Bluetooth

także dla Ciebie, czyli programowanie „w Androidzie”

Wielu osobom wydaje się, że zdalne sterowanie przez Bluetooth za pomocą smartfona czy tableta to podwójnie trudne zadanie. Bo z jednej strony trzeba przygotować sprzęt z modułem Bluetooth, jakimś mikroprocesorem i go zaprogramować, a z drugiej trzeba napisać odpowiednią aplikację dla smartfona/tableta.
Poniższy artykuł ma udowodnić, że zadanie wcale nie jest trudne i że może sobie z tym poradzić każdy, kto nie boi się, ma szczere chęci i choć odrobinę wiedzy o programowaniu w C i C++. Aby nie przestraszyć mniej doświadczonego Czytelnika, przedstawiono dwie wersje programu dla smartfona: podstawową aplikację uniwersalną oraz gotową aplikację dedykowaną. Opisano także (zaskakująco prosty) sposób tworzenia, a właściwie modyfikacji aplikacji za pomocą pakietu Android Studio.

Celem artykułu nie jest przedstawienie pełnego kursu programowania na platformę Android. Chodzi o to, aby ośmielić, zachęcić i przedstawić możliwie prosty sposób na przygotowanie w pełni użytecznej aplikacji, umożliwiającej wyświetlenie danych i sterowanie urządzeń z telefonu. I wszystko to bez konieczności dogłębnego zrozumienia języka Java! Opis powstał głównie z myślą o osobach, które mają ciekawy pomysł na praktyczne urządzenie, lecz nie wiedzą, jak ugryźć przygotowanie programów dla telefonów. Okazuje się, że szybko i prosto: przyjazny i estetyczny interfejs użytkownika można przygotować przeciągając elementy za pomocą myszki, natomiast do jego programowej obsługi wystarczy tylko kilka prostych linijek kodu.

Wszystko jest wytłumaczone i podane w artykule, a zarówno „półprodukt”, jak i finalny program dla smartfona są dostępne w Elportalu. Żaden chętny nie powinien więc mieć trudności z wykorzystaniem prezentowanego projektu.

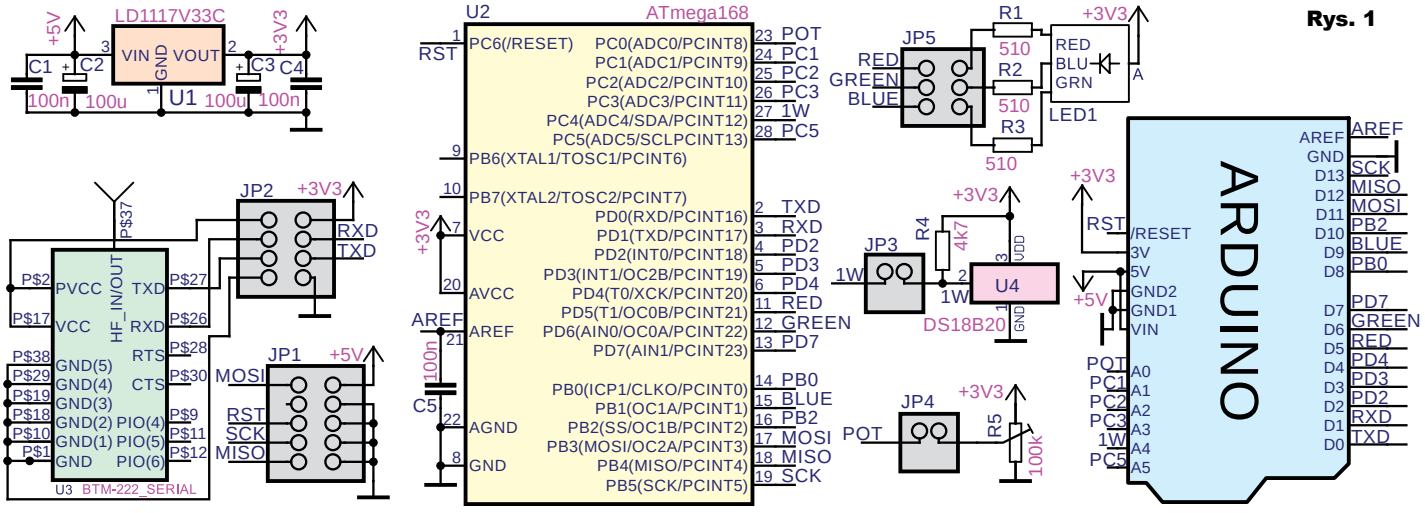
Artykuł przedstawia nieskomplikowaną płytkę, zawierającą moduł Bluetooth BT-222 i mikroprocesor, które pozwalają za pomocą smartfona obsługiwać umieszczoną tam diodę LED RGB, czujnik temperatury DS18B20 i... potencjometr. Zasadniczo płytka ma posłużyć do nauki. Jednak jest tak pomyślana, że jej moduł Bluetooth i procesor można później wykorzystać także do współpracy z dowolnymi innymi urządzeniami. Dzięki wyprowadzeniom zgodnym z Arduino, da się do niej podłączać przeznaczone dla tej platformy rozszerzenia.

Przedstawiony materiał udowadnia, że obsługa modułów Bluetooth od strony mikrokontrolera jest bardzo prosta. A po oswojeniu się z tą platformą także napisanie programu na telefon czy tablet z systemem Android wcale nie jest trudne, jak sądzi wiele osób. Zwłaszcza, że nie jest to pisanie programu od początku, tylko modyfikacja istniejącego „szkieletu”.

Przedstawiony program był testowany na telefonie Sony Ericsson Xperia Neo V z systemem Android 2.3 oraz Samsung GT-S6310N z systemem Android 4.1.2. Program działał także na tablecie Lenovo IdeaTab z systemem Android 4.0.3, lecz czasem występowały wtedy problemy z połączeniem się z modułem Bluetooth.

Opis układu

Układ elektroniczny modułu jest nieskomplikowany. Składa się tylko z kilku prostych elementów, służących do zilustrowania przykładowych programów. Schemat układu znajduje się na **rysunku 1**. Za sterowanie całością odpowiada mikrokontroler ATmega168 (U2), połączony z popularnym modułem Bluetooth BTM-222 (U3). Procesor ten współpracuje też z termometrem DS18B20 (U4) oraz odczytuje napięcie z potencjometru R5 i steruje kolorem diody LED1. I to właśnie przez łącze Bluetooth zrealizowana jest współpraca z tymi trzema elementami.



Rys. 1

Artykuł przedstawia prosty, edukacyjny przykład wykorzystania płytki, ale może ona być użyta do innych celów. Goldpiny JP2-JP5 pozwalają bowiem odłączyć peryferie od mikrokontrolera, a złącza zgodne z Arduino pozwolą później wykorzystać układ do własnych eksperymentów.

