Bluetooth Relay z Androidem

AVT-5295 to zestaw przygotowany przez Piotra Rosenbauma i opisany w numerze Elektroniki Praktycznej 06/2011. Choć zestaw nie jest najnowszy, cieszy się niemałym powodzeniem, szczególnie w dobie rosnącej popularności komunikacji bezprzewodowej. Pozwala w niedrogi sposób zrealizować zdalne sterowanie sześciu niezależnie włączanymi urządzeniami, a dodatkowo umożliwia monitorowanie temperatury. Niestety, od czasu powstania tego projektu, sytuacja w świecie elektroniki zmieniła się na tyle, że kontrola takiego urządzenia z komputera PC wydaje się mało atrakcyjnym rozwiązaniem. W dobie urządzeń mobilnych, sterowanie przekaźnikami przez Bluetooth z telefonu byłoby znacznie bardziej użyteczne. Dlatego prezentujemy nowe oprogramowanie do zestawu AVT-5295, przygotowane pod system Android.

Oryginalny projekt Piotra Rosenbauma zawierał skompilowane oprogramowanie pod system MS Windows. Opis dostarczony przez autora koncentrował się na części sprzętowej, a szczegóły dotyczące sposobu wymiany danych poprzez interfejs Bluetooth były niekompletne. Jednakże zastosowanie oprogramowanie do monitorowania komunikacji przez interfejs szeregowy umożliwiło pełny reverse-engineering zastosowanego, prostego protokołu komunikacyjnego i odtworzenie przesyłanych komunikatów w nowym programie.

Aplikacja pod system Android nie została jedna stworzona w Javie, czyli w natywnym języku, z którego korzysta system operacyjny, ale z użyciem platformy Apache Cordova. Platforma ta stanowi bezpłatny zbiór bibliotek, tłumaczących polecenia javascriptowe na polecenia języków stosowanych natywnie w mobilnych systemach operacyjnych – w naszym przypadku – w Javy. Cała platforma Cordova obejmuje też kompilatory i szereg pluginów. Korzystania z Cordovy można nauczyć się dzięki publikowanemu w Elektronice Praktycznej od lutego 2015 roku kursowi programowania aplikacji mobilnych. Niemal cała wiedza potrzebna do realizacji niniejszej aplikacji zawarta jest w pierwszych pięciu częściach kursu, przy czym w opisywanym programie, na potrzeby uzyskania odpowiedniego wyglądu aplikacji, skorzystano z języków HTML i CSS w stopniu wykraczającym poza zakres informacji ze wspomnianych części kursu.

Warto dodać, że udostępniamy czytelnikom nie tylko gotową, skompilowaną aplikację androidową, ale też jej pełny kod źródłowy, dzięki czemu każdy będzie mógł samodzielnie zmodyfikować wygląd i funkcje aplikacji, skompilować ją, a być może nawet przenieść na inny system operacyjny.

