird, stoppt das eigene Gerät die Suche, da sie nicht länger nötig ist. Anschließend wird der String des ausgewählten Elements ausgelesen. Von diesem wird ein Substring gebildet, der aus den letzten 17 Zeichen besteht, weil sich darin die Mac-Adress befindet. Diese wir in einem String gespeichert. Diese wird an eine Variable EXTRA_DEVICE_ADRESS übergeben, die noch deklariert werden muss.

Anschließend wird das Ergebnis auf RESULT_OK gesetzt und die Activity für die Gerätesuche beendet.

```
1 public class DeviceListActivity extends Activity {
2         // Fuer Rueckgabe
3     public static String EXTRA_DEVICE_ADDRESS =
         "device_address";
```

Listing 16.1: DeviceListActivity.java

```
1  // Listener fuer die Geraeteauswahl
2  private OnItemClickListener mDeviceClickListener = new
      OnItemClickListener() {
3    public void onItemClick(AdapterView<?> av, View v, int
       arg2, long arg3) {
4     // Suche stoppen
5     mBtAdapter.cancelDiscovery();
6
7     // MAC-Adresse auslesen
8     String info = ((TextView) v).getText().toString();
9     String address = info.substring(info.length() - 17);
10
11    // Eregbnis Intent mit der MAC-Adresse
12    Intent intent = new Intent();
13    intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
14
15    // Beenden
16    setResult(Activity.RESULT_OK, intent);
17    finish();
18   }
19 };
```

Listing 16.2: DeviceListActivity.java

Damit ist die Gerätesuche beendet. Es fehlt nur noch eine Methode, die regelt, was passiert, wenn die Activity geschlossen wird. Dafür wird die Methode onDestroy implementiert. Diese kümmert sich darum, dass die Suche nach Geräten eingestellt wird, falls dies noch nicht geschehen ist und die Broadcast-Receiver abgemeldet werden.

```
1  @Override
2      protected void onDestroy() {
3          super.onDestroy();
4
5          // Suche beenden
6          if (mBtAdapter != null) {
7              mBtAdapter.cancelDiscovery();
8          }
9
10         // Broadcast Reciever abmelden
11         this.unregisterReceiver(mReceiver);
12     }
```

Listing 16.3: DeviceListActivity.java

17 Sichtbar machen

Wenn man mit dem Gerät nicht selber die Suche starten will, sondern gefunden werden will, muss man das Gerät sichtbar machen. Dafür wird in dem Sichtbar-Button die Methode ensureDiscoverable aufgerufen.

```
1 // Sichtbar-Button mit Listener
2 buttonEnsureDiscoverable = (Button)
     findViewById(R.id.buttonEnsureDiscoverable);
3           buttonEnsureDiscoverable.setOnClickListener(new
               OnClickListener() {
4                public void onClick(View v) {
5                    // Sichtbar machen
6                        ensureDiscoverable();
```

Listing 17.1: MainActivity.java

In dieser Methode wird ein Intent erzeugt, das beim Betriebssystem eine Anfrage stellt, die vom Benutzer bestätigt werden muss. Wenn sie bestätigt wird, soll das Gerät für 300 Sekunden sichtbar sein.

```
1 private void ensureDiscoverable() {
2   if (mBluetoothAdapter.getScanMode() !=
3     BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
4     Intent discoverableIntent = new
         Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
5     discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DUR
         300);
6     startActivity(discoverableIntent);
7   }
8 }
```

Listing 17.2: MainActivity.java

18 Verbindung herstellen

Bevor Nachrichten gesendet werden können, muss eine Verbindung bestehen. Diese wird in der BluetoothConnectionService erstellt. Für eine Bluetooth-Verbindung muss ein Gerät die Rolle des Servers und ein Gerät die Rolle des Client einnehmen. Da am Anfang nicht klar ist, welches Gerät welche Rolle bekommt, starten zunächst beide Geräte als Server und warten auf einen Client.

