Programowanie aplikacji mobilnych (4)

Komunikacja Bluetooth

W dotychczasowych częściach kursu programowania aplikacji mobilnych pokazaliśmy, jak skorzystać z niektórych modułów dostępnych w telefonie lub tablecie oraz jak wysyłać proste zapytania przez Internet. Jednakże sensowne wykorzystanie zasobów sprzętowych wymaga bardziej dynamicznej komunikacji, której samymi żądaniami HTTP, inicjowanymi z systemu mobilnego, nie da się zrealizować. Dlatego teraz zademonstrujemy sposób użycia interfejsu komunikacyjnego Bluetooth.

Będziemy bazować na projekcie sterowania bramą, takim jak wykonaliśmy w trzeciej części kursu, czyli podzielonym na dwie części: pilot zdalnego sterowania i napęd, ale zastępujemy poprzednie funkcje nowymi, więc tworzymy pliki kodu aplikacji od nowa.

Standardowy Bluetooth

Modułu Bluetooth telefonu użyjemy do łączenia się z bramą, tak by móc sterować jej pracą bezprzewodowo, bez Wi-Fi czy sieci komórkowej. Niestety, ze względu na ograniczenia najbardziej popularnego, sprawdzonego i wygodnego pluginu do Cordovy obsługującego Bluetooth, będziemy zmuszeni łączyć się z niezależnym modułem Bluetooth napędu bramy. Używana przez nas wtyczka nie pozwala na komunikację pomiędzy dwoma urządzeniami mobilnymi. Nie zmniejsza to jednak wartości dydaktycznej przykładu – pokażemy jak wysyłać dane oraz jak je odbierać, a także jak w ogóle zarządzać całą komunikacją. Trzeba mieć przy tym na uwadze, że istnieją też duże różnice w implementacji Bluetooth pomiędzy poszczególnymi mobilnymi systemami operacyjnymi i wtyczka działa nieco inaczej na każdym z systemów. Jest kompatybilna z Androidem i Windows Phone, gdzie korzysta z typowego Bluetootha. W systemie iOS działa tylko z urządzeniami z modułami RedBearLab BLE, Adafruit Bluefruit LE, Laird BL600 i BlueGiga, czyli z iPhone 4S, iPhone 5, iPod 5 i iPadami 3 lub nowszymi – umożliwia na nich pracę jedynie w trybie Bluetooth Low Energy. Ponieważ (tak jak i większość czytelników) przykłady kompilujemy pod Androida, to na nim się skoncentrujemy.

Wymieniona wtyczka to **com.megster.cordova.bluetoothserial**, która w naszym przypadku pobrała się w wersji 0.4.2. Instalujemy ją tak jak inne pluginy, w projekcie pilota bramy. Będziemy za jej pomocą zamienić dotychczas używaną komunikację ethernetową na żądania Bluetooth.

Zaczniemy od wyjaśnienia, jak działa Bluetooth z punktu widzenia programisty, bez wnikania w szczegóły warstwy fizycznej.

Bluetooth jest interfejsem szeregowym, a komunikacja pomiędzy urządzeniami odbywa się po formalnym nawiązaniu połączenia. Urządzenia rozpoznają się po identyfikatorach, będących najczęściej numerami MAC. Oprócz nic, urządzenia mogą mieć swoje nazwy, dzięki którym użytkownik może łatwo rozpoznać, z którym sprzętem chce się komunikować. Aby rozpocząć komunikację, trzeba nawiązać połączenie z urządzeniem o określonym identyfikatorze. Jeśli identyfikator jest nieznany, przeprowadza się procedurę wykrywania, która pozwala na uzyskanie listy pobliskich urządzeń z włączonymi interfejsami Bluetooth. Można z niej wybrać pożądany sprzęt i spróbować nawiązać połączenie. Czasem jednak konieczne będzie tzw. parowanie, czyli autoryzowanie urządzeń, by mogły się ze sobą komunikować. Najczęściej parowanie odbywa się poprzez wprowadzenie 4-cyfrowego kodu PIN, podanego przez jedno z urządzeń (lub zapisanego w instrukcji tego urządzenia). Parowanie pozwala zapisać dany sprzęt Bluetooth w pamięci, co umożliwia automatyczne łączenie się np. telefonu z wybranym urządzeniem. Warto też wspomnieć o klasach urządzeń Bluetooth, które w ogólny sposób określają przeznaczenie danego sprzętu. Nie chodzi jednak o klasy interfejsu radiowego, które związane są z mocą nadawczą. Klasy urządzeń (Class of Device) są tworzone poprzez sumę bitową wartości odpowiadających poszczególnym funkcjom i cechom, opisującym dane urządzenie. Opis klas urządzeń Bluetooth zawarto w **tabelach 1, 2 i 3**.