Opis protokołu

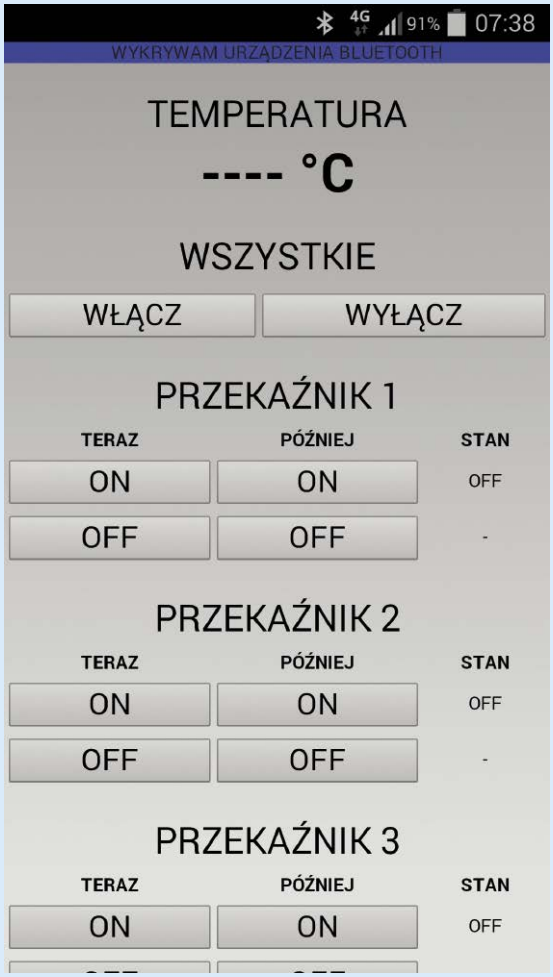
Protokół komunikacyjny, zastosowany w zestawie AVT-5295 jest bardzo prosty i polega na przesyłaniu pojedynczych znaków lub ciągów znaków przez interfejs Bluetooth. Na początek urządzenie mobilne powinno samo nawiązać połączenie z wykrytym urządzeniem Bluetooth o nazwie „Bluetooth Adaptor”. Następnie zestaw z przekaźnikami zaczyna cyklicznie, co 1 sekundę, przesyłać ciąg znaków, zawierający aktualną temperaturę, odczytywaną z wbudowanego sensora. Każdy taki komunikat zakończony jest znakiem końca linii, takim jaki stosuje się w systemie Windows, czyli bajtami 0x0D i 0x0A, określanymi też mianem znaków CR i LF. W systemie Windows te dwa następujące po sobie bajty są traktowane jako jeden znak końca linii, choć w praktyce są to następujące po sobie znaki „\r” i „\n”. Gdyby chcieć przenosić aplikację na inny system, należy wziąć pod uwagę fakt, że w Linuksie jako znak końca linii używany jest tylko 0x0A, a w systemach MacOS – tylko 0x0D. Ale mniejsza z tym.

Tabela 1. Zestaw komend (znaków) przesyłanych przez aplikację do modułu z przekaźnikami

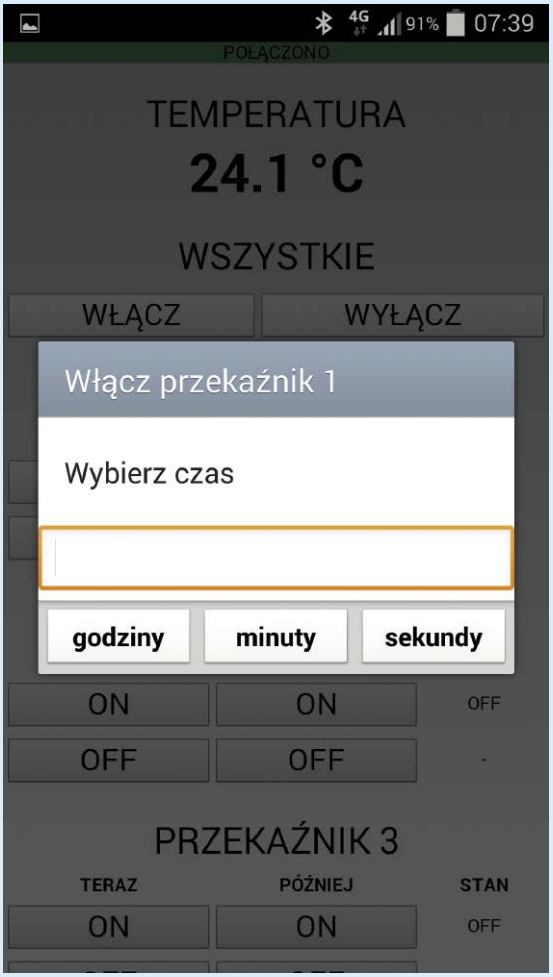
Znak	Przekaźnik	Operacja
a	1	włącz
b		wyłącz
c	2	włącz
d		wyłącz
e	3	włącz
f		wyłącz
g	4	włącz
h		wyłącz
i	5	włącz
j		wyłącz
k	6	włącz
l		wyłącz
y	1-6	włącz
z		wyłącz

W naszym przypadku informacja o temperaturze zajmuje najczęściej 6 bajtów, na które składają się następujące znaki: dwie cyfry przed przecinkiem, znak przecinka (autor użył w tym celu kropki), jedna cyfra po przecinku i wspomniane dwa znaki końca linii (CR i LF). To jedyne dane, jakie moduł Bluetooth Relay przesyła do aplikacji.