Um eine Verbindung herstellen zu können, benötigen sowohl Server als auch Client den Namen einer Verbindung und eine sogenannte UUID-Nummer, eine Nummer die pseudozufällig mit Hilfe eines UUID-Generators gebildet wird, den man im Internet finden kann. Über den Namen und die Nummer können die Apps sich gegenseitig finden und einen gemeinsamen Socket vereinbaren, über den dann die Daten gesendet werden können. Diese müssen also zunächst deklariert werden:

```
1 public class BluetoothConnectionService {
2         // Name fuer ServerSocket
3     private static final String NAME =
         "BluetoothConnectionTest";
4     // UUID
5     private static final UUID MY_UUID =
         UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");
```

Listing 18.1: BluetoothConnectionService.java

Da das Abhören eines ServerSockets den Thread blockiert, muss dies in einem extra Thread geschehen, damit der Rest der App weiter läuft. Daher wird dieser auch deklariert. Da für die Client-Seite und für den Datenaustausch dasselbe gilt, werden an dieser Stelle direkt alle drei Threads deklariert:

```
1 public class BluetoothConnectionService {
2         . . .
3     private AcceptThread mAcceptThread;
4     private ConnectThread mConnectThread;
5     private ConnectedThread mConnectedThread;
```

Listing 18.2: BluetoothConnectionService.java

Sie müssen dementsprechend auch implementiert werden, oder zunächst auskommentiert werden.

Bevor die einzelnen Thread implementiert werden, wird noch ein Hilfsmethode implementiert.

Über die Methode getState wir der aktuelle Status abgefragt:

```
1 public synchronized int getState() {
```

```
2           return mState;
3       }
```

Listing 18.3: BluetoothConnectionService.java

19 Der AcceptThread (Serverseite der Verbindung)

Anschließend wird die start Methode implementiert, die einen ServerSocket öffnen soll. Da der Server die Anfrage des Clients akzeptiert, heißt dieser Thread AcceptThread.

Bevor dieser aufgerufen wird, soll geprüft werden, dass kein ConnectThread, also ein Thread, der eine Verbindung als Client aufbauen will, geöffnet ist. Ansonsten wird dieser geschlossen. Falls bereits eine Verbindung mit einem anderen Gerät aufgebaut ist, wird diese auch geschlossen. Anschließend wird der Status auf STATE_LISTEN gesetzt und der AcceptThread geöffnet:

```
1  public synchronized void start() {
2    // Stopp Verbindungsaufbau als Client, falls vorhanden
3    if (mConnectThread != null) {
4      mConnectThread.cancel();
5      mConnectThread =  null;
6    }
7
8    // Stopp Verbindung, falls vorhanden
9    if (mConnectedThread != null) {
10     mConnectedThread.cancel();
11     mConnectedThread = null;
12   }
13
14   setState(STATE_LISTEN);
15
16   // Listen auf BluetoothServerSocket
17   if (mAcceptThread == null) {
18     mAcceptThread = new AcceptThread();
19     mAcceptThread.start();
20   }
21 }
```

Listing 19.1: BluetoothConnectionService.java

Der AcceptThread benötigt öffnet einen BluetoothServerSocket und lauscht auf diesem, ob ein anderes Gerät eine Anfrage zur Verbindung sendet. Da die Anfrage solange blockiert, wurde ein separater Thread gewählt. Da nur eine Verbindung hergestellt werden soll, wird nur solange geprüft, ob ein Gerät eine Verbindung herstellen will, bis dies mit einem Gerät geschehen ist. Sobald ein Gerät eine Verbindung anfragt, wird über die Methode accept ein socket zurückgegeben, über den die beiden Geräte danach verbunden sind und Daten austauschen können. Anschließend kann der ConnectedThread gestartet werde, über den der Datenaustausch stattfindet.

```
1  private class AcceptThread extends Thread {
2    // The local server socket
3    private final BluetoothServerSocket mmServerSocket;
4    private String mSocketType;
5
6    public AcceptThread() {
7      BluetoothServerSocket tmp = null;
8
9      // Create a new listening server socket
10     try {
11       tmp =
            mAdapter.listenUsingRfcommWithServiceRecord(NAME,
            MY_UUID);
12     }
13     catch (IOException e) {
14     }
15      mmServerSocket = tmp;
16   }
17
18   public void run() {
19     setName("AcceptThread" + mSocketType);
20
21     BluetoothSocket socket = null;
22
23     // Listen to the server socket if we're not connected
24     while (mState != STATE_CONNECTED) {
25       try {
26         // This is a blocking call and will only return on a
27         // successful connection or an exception
28         socket = mmServerSocket.accept();
29       }
30       catch (IOException e) {
31         break;
32       }
33
34       // If a connection was accepted
35       if (socket != null) {
36         synchronized (BluetoothConnectionService.this) {
37         switch (mState) {
38           case STATE_LISTEN:
39           case STATE_CONNECTING:
40             // Situation normal. Start the connected thread.
41             connected(socket, socket.getRemoteDevice(),
                  mSocketType);
42             break;
43           case STATE_NONE:
44           case STATE_CONNECTED:
45             // Either not ready or already connected.
                  Terminate new socket.
```

```
46                try {
47                  socket.close();
48                }
49                catch (IOException e) {
50                }
51              break;
52            }
53          }
54        }
55      }
56  }
```