Polecenia wtyczki Bluetooth Serial

Zainstalowany plugin tworzy obiekt **bluetoothSerial**, który zawiera 19 metod. Można je podzielić na dwie kategorie: metody odpowiadające za zestawianie połączeń i ustawienia interfejsu oraz funkcje do przesyłania i odbierania danych w oparciu o istniejące połączenie. Pierwsza grupa obejmuje następujące funkcje:

- **enable(sukces, porażka)** – funkcja ta działa tylko w systemie Android i pozwala na wyświetlenie użytkownikowi komunikatu, by włączył interfejs Bluetooth w urządzeniu mobilnym. Jeśli to zrobi, wywoływana jest funkcja, której nazwa jest podana jako parametr sukces, a jeśli nie – funkcja, której nazwa jest podana jako drugi parametr;
- **isEnabled(sukces, porażka)** – funkcja sprawdzająca, czy interfejs Bluetooth jest włączony;

Tabela 1. Bity 13…23 24-bitowego numeru CoD, opisującego klasę urządzenia Bluetooth. Najstarsze bity określają typu usług realizowanych przez urządzenie

Bity											Główna klasa oferowanych usług
23	22	21	20	19	18	17	16	15	14	13	
0	0	0	0	0	0	0	0	0	0	1	Ograniczony tryb wykrywania
0	0	0	0	0	0	0	0	0	1	0	Zarezerwowane
0	0	0	0	0	0	0	0	1	0	0	Zarezerwowane
0	0	0	0	0	0	0	1	0	0	0	Pozycjonowanie
0	0	0	0	0	0	1	0	0	0	0	Usługi sieciowe
0	0	0	0	0	1	0	0	0	0	0	Usługi wyjścia
0	0	0	0	1	0	0	0	0	0	0	Usługi wejścia
0	0	0	1	0	0	0	0	0	0	0	Przesyłanie obiektów
0	0	1	0	0	0	0	0	0	0	0	Audio
0	1	0	0	0	0	0	0	0	0	0	Telefonia
1	0	0	0	0	0	0	0	0	0	0	Informacje

Tabela 2. Bity 8…12 24-bitowego numeru CoD Bluetooth, określające główny typ urządzenia

Bity					Główna klasa urządzenia
12	11	10	9	8	
0	0	0	0	0	Pozostałe
0	0	0	0	1	Komputery
0	0	0	1	0	Telefony
0	0	0	1	1	Urządzenia sieciowe
0	0	1	0	0	Urządzenia audio/wideo
0	0	1	0	1	Urządzenia peryferyjne
0	0	1	1	0	Urządzenia do obrazowania
0	0	1	1	1	Urządzenia noszone
0	1	0	0	0	Zabawki
0	1	0	0	1	Urządzenia medyczne
1	1	1	1	1	Nieskategoryzowane