Aplikacja natomiast steruje pracą modułu poprzez wysyłanie pojedynczych liter alfabetu. Moduł obsługuje 14 różnych poleceń, z których każde reprezentowane jest przez inny znak. Małe litery od „a” do „l” służą do włączania i wyłączania kolejnych przekaźników, a znaki „y” i „z” do włączania i wyłączania (odpowiednio) wszystkich przekaźników na raz. Pełna lista komend przesyłanych do modułu została zawarta w **tabeli 1**. Wszystkie operacje związane z opóźnionym włączaniem lub wyłączaniem przekaźników oraz z kontrolą ich stanu są wykonywane w aplikacji. Aplikacja nie ma podglądu na rzeczywisty stan przekaźników ani nie może wysłać



Rysunek 1. Główne okno aplikacji przed dokonaniem połączenia z modułem z przekaźnikami



Rysunek 2. Okno dialogowe z pytaniem o czas zwłoki

polecenia zmiany stanu przekaźnika za jakiś czas, np. po jej wyłączeniu. Prezentowany w aplikacji stan przekaźników jest kalkulowany w oparciu o polecenia wydawane przez użytkownika.

Konstrukcja aplikacji

Przygotowana aplikacja wykorzystuje oprogramowanie Apache Cordova i zawartą w niej platformę androidową. Ponadto użyto dwóch pluginów:

- **com.megster.cordova.bluetoothserial**, potrzebnego do realizacji komunikacji przez interfejs Bluetooth,
- **org.apache.cordova.dialogs**, ułatwiającego prezentowanie okien dialogowych.

Program opiera się o standardową, czystą aplikację Cordovy, w której zmieniono jedynie treść plików index.js, index.html i index.css. Ponadto skorzystano z biblioteki jQuery, którą dodano do zasobów w katalogu z plikami Javascripta oraz wprowadzono ikonkę, by zastąpić domyślną ikonę aplikacji Cordovy.

Działanie aplikacji rozpoczyna się od przypisania poleceń do wszystkich przycisków oraz inicjalizacji interfejsu Bluetooth. Po nawiązaniu połączenia z modułem z przekaźnikami, uruchamiane jest nasłuchiwanie na ciągi znaków, przesyłane interfejsem Bluetooth i zakończone znakami końca linii (CR i LF). Pozyskiwane ciągi znaków są prezentowane jako wskazanie temperatury, w górnej części aplikacji.

W przypadku naciśnięcia któregokolwiek z przycisków, aplikacja wysyła adekwatną komendę do modułu z przekaźnikami lub wyświetla okno dialogowe z prośbą o wskazanie opóźnienia, z jakim ma być wysłana dana komenda. Ponieważ logika programu jest napisana z użyciem języka Javascript, bardzo łatwo było zrealizować asynchroniczną pracę aplikacji. W efekcie, nic nie stoi na przeszkodzie, by zażądać wykonania kilku opóźnionych operacji dla tego samego przekaźnika, wywoływanych w dowolnym czasie. Każda kolejna zlecona, opóźniona operacja nie wstrzymuje wykonywania poprzedniej, choć wyświetlany timer prezentuje odliczanie w dosyć specyficzny sposób: z upływem każdej jednostki czasu (godziny, minuty, sekundy), w jakiej podane było opóźnienie wykonania danej komendy, zmieniane jest wskazanie timera dla danego przekaźnika. Jeśli przykładowo, dla przekaźnika pierwszego, prowadzone są jednocześnie trzy odliczania, liczby prezentujące pozostały czas dla poszczególnych timerów będą prezentowane naprzemiennie.

Oprócz tej różnicy z harmonogramowaniem opóźnionych komend dla przekaźników, działanie aplikacji mobilnej jest identyczne z działaniem oryginalnego programu dla systemu operacyjnego MS Windows. Naturalnie istotnie różny jest graficzny interfejs użytkownika.