Listing 19.2: BluetoothConnectionService.java

20 Der ConnectThread (Client Seite der Verbindung)

Die Client-Seite ist ähnlich aufgebaut. über die Methode device.creatRfcommSocketToServiceRecord(M spricht der Client den Server direkt an. Die Adresse hat er von der Liste bei der Auswahl übergeben bekommen. Über connect wird die Verbindung hergestellt. Diese muss nicht bestätigt werden, da sie selber angefragt wird. Es wird direkt der socket zurückgegeben. Über den Socket und das Gerät kann dann der ConnectedThread aufgerufen werden, der den Datenaustausch managet.

```
1  private class ConnectThread extends Thread {
2    private final BluetoothSocket mmSocket;
3    private final BluetoothDevice mmDevice;
4    private String mSocketType;
5
6    public ConnectThread(BluetoothDevice device, boolean
       secure) {
7      mmDevice = device;
8      BluetoothSocket tmp = null;
9      mSocketType = secure ? "Secure" : "Insecure";
10
11     // BluetoothSocket vom gewaehltem Geraet erfragen.
12     try {
13       tmp =
           device.createRfcommSocketToServiceRecord(MY_UUID);
14     }
15     catch (IOException e) {
16     }
17     mmSocket = tmp;
18   }
19
20   public void run() {
21     setName("ConnectThread" + mSocketType);
22
23     // Suche nach Geraten beenden
24     mAdapter.cancelDiscovery();
25
26     // Verbindungsaufbau zum BluetoothSocket
27     try {
28     mmSocket.connect();
29     }
30     catch (IOException e) {
```

```
31        // Close the socket
32        try {
33          mmSocket.close();
34        }
35        catch (IOException e2) {
36        }
37        connectionFailed();
38        return;
39      }
40
41      // Reset the ConnectThread because we're done
42      synchronized (BluetoothConnectionService.this) {
43        mConnectThread = null;
44      }
45
46      // Start the connected thread
47      connected(mmSocket, mmDevice, mSocketType);
48    }
49
50    public void cancel() {
51      try {
52        mmSocket.close();
53      }
54      catch (IOException e) {
55      }
56    }
57  }
```

Listing 20.1: BluetoothConnectionService.java

Die Methode connectionFailed muss noch implementiert werden. Sie gibt eine Fehlermeldung an den Handler weiter:

```
1  private void connectionFailed() {
2          // Sende Fehlermeldung an MainActivity
3          Message msg =
               mHandler.obtainMessage(MainActivity.MESSAGE_TOAST);
4          Bundle bundle = new Bundle();
5          bundle.putString(MainActivity.TOAST, "Unable to
               connect device");
6          msg.setData(bundle);
7          mHandler.sendMessage(msg);
8
9          // Start den BluetoothConnectionService neu
10         BluetoothConnectionService.this.start();
11     }
```

Listing 20.2: BluetoothConnectionService.java

Der Toast in der MainAcitivity muss noch deklariert werden.

```
1  private void connectionFailed() {
```

```
2         // Schluessel fuer Handler
3     public static final String TOAST = "toast";
```