Urządzenie może być zasilane napięciem stałym 5V podpiętym do złącza programatora JP1 lub do wyprowadzeń Arduino. Układ U1 stabilizuje je do 3,3V, bowiem takie napięcie jest wymagane przez moduł BTM-222. Aby nie trzeba było stosować konwertera poziomów logicznych, mikrokontroler także jest zasilany napięciem 3,3V.

Teoretycznie można byłoby zasilać moduł bez stabilizatora U1, jednak trzeba pamiętać, że według karty katalogowej BT-222 zakres zasilania części radiowej modułu (PVCC) to 3...3,3V, natomiast część cyfrowa może być zasilana napięciem (VDD) 2,7...3,6V.

Program w procesorze

Program dla mikrokontrolera został napisany w języku C. Jest on krótki i bardzo prosty. W Elportalu wśród materiałów dodatkowych do tego numeru można znaleźć gotowy wsad, który znajduje się w pliku ***main.hex***. Należy także skonfigurować fusebajty. Dolny powinien mieć wartość 0xE2, a górny 0xDF.

Oprócz tego dla bardziej zaawansowanych są tam też wszystkie pliki źródłowe, pozwalające na własne eksperymenty. Program został podzielony na kilka plików: ***main.c*** zawiera pętlę główną, natomiast każdy z pozostałych z rozszerzeniem **.c** zawiera funkcję obsługującą inne peryferie mikrokontrolera. Odbiór danych z UART-u jest realizowany w przerwaniu. Pętla główna sprawdza, czy została odebrana nowa komenda, a jeśli tak, wykonuje ją. Dla uproszczenia dane są przesyłane jako ciągi znaków ASCII. Jest to sposób znacznie mniej wydajny niż wysyłanie w postaci binarnej, ale za to dane są zdecydowanie bardziej czytelne dla użytkownika. Układ rozumie trzy komendy przedstawione w **tabeli 1**.

Montaż i uruchomienie

Projekt płytki drukowanej prezentuje **rysunek 2**. Uwaga! Fotografie modelu pokazują wcześniejszą wersję, mniej uniwersalną, niekompatybilną z Arduino.

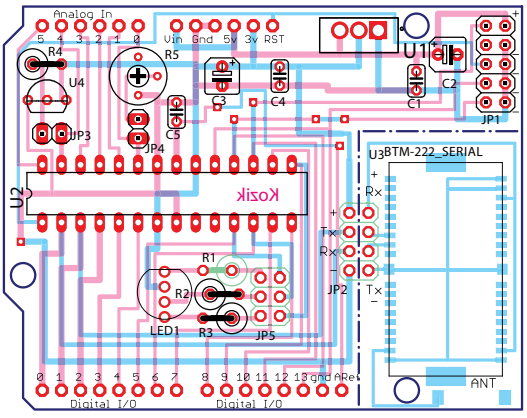
Układ, poza modułem Bluetooth, składa się z elementów przewlekanych. Montaż warto rozpocząć od przylutowania układu BTM-222. Na początku, przy użyciu pęsety układamy go tak, aby jak najdokładniej pasował do płytki. Dociskając pęsetą, aby się nie poruszył, lutujemy pierwsze złącze. Następnie sprawdzamy, czy układ się nie przesunął, ponieważ teraz da się go jeszcze stosunkowo łatwo odlutować. Jeśli wszystko pasuje, lutujemy po kolei każde złącze osobno. Gdy powstaną zwarcia, do ich usunięcia bardzo pomocna jest kalafonia oraz plecionka lutownicza. Stabilizator U1 oraz kondensatory elektrolityczne C2 i C3 warto zamontować w pozycji „leżącej", aby nie kolidowały z płytkami rozszerzeń. Z tego powodu należy także zamontować pozostałe elementy możliwie nisko. Jako złącza JP2–JP5 najlepiej użyć wtyków goldpin, na które można założyć zworki. Jeśli chodzi o zaznaczoną na schemacie antenę, na płytce zostało przewidziane miejsce na przylutowanie *anteny wewnętrznej Bluetooth* na pasmo 2,5GHz. Jej dodanie powinno zwiększyć zasięg, jednak nie jest ona niezbędna.

Przykład anteny wewnętrznej SMD: *https://sklep.avt.pl/antena-bluetooth-wewnetrzna-smd-2-45ghz.html* w skrócie *https://goo.gl/KlN2Yx*.

Przed włożeniem mikrokontrolera do podstawki warto woltomierzem sprawdzić, czy napięcie zasilania jest równe 3,3V. Jeśli tak, można włożyć procesor oraz założyć zworki na złącza JP2-JP5. Po wgraniu programu (dostępnego w Elportalu *main.hex*) powinna zaświecić się dioda LED. Najlepiej użyć diody RGB z matową soczewką lub nałożyć na nią coś, co rozproszy światło, aby miało ono jednorodny kolor.

Dzięki złączom JP2–JP5 można odłączyć peryferie i użyć płytki do własnych celów. Aby wykonać opisane eksperymenty, należy na nie nałożyć zworki.

Rys. 3



Rys. 2

Po ich zdjęciu da się wykorzystać zarówno mikrokontroler, jak i peryferie do innych doświadczeń. Szczególnie przydatny do eksperymentów może być moduł Bluetooth, który można za pomocą przewodów podłączyć do płytki stykowej. Można nawet przeciąć płytkę i podłączyć go dłuższymi przewodami. Dzięki temu po skończeniu eksperymentów układ znajdzie kolejne interesujące zastosowania.

Bluetooth najprościej

Moduł BTM-222 jest gotowy do pracy od razu po podłączeniu do niego napięcia zasilania (ważne: musi to być 3...3,3V). Procesor komunikuje się z nim w bardzo prosty sposób: aby wysłać dane, wystarczy przesłać do modułu bajt przez interfejs UART z ustawieniami: prędkość 19200, długość ramka: 8 bitów, z jednym bitem stopu i bez bitu parzystości.

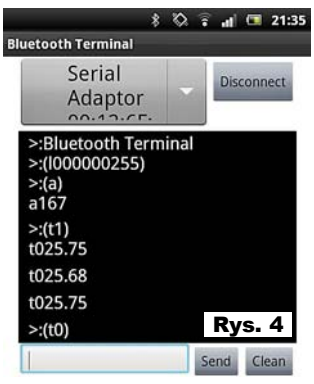
Aby przekonać się, że nasz prosty układ naprawdę działa, można włączyć Bluetooth w telefonie i wyszukać dostępne urządzenia. Powinno pojawić się jedno o nazwie *Serial adaptor* tak jak na **rysunku 3**. Klikamy go i parujemy z nim telefon. Zostaniemy poproszeni o hasło, które domyślnie jest ustawione na **1234**. Najprostszym sposobem komunikacji jest użycie jednego ze „smartfonowych" terminali umożliwiających przesyłanie tekstu przez Bluetooth, na przykład *BluetoothTerminal* (wersja na Androida dostępna w Google Play, np. pod adresem: *https://play.google.com/store/apps/details?id=ptah.apps.bluetoothterminal)*.