Szczegóły implementacji HTML

Na samym początku sekcji BODY znajduje się warstwa „connection", która informuje o aktualnym stanie połączenia Bluetooth. Poniżej znalazło się pole w które wpisywana jest wartość temperatury podana przez moduł z przekaźnikami (pole „temp"). Następnie umieszczono

Listing 1. Fragmenty kodu HTML aplikacji

```
<!-- Fragment nagłówka -->
<DIV id="connection">ROZŁĄCZONO</DIV>
<DIV>
        <H1>Temperatura</H1>
        <DIV id="temp-wrapper">
                <SPAN id="temp">----</SPAN> °C</DIV>
        <H1>Wszystkie</H1>
        <TABLE><TR>
                <TD><BUTTON id="all-on">WŁĄCZ</BUTTON></TD>
                <TD><BUTTON id="all-off">WYŁĄCZ</BUTTON></TD>
        </TR></TABLE>
</DIV>

<!-- Fragment powtarzający się dla kolejnych przekaźników -->
<H1>Przekaźnik 1</H1>
<TABLE class="relay">
        <TR><TH>TERAZ</TH><TH>PÓŹNIEJ</TH><TH>STAN</TH></TR>
        <TR>
                <TD><BUTTON id="now-on-1">ON</BUTTON></TD>
                <TD><BUTTON id="later-on-1" class="later on">ON</BUTTON></TD>
                <TD><SPAN id="state-1" class="state">OFF</SPAN></TD>
        </TR>
        <TR>
                <TD><BUTTON id="now-off-1">OFF</BUTTON></TD>
                <TD><BUTTON id="later-off-1" class="later off">OFF</BUTTON></TD>
                <TD><SPAN id="time-1">-</SPAN></TD>
        </TR>
</TABLE>
```

dwa duże przyciski „Włącz" i „Wyłącz", które sterują wszystkimi przekaźnikami jednocześnie.

W dalszej części umieszczono powtarzające się tabele klasy „relay" – po jednej dla każdego przekaźnika. Ich pola (przyciski i spany) różnią się tylko ostatnią cyfrą, która pozwala rozpoznać, którego przekaźnika dotyczą. Każdy z przekaźników obsługiwany jest przez cztery przyciski: dwa do natychmiastowego włączenia i wyłączenia („now-on-X" i „now-off-X", gdzie X to numer przekaźnika) oraz dwa do opóźnionego włączenia i wyłączenia („later-on-X" i „later-off-X"). Pole „state-X" informuje o aktualnym stanie danego przekaźnika, a pole „time-X" wskazuje czas pozostały w timerach powiązanych z danym przekaźnikiem. Dzięki zastosowaniu wtyczki **org.apache.cordova.dialogs**, nie było konieczne samodzielne tworzenie okien dialogowych, potrzebnych do wpisywania czasu opóźnienia wysłania komend. Fragment pliku index.html został pokazany na **listingu 1**.

Szczegóły implementacji CSS

Plik CSS bazuje na standardowym pliku stylu do świeżej aplikacji Cordovy, pozbawionym kilku niepotrzebnych elementów. Zmieniono m.in. parametry nagłówka H1 oraz wykorzystano klasę .blink wraz ze zmodyfikowanymi parametrami aplikacji, by sygnalizować że trwa nawiązywanie połączenia Bluetooth. Elementom tabeli i przyciskom (TABLE, TD, BUTTON) przypisano odpowiednie rozmiary i wielkości czcionek, tak by nie pozostawiały pustego miejsca na ekranie. Dodano też klasy:

- .relay-on – by kolorem oznaczyć przekaźniki włączone,
- .timed-on – by kolorem zielonym oznaczać odliczanie timera powodującego włączenie przekaźnika,
- .timed-on – by kolorem czerwonym oznaczać odliczanie timera powodującego wyłączenie przekaźnika.

Fragment pliku index.css znalazł się na **listingu 2**.