Listing 20.3: MainAcitivity.java

21 Der ConnectionThread

Der ConnectionThread stellt die Verbindung her und kümmert sich um den Datenaustausch. Er kann über die Methode connect aufgerufen werden. Diese prüft, ob bereits eine Verbindung geöffnet ist, bevor eine neue geöffnet wird.

```
1  public synchronized void connect(BluetoothDevice device,
       boolean secure) {
2    // Beende vorhandene Verbindungen
3    if (mState == STATE_CONNECTING) {
4      if (mConnectThread != null) {
5        mConnectThread.cancel();
6        mConnectThread = null;
7      }
8    }
9
10
11   if (mConnectedThread != null) {
12       mConnectedThread.cancel();
13       mConnectedThread = null;
14   }
15
16   // Start den Thread
17   mConnectThread = new ConnectThread(device, secure);
18   mConnectThread.start();
19   setState(STATE_CONNECTING);
20 }
```

Listing 21.1: BluetoothConnectionService.java

Eine weitere Methode, die Methode connected schließt den AcceptThread und den ConnectedThread, sobald die Verbindung hergestellt wurde, da diese dann nicht mehr benötigt werden. Anschließend wird de ConnectedThread aufgerufen und dem Handler mitgeteilt, mit welchem Gerät sich verbunden wurde. Der Name wird als Stringgespeichert, damit er später bei der Ausgabe der Nachrichten voran gesetzt werden kann.

```
1  public synchronized void connected(BluetoothSocket socket,
       BluetoothDevice device, final String socketType) {
2    // Schliesse Thread nach erfolgreicher Verbindung
3    if (mConnectThread != null) {
4      mConnectThread.cancel();
5      mConnectThread = null;
6    }
7
8    // Schliesse alle anderen Verbindungen
```

```
9   if (mConnectedThread != null) {
10    mConnectedThread.cancel();
11    mConnectedThread = null;
12  }
13
14  // Schliesse Thread nach erfolgreicher Verbindung
15  if (mAcceptThread != null) {
16    mAcceptThread.cancel();
17    mAcceptThread = null;
18  }
19
20  // Starten des ConnectedThreads
21  mConnectedThread = new ConnectedThread(socket, socketType);
22  mConnectedThread.start();
23
24  // Sende den Geratenamen
25  Message msg =
       mHandler.obtainMessage(MainActivity.MESSAGE_DEVICE_NAME);
26  Bundle bundle = new Bundle();
27  bundle.putString(MainActivity.DEVICE_NAME,
       device.getName());
28  msg.setData(bundle);
29  mHandler.sendMessage(msg);
30
31  setState(STATE_CONNECTED);
32 }
```

Listing 21.2: BluetoothConnectionService.java

Die Konstante DEVICE_NAME wird in der MainActivity deklariert.

```
1 // Schluessel fuer Handler
2     public static final String TOAST = "toast";
3     public static final String DEVICE_NAME = "device_name";
```

Listing 21.3: MainActivity.java

Der ConnectedThread regelt den Datenaustausch. Er benötigt einen Socket und einen Input- und Outputstream. Nachdem diese erstellt sind, kann er Daten lesen in dem er den Inputstream abhört und die Daten in ein Byte-Array umspeichert und dann dem Handler übergibt.