Pozwala on odebrać i wysyłać znaki ASCII. Jest on bardzo wygodny do wstępnych eksperymentów i wystarczający do wielu prostych projektów: nie trzeba pisać swojego oprogramowania dla telefonu. Korzystając z niego, można szybko sprawdzić działanie oprogramowania wgranego do mikrokontrolera.

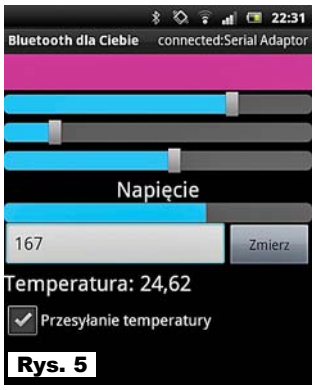
Wygląd menu programu pokazuje **rysunek**

Tabela 1

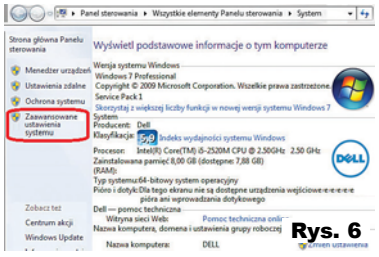
Odbierane:		Wysyłane:	
Komenda	**Funkcja**	**Komenda**	**Funkcja**
(lrrrgggbbb)	Ustawia jasność diod. rrr, ggg, bbb to liczby dziesiętne od 000 do 255	txxx.xx	Aktualna temperatura
(tx)	x=1/0 włącza/wyłącza przesyłanie temperatury co 2 sekundy	axxx	Odczyt z ADC
(a)	Zwraca liczbę od 0 do 255 proporcjonalną do napięcia na R5		



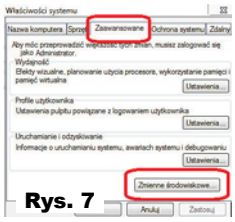
Rys. 4



Rys. 5



Rys. 6

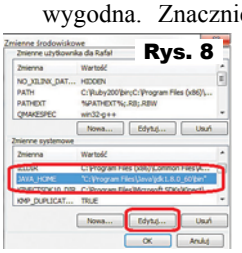


Rys. 7

4. Z listy na górze wybieramy *Serial Adaptor* i klikamy *Connect* (ang. *połącz*). Po udanym połączeniu możemy przesyłać komendy. Ich lista znajduje się w tabeli 1. Zachęcam do wypróbowania działania opisywanej płytki z *BluetoothTerminal*. Na przykład jeżeli chcemy zmienić kolor diody LED na niebieski, wpisujemy *(I000000255)*. Każde trzy kolejne cyfry odnoszą się odpowiednio do natężenia światła struktury czerwonej, zielonej i niebieskiej. Wartość 000 odpowiada wygaszeniu, a 255 maksymalnej jasności. A jeżeli chcemy odczytać wartość napięcia na potencjometrze, wpisujemy komendę *(a)*. Mikrokontroler odpowie nam wtedy liczbą od 0 (odpowiada masie) do 255 (dodatnia szyna zasilania).

Przesłanie komendy *(t1)* powoduje, że co 2 sekundy mikrokontroler przesyła aktualną temperaturę. Wpisanie *(t0)* wyłącza tę opcję. Przykładowy przebieg „rozmowy" z modułem przedstawia rysunek 4. Gdy chcemy zakończyć połączenie, klikamy przycisk *disconnect* (ang. *rozłącz*). I to jest pierwszy, bardzo prosty sposób komunikacji, który też daje dużo satysfakcji, gdyż możemy porozumieć się z naszym modułem. Jednak taka obsługa nie jest ani zbyt elegancka, ani wygodna. Znacznie lepiej jest przygotować własną aplikację. Program, który przygotujemy w tym module, można także pobrać z Google Play spod adresu: *play.google.com/store/apps/details?id=tk.kozik.bdc*.

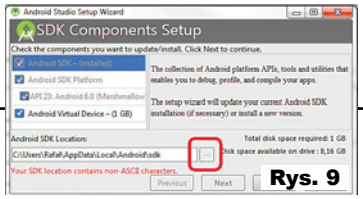
Jej wygląd przedstawia **rysunek 5**. Przyciski, suwaki i pola tekstowe pozwalają przygotować znacznie przyjemniejszy interfejs użytkownika niż wpisywanie komend. Przykłady będą podane w języku Java. Postaram się jednak opisać je w taki sposób, aby nawet osoby nieznające tego języka, ale obeznane choćby tylko z podstawami C\C++, poradziły sobie z ich zrozumieniem i dostosowaniem do własnych potrzeb.
Nie bój się – niczym nie ryzykujesz! Spróbuj!


Rys. 8

Instalacja Android Studio

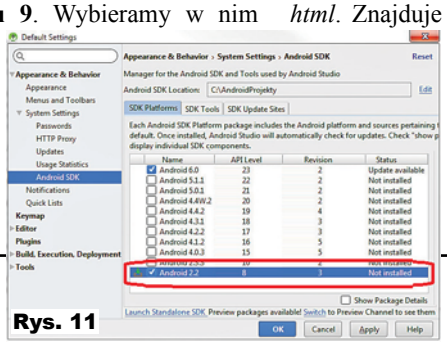
Środowisko programistyczne umożliwiające pisanie pod system Android dostarcza za darmo firma Google. Można je pobrać spod adresu: *http://developer.android.com/sdk*. Niestety jest ono całkiem spore – będziemy potrzebować około 4 GB wolnego miejsca. Po pobraniu uruchamiamy ściągnięty plik. Proces instalacji przebiega standardowo, lecz trwa dość długo. Do uruchomienia Android Studio niezbędny jest Java Development Kit w wersji 1.7, który można pobrać ze strony: *http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html*. Poprawna ścieżka do folderu instalacyjnego Java JDK musi być zapisana w zmiennej systemowej **JAVA_HOME**. Aby ją zmienić, wybieramy *Start->Komputer*. Następnie klikamy prawym przyciskiem myszy i wybieramy *Właściwości*. Pojawi się okno jak na **rysunku 6**. Klikamy *Zaawansowane ustawienia systemu*. Pojawi się okno jak na **rysunku 7**. Wybieramy zakładkę *Zaawansowane*. Klikamy przycisk *zmienne systemowe*. W kolejnym oknie (**rysunek 8**) w polu *Zmienne systemowe* odnajdujemy zmienną **JAVA_HOME** i klikamy *Edytuj*. Warto zapisać starą wartość tej zmiennej, aby w razie problemów móc ją przywrócić. Inne programy mogą wymagać innych ustawień.