Szczegóły implementacji Javascript

Naturalnie najbardziej złożony jest kod Javascriptu. W funkcji **app.onDeviceReady()** umieszczono wywołania dwóch funkcji:

- **app.bt1()**, która inicjalizuje połączenie Bluetooth z modułem z przekaźnikami oraz prowadzi nasłuch danych dotyczących temperatury,
- **app.assign_buttons()**, która powoduje przypisanie akcji do poszczególnych przycisków.

Oprócz tego stworzono funkcję **app.countdown()**, która służy do prezentacji czasu pozostałego do zakończenia odliczania czasu timerami.

Funkcja **app.bt1()** została pokazana na **listingu 3**. Rozpoczyna się od zmiany wyglądu paska informującego o stanie interfejsu Bluetooth. Następnie aplikacja

Listing 2. Fragment kodu CSS aplikacji

```
h1 {
    font-size:24px;
    font-weight:normal;
    margin: 20px 0 5px;
    overflow:visible;
    padding:0px;
    text-align:center;
}

@keyframes fade {
    from { opacity: 1.0; }
    50% { opacity: 0.4; color:#FFF; }
    to { opacity: 1.0; }
}

@-webkit-keyframes fade {
    from { opacity: 1.0; }
    50% { opacity: 0.4; color:#FFF; }
    to { opacity: 1.0; }
}

.blink {
    animation:fade 3000ms infinite;
    -webkit-animation:fade 3000ms infinite;
}

TABLE{
        width:100%;
}
BUTTON{
        width:100%;
        font-size:20px;
}
TD{
        text-align:center;
}
#temp-wrapper{
        text-align: center;
        font-size: 35px;
        font-weight: bold;
}
#connection{
        text-align:center;
        background-color:red;
}
.relay-on{
        background-color:lightgreen;
}
.timed-on{
        color:green;
}
.timed-off{
        color:red;
}
```

Listing 3. Funkcja app.bt1()

```
bt1: function() {
        $(„#connection”).html(„WYKRYWAM URZĄDZENIA BLUETOOTH”).css(„background-color”,”blue”).
addClass(,blink’);

//próba znalezienia urządzenia Bluetooth
        bluetoothSerial.discoverUnpaired(function(devices) {
                $(„#connection”).html(„TRWA ŁĄCZENIE”).css(„background-color”,”blue”);
                devices.forEach(function(device) {
                        if (device.name == „Serial Adaptor”) {
                                app.btaddr = device.address;

//próba połączenia się z urządzeniem BT o pożądanej nazwie
                                bluetoothSerial.connect(app.btaddr, function() {
                                        $(„#connection”).html(„POŁĄCZONO”).css(„background-
color”,”green”).removeClass(,blink’);

//funkcja nasłuchująca danych o temperaturze                                    bluetoothSerial.
subscribe(,\n’,function(data){
                                                $(„#temp”).html(data);
                                        }, function(){alert(„niepowodzenie subskrypcji”);});
                                },
                                function() {
                                        alert(„zakończono połączenie”);
                                });
                        }
                })
        }, function() {
                $(„#connection”).html(„POŁĄCZENIE NIEUDANE”).css(„background-color”,”red”).
removeClass(,blink’);
        });
},
```

Listing 4. Fragment funkcji app.assign_buttons() obejmujący przypisanie akcji do przycisków natychmiastowo wywołujących komendy

```
//fragment niepowtarzający się
$(„#all-on”).click(function() {
        bluetoothSerial.write(,y’);
        $(„.state”).html(„ON”).addClass(,relay-on’);
});
$(„#all-off”).click(function() {
        bluetoothSerial.write(,z’);
        $(„.state”).html(„OFF”).removeClass(,relay-on’);
});

//fragment powtarzający się dla każdego z przekaźników
$(„#now-on-1”).click(function() {
        bluetoothSerial.write(,a’);
        $(„#state-1”).html(„ON”).addClass(,relay-on’);
});
$(„#now-off-1”).click(function() {
        bluetoothSerial.write(,b’);
        $(„#state-1”).html(„OFF”).removeClass(,relay-on’);
});
```

wykrywa urządzenia Bluetooth w otoczeniu i gdy skończy, zmienia pasek informujący o stanie interfejsu oraz zaczyna, dla każdego ze znalezionych urządzeń, sprawdzać czy nazwa urządzenia to „Serial Adaptor”. Jeśli tak się stanie, aplikacja nawiązuje z danym urządzeniem połączenie, zmienia stan pasku informującego o interfejsie BT i zaczyna nasłuchiwać, korzystając z polecenia **bluetoothSerial.subscribe()** na ciągi znaków zakończone znakiem końca linii. Po każdym odebranym pakiecie takich danych, treść pola „temp” w aplikacji podmieniana jest na nowe wskazanie temperatury.