Ebenso kann er schreiben in dem er ein Byte-Array in den Outputstream schreibt.

```
1 private class ConnectedThread extends Thread {
2   private final BluetoothSocket mmSocket;
3   private final InputStream mmInStream;
4   private final OutputStream mmOutStream;
5
6   public ConnectedThread(BluetoothSocket socket, String
       socketType) {
7     mmSocket = socket;
```

```
8     InputStream tmpIn = null;
9     OutputStream tmpOut = null;
10
11    // BluetoothSocket input and output stream zuordnen
12    try {
13      tmpIn = socket.getInputStream();
14      tmpOut = socket.getOutputStream();
15    }
16    catch (IOException e) {
17    }
18
19    mmInStream = tmpIn;
20    mmOutStream = tmpOut;
21  }
22
23  public void run() {
24    byte[] buffer = new byte[1024];
25    int bytes;
26
27    // InputStream abhoeren, solange Verbindung besteht
28    while (true) {
29      try {
30        // InputStream einlesen
31        bytes = mmInStream.read(buffer);
32
33        // Handler informieren
34        mHandler.obtainMessage(MainActivity.MESSAGE_READ,
             bytes, -1, buffer).sendToTarget();
35      }
36      catch (IOException e) {
37        connectionLost();
38        // Service neustarten
39        BluetoothConnectionService.this.start();
40        break;
41      }
42    }
43  }
44
45
46  public void write(byte[] buffer) {
47    try {
48      mmOutStream.write(buffer);
49
50      mHandler.obtainMessage(MainActivity.MESSAGE_WRITE, -1,
           -1, buffer).sendToTarget();
51    }
52    catch (IOException e) {
53    }
54  }
```

```
55
56   public void cancel() {
57     try {
58       mmSocket.close();
59     }
60     catch (IOException e) {
61     }
62   }
63 }
```

Listing 21.4: BluetoothConnectionService.java

Hierfür muss noch die Methode connectionLost implementiert werden, die sich darum kümmert, dass die Verbindung neu gestartet wird, wenn die Verbindung verloren geht.

```
1 private void connectionLost() {
2         // Fehlermeldung an MainAcitivity senden
3         Message msg =
             mHandler.obtainMessage(MainActivity.MESSAGE_TOAST);
4         Bundle bundle = new Bundle();
5         bundle.putString(MainActivity.TOAST, "Device
             connection was lost");
6         msg.setData(bundle);
7         mHandler.sendMessage(msg);
8
9         // Neustart
10        BluetoothConnectionService.this.start();
11    }
```

Listing 21.5: BluetoothConnectionService.java

Zum Schreiben in den ConnectedThread wird noch die Methode write implementiert, welche sicherstellt, dass keine parallelen Zugriffe erfolgen.

```
1 public void write(byte[] out) {
2
3         ConnectedThread r;
4         // Synchronisiere eine Kope des ConnectedThread
5         synchronized (this) {
6                 if (mState != STATE_CONNECTED) return;
7                 r = mConnectedThread;
8         }
9         // Schreiben
10        r.write(out);
11 }
```

Listing 21.6: BluetoothConnectionService.java

Damit ist die Service-Klasse im Prinzip fertig. Es fehlt nur noch eine stop Methode, die alle Threads beendet.

```
1 public synchronized void stop() {
2         if (mConnectThread != null) {
```

```
3                   mConnectThread.cancel();
4                   mConnectThread = null;
5           }
6
7           if (mConnectedThread != null) {
8                   mConnectedThread.cancel();
9                   mConnectedThread = null;
10          }
11
12          if (mAcceptThread != null) {
13                  mAcceptThread.cancel();
14                  mAcceptThread = null;
15          }
16
17          setState(STATE_NONE);
18  }
```

Listing 21.7: BluetoothConnectionService.java

22 Der Handler

Nachdem die Service-Klasse damit abgeschlossen ist, muss der Handler die Informationen auch noch in der MainActivity verarbeiten. Dazu fragt er weiter den Status ab. Im Falle des Schreibens, wandelt er das empfangene Objekt in ein Byte-Array um und gibt es mit der Voranstellung von Me: an den MessageAdapter.

Beim Lesen passiert dasselbe, nur dass dieses Mal der Gerätename des Kommunikationspartners vorangestellt wird.