Przy pierwszym uruchomieniu Android Studio pojawi się okno ustawień takie jak na **rysunku 9**. Wybieramy w nim katalog, w którym chcemy doinstalować Android SDK – dodatkowy pakiet niezbędny do tworzenia aplikacji. Klikamy *Finish*. Pobranie i instalacja zajmie kolejne kilka minut. Po pomyślnej instalacji zobaczymy ekran powitalny taki jak na **rysunku 10**. Klikamy w nim *Configure* i w kolejnym oknie *SDK Manager*. Pojawi się okno jak na **rysunku 11**. Zaznaczamy *Android 2.2* i klikamy *OK*. Pojawia się kolejne okno, gdzie znów klikamy *OK*.



Rys. 9

Rys. 10



Rys. 11

Pierwsza aplikacja

Żeby tworzenie pierwszej aplikacji przebiegło (dużo) łatwiej, jako podstawę wykorzystamy przykładowy program *BluetoothChat* pochodzący ze strony *http://blog.bastelhalde.de/?p=274*. Kod projektu znajduje się też w Elportalu w folderze */java* w materiałach dodatkowych do artykułu. Po pobraniu i rozpakowaniu archiwum otwieramy Android Studio. W ekranie powitalnym klikamy *Open existing Android Studio projekt* (otwórz istniejący projekt) i odszukujemy folder, do którego rozpakowaliśmy pliki. Są tam przygotowane trzy projekty: ***szkielet*** jest początkowym projektem, od którego zaczynamy. Projekt ***LED*** zawiera zmiany, które wprowadzimy w części o sterowaniu diodami LED, a ***odczyt_danych*** to gotowa aplikacja. W zasadzie wszystko jest więc gotowe, zachęcam jednak do samodzielnego modyfikowania „szkieletu" i później do stworzenia własnych aplikacji komunikujących się przez Bluetooth.
Po otwarciu aplikacji zobaczymy okno programu jak na **rysunku 12**. Po lewej stronie widzimy strukturę plików projektu. Jeżeli się nie pojawia, klikamy zakładkę *Project*. Większość ekranu zajmuje obszar edytora.

Debugowanie w telefonie.
Aby uruchomić przykładowy program w telefonie komórkowym, musimy najpierw odblokować w nim tryb debugowania. W telefonie pracującym pod kontrolą Androida 2.3 wchodzimy w *Ustawienia*, następnie *Aplikacje* i *Dla programistów*, gdzie aktywujemy *Debugowanie USB*. Natomiast w Androidzie 4.0 wchodzimy w *Ustawienia systemu*, następnie w *Opcje programisty* i tam aktywujemy *Debugowanie USB*. Musimy jeszcze pobrać sterowniki dla telefonu ze strony *http://developer.android.com/tools/extras/oem-usb.html*. Znajduje się tam lista odnośników do stron poszczególnych producentów telefonów.

Podłączamy telefon do komputera za pomocą kabla USB. Gdy komputer odnajdzie nasz telefon, wchodzimy

w *Start->Urządzenia i drukarki*. Klikamy dwa razy nasz telefon i powinniśmy zobaczyć okno jak na **rysunku 13**. Z listy wybieramy to urządzenie, które nie ma poprawnie zainstalowanego sterownika i klikamy *właściwości*. Pojawi się kolejne okno jak na **rysunku 14**. Klikamy *Zmień ustawienia,* a następnie *Aktualizuj sterownik*. W kolejnym oknie klikamy *Przeglądaj mój komputer w poszukiwaniu oprogramowania sterownika*. Wybieramy *Pozwól mi wybrać z listy sterowników na moim komputerze*. W kolejnym wybieramy *Pokaż wszystkie urządzenia* i klikamy *dalej*. W następnym klikamy *Z dysku* i otwieramy plik z lokalizacji, gdzie pobraliśmy sterowniki i naciskamy *OK*. Na pytanie, czy na pewno chcemy zainstalować ten sterownik, klikamy *Tak*. Po zakończeniu instalacji klikamy *Zamknij*.

Następnie w oknie Android Studio, mając otwarty projekt, naciskamy przycisk *Run* (zielona strzałka) zaznaczony na **rysunku 12** czerwoną obwódką.

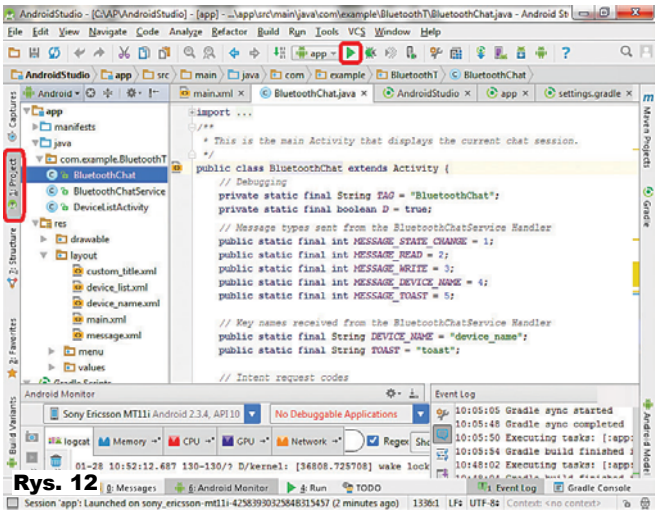
Gdy pojawi się jakiś błąd, warto skopiować jego treść i wkleić w Google. Jest duża szansa, że ktoś go już wcześniej spotkał. Gdy wszystko pójdzie poprawnie, zobaczymy okno, w którym możemy wybrać urządzenie, na którym chcemy zainstalować aplikację (**rysunek 15**). Klikamy dwukrotnie nasze urządzenie. Program zostanie zainstalowany, a następnie uruchomiony.

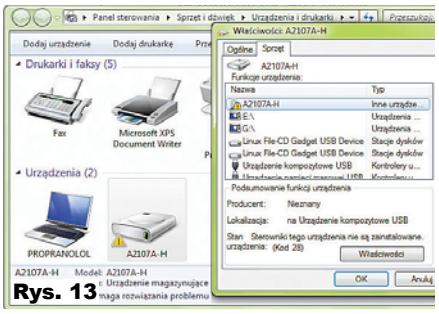
Teraz już możemy korzystać z naszej aplikacji jak z każdej innej zainstalowanej w telefonie, także po odłączeniu kabla USB. Przy uruchomieniu aplikacji zostaniemy poproszeni o zgodę na uruchomienie Bluetooth w telefonie. Jednak jest to jedynie pusty szkielet. Potrafi tylko nawiązać połączenie z modułem Bluetooth. Jak przerobić go na praktyczną aplikację, dowiemy się za chwilę.