Listing 5. Fragment kodu funkcji app.assign_buttons() obejmujący przypisanie akcji do przycisków wywołujących komendy ze zwłoką

```
$(„.later”).click(function() {
//pobranie identyfikatora
        var identyfikator=$(this).attr(„id”);
//wywołanie okna dialogowego
        navigator.notification.prompt(
                „Wybierz czas”,
                function(result){
                        if (result.buttonIndex)
                        {
                                var time=parseInt(result.input1);
                                if (time){
                                        var multi=0;
                                        var jednostka=””;
                                        switch (result.buttonIndex)
                                        {
                                                case 1:
                                                        multi=60*60;
                                                        jednostka=’h’;
                                                        break;
                                                case 2:
                                                        multi=60;
                                                        jednostka=’m’;
                                                        break;
                                                case 3:
                                                        multi=1;
                                                        jednostka=’s’;
                                                        break;
                                        }
                                        var seconds=time*multi;
//wywołanie funkcji z opóźnieniem
                                        setTimeout(function(){
                                        $(„#now”+identyfikator.slice(5)).trigger(,click’);
                                        },seconds*1000);
                                        app.countdown(identyfikator.slice(-1),(identyfikator.slice(6,-2) ==
„on” ? 1 : 0),jednostka,time);
                                }
                        }
                },
                (identyfikator.slice(6,-2) == „on” ? „Włącz” : „Wyłącz”) + „ przekaźnik „+identyfikator.
slice(-1),
                [„godziny”, „minuty”, „sekundy”]);
});
```

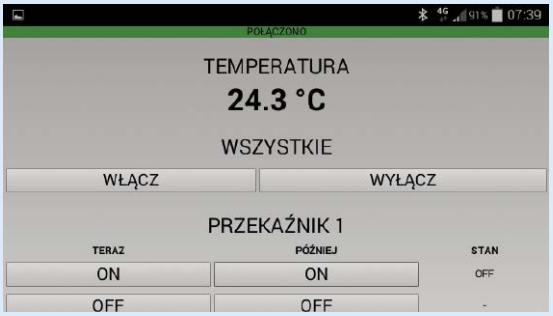
Listing 6. Funkcja app.countdown()

```
countdown:function(przekaznik,stan,jednostka,czas){
        if (czas!=0){
                var nczas=czas-1;
                if (stan=="1")
                        $(„#time-"+przekaznik).html(czas + „ „ +jednostka).addClass(„timed-on").
removeClass(„timed-off");
                else if (stan=="0")
                        $(„#time-"+przekaznik).html(czas + „ „ +jednostka).addClass(„timed-off").
removeClass(„timed-on");
                var multi=0;
                switch (jednostka)
                {
                        case ‚h':
                                multi=60*60;
                                break;
                        case ‚m':
                                multi=60;
                                break;
                        case ‚s':
                                multi=1;
                                break;
                }
//wywołanie samej siebie z opóźnieniem
                setTimeout(function(){
                        app.countdown(przekaznik,stan,jednostka,nczas);
                },multi*1000);
        }else $(„#time-"+przekaznik).html(„-").removeClass(„timed-on").removeClass(„timed-off");
},
```

Funkcja **app.assign_buttons()** składa się w większości z powtarzających się elementów. Dla każdego z przycisków definiowane jest odpowiednie przypisanie akcji następującej po jego naciśnięciu. Jest to przesłanie odpowiedniej komendy przez interfejs Bluetooth oraz zmiana etykietki ze stanem adekwatnych przekaźników. Fragment tego kodu przedstawiono na **listingu 4**. Warto zwrócić uwagę na to, że w przypadku przycisków włączających i wyłączających wszystkie przekaźniki jednocześnie, zmiana sygnalizacji stanu przekaźników jest realizowana w oparciu o klasę .state, a nie o unikalne identyfikatory.