Wird eine Nachricht mit dem Gerätenamen übergeben, so wird per Popup darauf hingewiesen, dass sich mit diesem Gerät verbunden wurde.

```
1  // Handler zur Verarbeitung der Informationen von
       BluetoothConnectionService
2  private final Handler mHandler = new Handler() {
3
4    @Override
5    public void handleMessage(Message msg) {
6      // TODO Auto-generated method stub
7      super.handleMessage(msg);
8      switch (msg.what) {
9      case MESSAGE_STATE_CHANGE:
10       switch (msg.arg1) {
11       case BluetoothConnectionService.STATE_CONNECTED:
12         setStatus(getString(R.string.title_connected_to,
             mConnectedDeviceName));
13         mMessagesArrayAdapter.clear();
14         break;
15       case BluetoothConnectionService.STATE_CONNECTING:
16         setStatus(R.string.title_connecting);
17         break;
18       case BluetoothConnectionService.STATE_LISTEN:
19       case BluetoothConnectionService.STATE_NONE:
20         setStatus(R.string.title_not_connected);
21         break;
22       }
23       break;
24     case MESSAGE_WRITE:
25       byte[] writeBuf = (byte[]) msg.obj;
26       // construct a string from the buffer
27       String writeMessage = new String(writeBuf);
28       mMessagesArrayAdapter.add("Me:  " + writeMessage);
29       break;
```

```
30      case MESSAGE_READ:
31        byte[] readBuf = (byte[]) msg.obj;
32        // construct a string from the valid bytes in the
             buffer
33        String readMessage = new String(readBuf, 0, msg.arg1);
34        mMessagesArrayAdapter.add(mConnectedDeviceName+":  " +
             readMessage);
35        break;
36      case MESSAGE_DEVICE_NAME:
37        // save the connected device's name
38        mConnectedDeviceName =
             msg.getData().getString(DEVICE_NAME);
39        Toast.makeText(getApplicationContext(), "Connected to
             " + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
40        break;
41      case MESSAGE_TOAST:
42        Toast.makeText(getApplicationContext(),
             msg.getData().getString(TOAST),
             Toast.LENGTH_SHORT).show();
43        break;
44      }
45    }
46 };
```

Listing 22.1: MainActivity.java

23 Nachrichten senden

Nachrichten werden gesendet, wenn der Senden-Button betätigt wird. Also muss die OnCLick Methode des Sende-Buttons dafür implementiert werden. In dieser Methode liest man den Text des Textfeldes aus und speichert ihn in einem String. Dieser String wird dann eine Methode sendMessage übergeben, die sich um das eigentliche Senden kümmert.

```
1 buttonSendMessage = (Button)
     findViewById(R.id.buttonSendMessage);
2         buttonSendMessage.setOnClickListener(new
             OnClickListener() {
3                 public void onClick(View v) {
4
5                         String message =
                             editTextMessage.getText().toString();
6                         sendMessage(message);
7                 }
8         });
```

Listing 23.1: MainActivity.java

Die Methode sendMessage prüft, ob überhaupt eine Verbindung besteht. Ansonsten wird eine Meldung zurückgegeben. Falls eine Verbindung besteht, wird geprüft, ob überhaupt etwas in dem Textfeld stand, also jetzt in dem String. Ist dies der Fall, wird der String in ein Byte-Array umgewandelt und an die Service-Klasse übergeben.

Anschließend werden der OutStringBuffer und das Textfeld noch geleert.

```
1 private void sendMessage(String message) {
2         // Pruefen, ob verbunden
3         if (mBluetoothConnectionService.getState() !=
             BluetoothConnectionService.STATE_CONNECTED) {
4                 Toast.makeText(this, R.string.not_connected,
                     Toast.LENGTH_SHORT).show();
5                 return;
6         }
7
8         // Wurde was geschrieben?
9         if (message.length() > 0) {
10
11                byte[] send = message.getBytes();
12                mBluetoothConnectionService.write(send);
13
14                // Buffer und Textfeld leeren
15                mOutStringBuffer.setLength(0);
```

```
16                  editTextMessage.setText(mOutStringBuffer);
17          }
18 }
```

Listing 23.2: MainActivity.java

Der String not_connected muss noch in die string.xml eingefügt werden:

```
1          <!--  Bluetooth Connection -->
2      <string name="bt_not_enabled_leaving">Bluetooth ist
         deaktiviert. Beende App.</string>
3      <string name="title_connecting">verbinde...</string>
4      <string name="title_connected_to">verbunden</string>
5      <string name="title_not_connected">nicht
         verbunden</string>
6      <string name="not_connected">Sie sind nicht mit einem
         Geraet verbunden</string>
```

Listing 23.3: string.xml

Damit ist die App fertig.