Struktura plików.

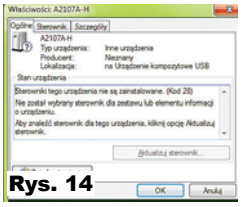
Drzewo plików aplikacji na Androida jest rozległe, jednak nie bój się tego: nas interesuje tylko kilka najważniejszych miejsc.

W folderze **/java** (na dysku: *...projekt.../app/src/main/java/com/example/BluetoothT/*) znajdują się pliki z kodem źródłowym aplikacji. Dobrym zwyczajem jest, aby każda klasa była napisana w osobnym pliku. Nasz projekt zawiera trzy klasy: *BluetoothChatService* zajmującą się wysyłaniem i odbieraniem danych, *DeviceListActivity* wyświetlającą listę dostępnych urządzeń oraz *BluetoothChat* zajmującą się interfejsem użytkownika.


Rys. 12


Rys. 13

W tym projekcie będziemy modyfikować tylko tę ostatnią. Folder **/res** (dokładniej: *...projekt.../app/src/main/res/*) zawiera zasoby takie jak obrazki, teksty czy „layouty” (wygląd okien aplikacji). Większość tych danych jest przechowywana w postaci plików XML.


Rys. 14


Rys. 15

Nie mój się tych informacji. Jeśli masz obawy lub wątpliwości – skopiuj projekty (w sumie 1,22MB) i rozpakuj do oddzielnego folderu, by bez obaw w nich „pogrzebać” w celu bliższego zapoznania się i stwierdzenia, że większość folderów jest... zupełnie pusta.

Sterowanie diodą LED.

Nasz pierwszy program będzie zajmował się sterowaniem diodą LED. Składa się z przycisku w kolorze świecenia diody oraz 3 suwaków do zmiany jasności poszczególnych kanałów.

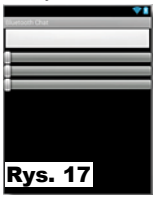
W Elportalu, w materiałach dodatkowych w folderze **/szkielet.zip** znajduje się szkielet aplikacji obsługującej Bluetooth. Na bazie tego szkieletu można tworzyć własne programy. Zawiera on całą obsługę Bluetooth, taką jak znajdowanie urządzeń, łączenie się oraz odbieranie i wysyłanie wiadomości. Na jego podstawie szybko można stworzyć dedykowaną aplikację do swojego projektu. Aby rozpocząć, rozpakowujemy folder i otwieramy projekt w Android Studio.

Najpierw przygotujemy interfejs użytkownika. W tym celu otwieramy plik **.../res/layout/main.xml** przez dwukrotne kliknięcie go w drzewie folderów w Android Studio. Po chwili pojawi się graficzny edytor, gdzie za pomocą myszki możemy stworzyć wygląd programu (**rysunek 16**). Przeciągamy przycisk (*Button*) oraz 3 suwaczki (*seekBar*). Wszystkie dodane elementy pojawiają się w okienku aplikacji oraz w oknie *Component Tree* po prawej stronie. Po kliknięciu wybranego elementu pojawią się jego właściwości w zakładce *Properties*. Znajdujemy pole **Id** przycisku i zmieniamy jego wartość na *color*. To będzie nazwa, pod którą nasz przycisk będzie „widoczny” w kodzie programu. Następnie kasujemy wyświetlany na nim tekst, usuwając wartość z pola **Text**. Możemy jeszcze rozszerzyć myszką przycisk, aby zajął całą szerokość ekranu. W tym celu chwytamy prawą krawędź i rozciągamy ją do boku ekranu, aż pojawi się tekst *match parent*. Zmieniamy **id** suwaków na *red*, *green* i *blue* oraz wartości pola **max** na *255*. Przykładowy gotowy layout przedstawia **rysunek 17**. Teraz pozostało zmodyfikować logikę aplikacji. Otwieramy plik *BluetoothChat.java*. Najpierw tworzymy trzy zmienne wewnątrz klasy *BluetoothChat*. Klasa zaczyna się za linijką:

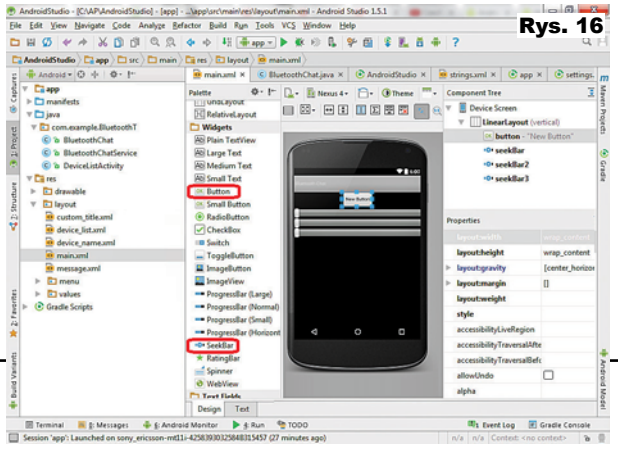
```
public class BluetoothChat extends
Activity {
```

i za nią wpisujemy:

```
private SeekBar red;
private SeekBar green;
private SeekBar blue;
private Button color;
```


Rys. 17

Po wpisaniu pojawi się błąd: *SeekBar can not be resolved to a type*. Oznacza on, że nie udało


Rys. 16

się znaleźć obiektu o takiej nazwie. Błąd ten wystąpił, ponieważ nie dodaliśmy odpowiednich bibliotek. W C należałoby znaleźć odpowiednią bibliotekę i dopisać dyrektywę *#include*. W Javie odpowiednikiem jest instrukcja *import*. Lecz nie musimy jej wpisywać sami. Android Studio może zrobić to za nas. Wystarczy najechać kursorem na podkreśloną linijkę i nacisnąć skrót *Alt+Enter*. Następnie szukamy funkcji ***onCreate***. Wywołuje się ona przy uruchamianiu aplikacji. Na jej końcu dopiszemy obsługę suwaków. Najpierw do stworzonych zmiennych musimy przypisać nasze suwaki utworzone w edytorze graficznym.