Znacznie ciekawszy jest kod służący do przypisania akcji przyciskom zwłocznym (**listing 5**). Został on tak zoptymalizowany, by zajął jak najmniej miejsca – wszystkim tym przyciskom (klasy .later) przypisywana jest jedna i ta sama funkcja. Wstępnie jednak pobierany jest identyfikator danego przycisku, dla którego funkcja została wywołana. Identyfikator ten pozwala później rozpoznać, którego przekaźnika dotyczy dana operacja.

Dla ułatwienia, wykorzystano polecenie **navigator.notification.prompt()**, które wywołuje okno do wprowadzania czasu opóźnienia wysłania komendy. Na dole okna umieszczono trzy przyciski, odpowiadające trzem jednostkom podawanej wartości opóźnienia: godzinom, minutom i sekundom. Naciśnięcie jednego z tych przycisków powoduje wywołanie kolejnej funkcji, w której obliczany jest odpowiedni mnożnik czasu dla wybranej jednostki: 3600, 60 lub 1. W przypadku gdy użytkownik nie naciśnie żadnego z tych przycisków, tylko zamknie okienko dialogowe, korzystając z systemowego przycisku „cofnij", timer nie zostanie ustawiony. Timer nie zostanie też ustawiony, jeśli wartość wpisana w oknie dialogowym nie jest liczbą całkowitą. Obliczona liczba sekund (a następnie milisekund), jaka musi upłynąć przed wysłaniem komendy jest podawana w funkcji **setTimeout()**. Aby skrócić ilość kodu potrzebnego do obsługi przycisków zwłocznych, z zapamiętanego wcześniej identyfikatora wycinane są fragmenty określające numer przekaźnika, którego dotyczy przycisk oraz rodzaj wywoływanej operacji (włączenie lub wyłączenie). By uniknąć problemu ponownego definiowania komend Bluetooth, odpowiadających poszczególnym operacjom, w funkcji uruchamianej po żądanym czasie zwłoki wyzwalane



Rysunek 3. Okno aplikacji w trakcie jej pracy w orientacji poziomej

jest kliknięcie w przycisk odpowiadający danej operacji i danemu przekaźnikowi; przyciskom tym przypisano bowiem już odpowiednie komendy we wcześniejszym fragmencie kodu funkcji **app.assign_buttons()**.

Na koniec, już poza funkcją **setTimeout()** wywoływana jest funkcja **app.countdown()** z parametrami określającymi numer przekaźnika, rodzaj operacji oraz wybrane jednostkę czasu i czas zwłoki. Funkcja **app.countdown()** jest praktycznie niezależna i służy tylko temu, by w odpowiednich miejscach interfejsu prezentować odliczanie czasu timerów. Funkcja ta została zaprezentowana na **listingu 6**.

Działanie funkcji **app.countdown()** polega na sprawdzaniu, czy pozostały czas nie osiągnął zera i w razie potrzeby na zmniejszeniu licznika czasu o 1, wpisaniu nowej wartości w odpowiednim polu interfejsu użytkownika oraz na ponownym zaplanowaniu wywołania siebie samej, z wykorzystaniem funkcji **setTimeout()**, z opóźnieniem odpowiadającym jednej jednostce czasu, jaką użytkownik wybrał dla danego timera.

Podsumowanie

Gotowa aplikacja zajmuje niecałe 2 MB i uruchamia się praktycznie błyskawicznie. Pozwala zastąpić program w wersji na Windows. Dodatkowo, przenośny kod sprawia, że bardzo łatwo dokonać kompilacji kodu dla innych systemów operacyjnych, takich jak Windows Phone 8 i iOS, z tym że trzeba mieć na uwadze, że niektóre urządzenia mogą mieć problemy z komunikacją z użyciem interfejsu Bluetooth.

Marcin Karbowniczek, EP