Dopiszemy więc linijkę kodu:

```
red = (SeekBar) findViewById(R.id.red);
```

Funkcja *findViewById* zwraca obiekt typu *View*, po którym dziedziczą obiekty wyświetlane na ekranie. Aby mieć dostęp do funkcji obsługujących szczególne właściwości danego elementu, trzeba wykonać rzutowane. Analogicznie postępujemy dla suwaka *green* i *blue*. Dla przycisku musimy napisać:

```
color = (Button) findViewById(R.id.color);
```

W Android Studio bardzo dobrze działa podpowiadanie. Wystarczy, że zaczniemy pisać, a od razu pojawia się lista możliwości. Możemy wybrać interesującą nas opcję strzałkami i kliknąć *enter* lub pisać dalej. Następnie musimy stworzyć obiekt tak zwanego ***Listenera***. Jest to obiekt, którego funkcje są wywoływane przy określonym zdarzeniu. Nadal wewnątrz funkcji **onCreate** piszemy (**listing1**):

Jest to podejście niejako odwrotne niż normalnie. Zwykle korzystamy z napisanej przez kogoś innego funkcji znajdującej się w jakiejś bibliotece. Tym razem cudzy kod obsługi przycisku korzysta z funkcji, którą my dopiero napiszemy. Musimy więc stworzyć funkcję, która przyjmuje i zwraca odpowiednie parametry. Nie będziemy jej nigdzie wywoływać. Musimy tylko „zgłosić", że stworzyliśmy funkcję, która ma być wywoływana, gdy użytkownik przesunie suwaczek. Analogiczny kod musimy stworzyć dla dwóch pozostałych suwaków.

Ponieważ chcemy mieć możliwość płynnej zmiany koloru, interesować nas będzie funkcja *onProgressChanged*, która jest wywoływana przy każdym przesunięciu suwaka. Choć nie używamy dwóch pozostałych funkcji, musimy je jednak zdefiniować.

Dla wszystkich trzech suwaków będziemy postępować tak samo. W funkcji *onProgressChanged* każdego z nich wywołamy funkcję *setLed()*, którą zaimplementujemy jako funkcję składową naszej klasy (czyli wewnątrz klasy, ale poza jakąkolwiek inną funkcją) – **listing 2**:

Najpierw są odczytywane wartości, na które są ustawione kolejne suwaki, a następnie liczby są konwertowane na ciągi znaków dopełnione zerami tak, aby ciąg zawsze miał długość trzech znaków. Dalej następuje najważniejsza linijka: wywołanie funkcji *sendMessage*, która wysyła dane do urządzenia Bluetooth. Ostatnia linijka zmienia kolor przycisku. W Androidzie kolor jest zapisany na 4 bajtach: najstarszy to przezroczystość, a trzy kolejne to czerwony, zielony i niebieski. Ostatnia linijka służy do debugowania. Przyjmuje ona dwa parametry: *tag* i *text*. *Tag* identyfikuje naszą aplikację, a *text* może zawierać dowolny komunikat, który ma nam pomóc przy debugowaniu programu. Oba są typu *string*. Funkcja ta wysyła je do komputera. Są one wyświetlane w czasie działania programu w zakładce *LogCat*. Opcja ta bardzo ułatwia uruchamianie.

Gotowe?

Gratuluję!

Właśnie przygotowałeś pierwszą aplikację dla telefonu. Teraz spróbujemy ją uruchomić. W tym celu podłączamy do komputera telefon i naciskamy znany nam już przycisk **Run** (zielona strzałka). Wybieramy nasze urządzenie z listy i klikamy *OK*. Na telefonie uruchomiła się nasza aplikacja. Włączamy płytkę testową z modułem BTM-222. Z menu naszej aplikacji wybieramy opcję **Connect a device** i z listy urządzeń wybieramy **Serial Adaptor**. Po chwili w górnym oknie aplikacji powinien się pojawić napis **connected**. Teraz za pomocą suwaków możemy zmieniać kolor diody LED! Trzeba uważać, aby nie zmienić orientacji ekranu. Niestety powoduje to rozłączenie z układem. Musimy wtedy jeszcze raz wejść do menu i ponownie się połączyć.

Popatrzmy jeszcze na chwilę na komputer. W Android Studio pojawiły się nowe zakładki. Możemy je zobaczyć na **rysunku 18**. Gdy przesuwamy suwak w aplikacji na telefonie, pojawiają się wysyłane funkcją *Log.e()* komunikaty. Gdy skończymy testowanie, po prostu wychodzimy z aplikacji na telefonie.

Jeżeli pojawią się problemy podczas opisanego właśnie tworzenia własnego programu, można porównać swój kod z projektem zapisanym w folderze ***LED***.

Odbieranie danych.

Teraz rozbudujemy nasz poprzedni program o możliwość odbierania danych. Możemy kontynuować na naszym poprzednim własnym programie lub rozpocząć od projektu **LED**.

Aby odebrać dane z naszego modułu – płytki, najpierw przygotujemy miejsce, w którym dane będą mogły zostać wyświetlone. W tym celu znów edytujemy plik **/res/layout/main.xml**. Tym razem przeciągamy etykietę tekstową *Large Text* i rozciągamy ją w prawo, aż pojawi się napis *match parent* – wtedy zajmie całą szerokość ekranu. W jej ustawieniach wybieramy wartość parametru **gravity** jako *center*. Następnie niżej umieszczamy *progressBar (Horizontal)* i także go rozciągamy. Zmieniamy jego **id** na *voltageBar* oraz **max** (wartość, dla której pasek jest „pełny") na *255*. Jego

Listing 2

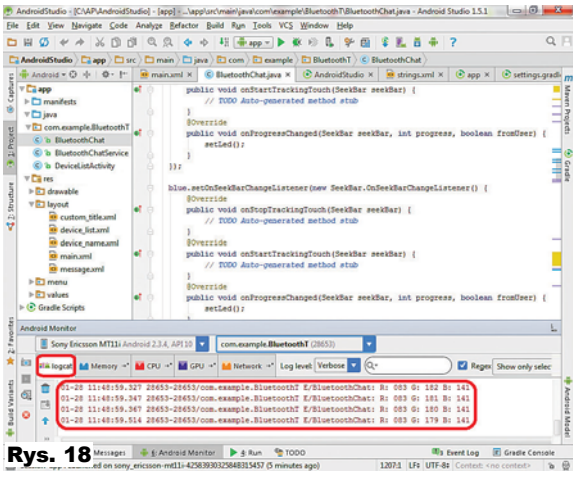
```
private void setLed(){
   int redVal = red.getProgress();
   String redStr = String.format(„%03d", redVal);
   int greenVal = green.getProgress();
   String greenStr = String.format(„%03d", greenVal);
   int blueVal = blue.getProgress();
   String blueStr = String.format(„%03d", blueVal);

   sendMessage(„(l"+redStr+greenStr+blueStr+")");
   color.setBackgroundColor((255<<24)+(redVal<<16)+(greenVal<<8)+blueVal);
   Log.e(TAG, „R: „ + redStr + „ G: „+ greenStr + „ B: „ + blueStr);
}
```

Listing 1

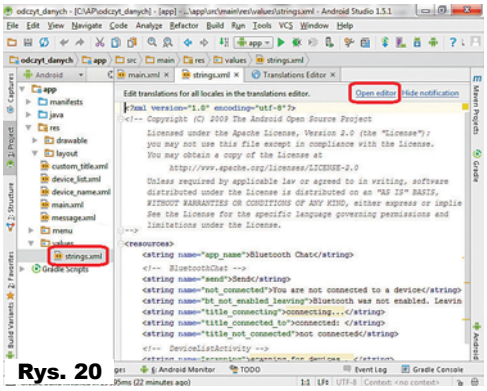
```
red.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {
  @Override
  public void onStopTrackingTouch(SeekBar seekBar) {
    // TODO Auto-generated method stub
    }
  @Override
  public void onStartTrackingTouch(SeekBar seekBar) {
    // TODO Auto-generated method stub
  }
  @Override
  public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    // TODO Auto-generated method stub
  }
});
```

Rys. 19

wypełnienie będzie proporcjonalne do zmierzonego na potencjometrze R5 napięcia. Następnie na liście elementów przechodzimy do zakładki *Layouts* i przeciągamy element *LinearLayout (Horizontal)*. Jest to element, który ułatwia rozmieszczanie innych widgetów. Przekonamy się, jak on działa, przeciągając do niego (puszczamy, gdy zaświeci się na pomarańczowo) najpierw pole tekstowe *Plain Text* (z zakładki *TextFields*), a później przycisk *Buttton*. Przycisk puszczamy „na" pole tekstowe, gdy zaświeci się wokół niego pomarańczowa obwódka. Zmieniamy ich **id** na *voltageText* i *voltageButton*. W polu tekstowym będzie wyświetlana wartość napięcia, a przycisk będzie wysyłał do układu prośbę o przesłanie zmierzonej wartości. Poniżej umieszczamy jeszcze jedną etykietę. Jej **id** ustawiamy jako *tempView*. W niej wyświetlimy aktualną temperaturę. Na samym dole umieszczamy *CheckBox* i ustawiamy jego **id** na *tempCheckBox*. Pozwoli on wybrać, czy chcemy, żeby układ przesyłał nam kolejne wartości temperatury. Rozmieszczenie elementów przedstawia **rysunek 19**. Teraz musimy dodać jeszcze opisy. W aplikacjach dla systemu Android zaleca się, aby wszystkie napisy nie znajdowały się bezpośrednio w kodzie źródłowym, ale były przechowywane w osobnym pliku **/res/values/strings.xml**. Otwórzmy go, aby dodać napisy, potrzebne w aplikacji. Pojawi się okno jak na **rysunku 20**. Przy edycji napisów mamy dwie opcje do wyboru. Możemy edytować kod **XML** lub użyć kreatora. Przełączanie odbywa się za pomocą zaznaczonego na rysunku przycisku *Open editor*. W kursie edytujemy plik XML. Do pliku (pomiędzy tagi *<resources></resources>*) dopisujemy (**listing 3**): Spowoduje to stworzenie czterech dodatkowych stałych tekstowych, które wykorzystamy w dalszej części programu. Następnie zapisujemy pracę i wracamy do edycji widoku. Teraz możemy ustawić napisy, wybierając etykietę tekstową. Klikamy pierwszą etykietę. W zakładce *Properties* znajdujemy pole **text** i klikamy przycisk z trzema kropkami jak na **rysunku 21**. Pojawi się okno z **rysunku 22**. Mamy w nim listę zdefiniowanych stałych tekstowych. Gdy na którąś klikniemy, na dole pojawi się jej wartość. Wybieramy *voltage* i klikamy OK. Analogicznie dodajemy podpisy pozostałym elementom, aby otrzymać taki wynik jak na rysunku 19.


Rys. 18

```
<string name="voltage" >Napięcie</string>
<string name="measure">Zmierz</string>
<string name="temperature">Temperatura: %.2f</string>
<string name="get_temperature">Przesyłanie temperatury</string>
```
listing 3


Rys. 20

Tak jak poprzednio, dopiszemy logikę aplikacji w pliku *BluetoothChat.java*. Zaczynamy od obsłużenia wysyłania komend sterujących. Kody umieszczamy w funkcji *onCreate*. Obsługa przycisku wygląda następująco (**listing 4**):

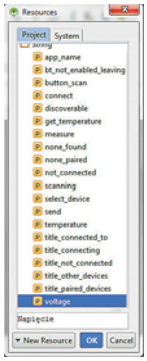
Najpierw znajdujemy nasz przycisk utworzony w edytorze graficznym. Następnie tworzymy obiekt klasy ***listener***, którego funkcja *onClick* będzie wywoływana za każdym razem, gdy użytkownik naciśnie przycisk.

Kod obsługi przycisku wyboru (*check box*) jest analogiczny (**listing 5**):

Kilka razy będziemy znowu musieli użyć skrótu *Alt+Enter* dodającego nagłówki. Przy importowaniu *ProgressBar* będziemy mieli kilka możliwości do wyboru. Wygląd listy będzie podobny jak na **rysunku 23**. Wybieramy opcję zaznaczoną czerwoną obwódką. Początkowo środowisko podkreśli nam na czerwono także adnotacje *@Override*. Jeżeli najedziemy na nie myszką i naciśniemy Alt+Enter, także pojawi się lista możliwych rozwiązań tego problemu, lecz ignorujemy je. Gdy zaimportujemy brakujące klasy, podkreślenia te znikną.

```
Button getVoltage = (Button) findViewById(R.id.voltageButton);
getVoltage.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
   sendMessage(„(a)");
   }
});
```
Listing 4


Rys. 21


Rys. 22

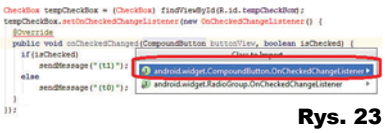
Tym razem stosujemy ***listener*** reagujący na zmianę stanu przycisku. Przy każdej akcji wykonywana jest funkcja *onCheckedChanged*. Dzięki jej parametrowi *isChacked* dowiadujemy się, jaki jest obecny stan przełącznika.

Teraz pozostaje wyświetlić odebrane dane w polu tekstowym oraz na pasku. Od razu zajmiemy się także etykietą wyświetlającą aktualną temperaturę. W tym celu najpierw tworzymy trzy zmienne prywatne w klasie będącej oknem naszej aplikacji, ale poza funkcjami:

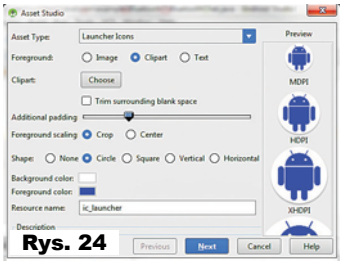
```
private ProgressBar voltageBar;
private EditText voltageText;
private TextView tempView;
```

Następnie, tak jak to robiliśmy wcześniej, musimy powiązać elementy GUI z naszymi zmiennymi. W tym celu w funkcji *onCreate* dodajemy linie (**listing 6**):

```
CheckBox tempCheckBox = (CheckBox) findViewById(R.id.tempCheckBox);
tempCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
  @Override
  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    if(isChecked)
      sendMessage(„(t1)");
    else
      sendMessage(„(t0)");
  }
});
```
Listing 5



Rys. 23



Rys. 24

Teraz zajmiemy się najważniejszą rzeczą: odbieraniem danych wysłanych przez urządzenie. Po odebraniu wiadomości jest wywoływana funkcja *handleMessage*, a dokładniej element instrukcji *switch case* dla warunku *MESSAGE_READ*. Aby łatwiej było go odnaleźć, jest przy nim umieszczony komentarz o treści */*TODO Odebrano nową wiadomość*/*.

Jest to bardzo interesująca opcja edytora. Miejsca te są także zaznaczone niebieskimi prostokątami na pasku przewijania. Jeśli klikniemy taki prostokąt, zostaniemy przeniesieni do miejsca z komentarzem TODO. Kod umieszczony poniżej tego komentarza zostanie wykonany za każdym razem, gdy zostanie odebrana nowa wiadomość. Jej treść znajduje się w ciągu znaków *readMessage*. Ponieważ wiadomości wysyłane przez moduł Bluetooth dochodzą do telefonu we fragmentach, trzeba połączyć je w jedną całość. W tym celu tworzymy zmienną *temp* wklejając poniższą linijkę wewnątrz naszej klasy, ale poza jakąkolwiek funkcją:

```
String temp = new String();
```

Zmienna ta posłuży do łączenia kolejnych elementów wiadomości.

Nasz moduł wysyła nam dwa różne komunikaty: napięcie odczytane z przetwornika ADC oraz temperaturę. Każda z tych wiadomości ma na początku literę (odpowiednio **a** lub **t**) oraz liczbę. Kod, który interpretuje i wyświetla otrzymane dane, wklejamy poniżej komentarza */*TODO Odebrano nową wiadomość*/*. Prezentuje się on następująco (**listing 7**): Jest on trochę rozbudowany, ale nie ma w nim nic trudnego. Najpierw następuje sprawdzenie, czy pierwszy znak wiadomości to ‚**a**' lub ‚**t**'. Jeśli tak jest, odebrano nową wiadomość. Stara zawartość zmiennej *temp* jest więc kasowana i zastępowana w całości obecnie odebraną. W przeciwnym razie następuje dodane odebranej wiadomości na koniec zmiennej *temp*.

Następnie instrukcja *switch*, na podstawie pierwszego znaku zmiennej *temp* decyduje, którą instrukcję otrzymała. Sprawdzana jest także długość ciągu. W celu lepszej eliminacji błędów można testować, czy odpowiednie bajty danych są liczbami. Powyższy kod nie realizuje tej funkcji. Ponieważ wysyłane komendy mają zawsze taką samą długość, zamienienie ciągu znaków ASCII na liczbę jest bardzo proste. Na końcu odebrane dane są wyświetlane.

Zmiana nazwy aplikacji.

Aby zmienić wyświetlaną nazwę aplikacji, w pliku **/res/values/strings.xml** znajdujemy pozycję *app_name* i zmieniamy jej wartość. Zmiana ikony także nie jest trudna. W tym celu wybieramy w panelu *Project*, zaznaczamy folder **app** i z menu wybieramy *file->New->Image Asset*. Pojawia się okno (**rysunek 24**), w który możemy stworzyć naszą ikonę. Mamy możliwość wgrać własny obrazek, wybrać jeden z gotowych clipartów lub wstawić tekst. Dalsze pola formularza pozwalają wybrać jeden z kilku możliwych kształtów oraz zmienić kolor. Po prawej stronie okna widzimy podgląd naszego dzieła. Gdy będziemy zadowoleni z efektu klikamy *Next* i potem *Finish*.

Co dalej?

Celem artykułu było pokazanie, jak szybko utworzyć prostą aplikację dla smartfona, która pozwoli wysyłać i odbierać dane z mikrokontrolera. Autor chętnie udzieli dodatkowych wyjaśnień i na życzenie rozszerzy temat. A jeśli ktoś chciałby dowiedzieć się więcej na temat Androida, może sięgnąć do jednej z wielu książek na ten temat. Mnóstwo informacji znajduje się na stronie *developer.android.com*.

Wszystkie ilustracje z artykułu dostępne są w Elportalu.

Rafał Kozik
rafkozik@gmail.com

Od redakcji. Bardzo prosimy osoby, które przedstawiony artykuł zachęci do pierwszych działań w zakresie wykorzystania łącza Bluetooth i stworzenia pierwszego programu, by na gorąco podzieliły się swoimi wrażeniami, odczuciami i doświadczeniami!
Zwłaszcza wtedy, gdy napotkanie problem lub coś będzie niejasne, piszcie do Autora lub na adres *edw@elportal.pl*.

Wykaz elementów

R1-R3	510
R4	4,7k
R5	potencjometr 100k
C1, C4, C5	100nF
C2, C3	100uF
LED1	Led RGB wspólna anoda
U1	LD1117V33
U2	ATmega48/88/168
U3	BTM-222
U4	DS18B20
A1	antena BT (AN9520)
JP1	goldpiny wtyk 5*2
JP2	goldpiny wtyk 4*2
JP3, JP4	goldpiny wtyk 2*1
JP5	goldpiny wtyk 3*2
Arduino (goldpin gniazdo 6*1, 6*1, 8*1, 8*1)	
Jumperki na goldpiny 11 sztuk	

Komplet podzespołów z płytką jest dostępny w sieci handlowej AVT jako kit szkolny AVT-3146.

```
voltageBar = (ProgressBar) findViewById(R.id.voltageBar);
voltageText = (EditText) findViewById(R.id.voltageText);
tempView = (TextView) findViewById(R.id.tempView);
```

Listing 6

```
if(readMessage.charAt(0) == ,a' || readMessage.charAt(0) == ,t')
   temp = readMessage;
  else
   temp = temp + readMessage;
  Log.e(TAG, temp);
  switch(temp.charAt(0)){
    case ,a':
      if(temp.length() > 4){
          int voltage = 0;
          for(int i = 1; i < 4; i++)
            voltage = 10*voltage+temp.charAt(i)-'0';
          voltageBar.setProgress(voltage);
          voltageText.setText(„"+voltage);
      }
      break;
    case ,t':
      if(temp.length() > 7 && temp.charAt(4) == ,.'){
          float temperature = 0;
          for(int i = 1; i < 4; i++)
            temperature = temperature*10+temp.charAt(i)-'0';
          for(int i = 5; i < 7; i++)
            temperature = temperature*10+(temp.charAt(i)-'0');
          temperature = temperature / 100;
          tempView.setText(String.format(getResources()
          .getString(R.string.temperature), temperature));
      }
      break;
    default:
      break;
                }
```

Listing 7