- **discoverUnpaired(sukces, porażka)** – kolejna funkcja wspierana tylko przez system Android. Pozwala na wykrycie w otoczeniu niesparowanych urządzeń Bluetooth. Jeśli operacja się powiedzie, do funkcji o nazwie podanej jako pierwszy parametr przekazywany jest obiekt zawierający listę znalezionych urządzeń. Obiekt zwiera ich klasy, identyfikatory, adresy (w przypadku Androida identyczne z identyfikatorami) i nazwy;
- **list(sukces, porażka)** – funkcja przeznaczona do wykrywania pobliskich, sparowanych urządzeń Bluetooth, choć w systemie iOS wyświetlane są wszystkie okoliczne, a nie tylko sparowane. W Androidzie przekazuje funkcji sukces obiekt identycznie skonstruowany, jak w przypadku funkcji **discoverUnpaired()**. W iOSie obiekt nie zawiera klasy ani adresu, tylko id, uuid (identyczne z id i będące zarazem adresem), nazwę, a niekiedy także informację o mocy sygnału. W Windows Phone podawane są identyfikatory, będące adresami MAC z dodatkowymi nawiasami oraz nazwy;
- **connect(adres_lub_identyfikator, sukces, porażka)** – funkcja ta służy do nawiązywania połączenia z urządzeniem o znanym adresie lub identyfikatorze. W przypadku iOSa, jeśli nie poda się żadnego identyfikatora, nawiązywane jest połączenie z pierwszym znalezionym sprzętem. Funkcje sukcesu i porażki nie przyjmują żadnych parametrów. Ostatnia z nich wywoływana jest nie tylko gdy wystąpi błąd, ale również w momencie zakończenia połączenia poleceniem **disconnect()**;
- **connectInsecure(adres_lub_identyfikator, sukces, porażka)** – funkcja działająca jedynie na Androidzie i pozwalająca na podłączenie niektórych starszych urządzeń Bluetooth. Takie połączenie jest niezabezpieczone i jest podatne na różnego rodzaju ataki, w związku z czym zaleca się stosowanie zwykłej metody **connect()**, o ile tylko to możliwe;
- **isConnected(sukces, porażka)** – prosta funkcja, pozwalająca sprawdzić, czy telefon jest aktualnie podłączony do jakiegoś urządzenia Bluetooth;
- **disconnect(sukces, porażka)** – funkcja umożliwiająca przerwanie aktywnego połączenia. Przyjmuje opcjonalne parametry. Funkcja porażki jest wywoływana tylko, gdy wystąpi błąd (np. gdy interfejs Bluetooth jest wyłączony), a samo zakończenie połączenia powoduje wywołanie funkcji podanej jako ostatni parametr w momencie wywoływania funkcji **connect()** lub **connectInsecure()**;
- **readRSSI(sukces, porażka)** – funkcja umożliwiająca odczyt aktualnej mocy sygnału pochodzącego z podłączonego urządzenia Bluetooth. W przypadku sukcesu, przekazuje stosowną wartość jako parametr. Funkcja eksperymentalna;
- **showBluetoothSettings(sukces, porażka)** – funkcja otwierająca okno ustawień Bluetooth, przyjmująca opcjonalnie dwa parametry. Metoda ta nie działa w systemie iOS.

Przykładowa procedura nawiązania połączenia, może obejmować następujące funkcje, wywoływane po sobie, z tym, że poprzez przekazywanie nazw jako parametru „sukces”:

- sprawdzenie, czy Bluetooth jest włączony: **isEnabled()**,
- poproszenie użytkownika o włączenie interfejsu: **enable()**,
- wykrycie dostępnych, sparowanych urządzeń: **list()**,
- nawiązanie połączenia: **connect()**,
- przesyłanie danych z regularnym sprawdzaniem funkcją **isConnected()** czy połączenie jest wciąż utrzymane,
- rozłączenie połączenia: **disconnect()**.

Transmisję danych z użyciem omawianego pluginu może być prowadzony na kilka różnych sposobów. Oto przydatne funkcje:

- **available(sukces, porażka)** – funkcja sprawdza, ile danych znajduje się w buforze odczytu interfejsu Bluetooth urządzenia mobilnego. Uzyskana wartość jest przekazywana jako parametr funkcji sukces;

Tabela 3. Bity 2…7 24-bitowego numeru CoD Bluetooth, określające podklasy dla poszczególnych klas głównych urządzenia. Wartości „X" oznaczają możliwość przeprowadzenia sumowania bitowego w ramach danej podklasy, celem wskazania wielofunkcyjności danego urządzenia. Pozostałe konfiguracje bitowe są zarezerwowane. Bity #0 i #1 przyjmują wartość 0.

Bity						Podklasa urządzenia Bluetooth
7	6	5	4	3	2	
Klasa główna: komputer						
0	0	0	0	0	0	Nieskategoryzowane
0	0	0	0	0	1	Komputer biurkowy
0	0	0	0	1	0	Serwer
0	0	0	0	1	1	Laptop
0	0	0	1	0	0	Mały komputer przenośny
0	0	0	1	0	1	Komputer wielkości dłoni
0	0	0	1	1	0	Komputer wielkości zegarka
0	0	0	1	1	1	Tablet
Klasa główna: telefon						
0	0	0	0	0	0	Nieskategoryzowane
0	0	0	0	0	1	Komórkowy
0	0	0	0	1	0	Bezprzewodowy
0	0	0	0	1	1	Smartfon
0	0	0	1	0	0	Modem przewodowy
0	0	0	1	0	1	ISDN
Klasa główna: urządzenie sieciowe						
0	0	0	X	X	X	W pełni dostępne
0	0	1	X	X	X	Obciążone w od 1% do 17%
0	1	0	X	X	X	Obciążone w od 17% do 33%
0	1	1	X	X	X	Obciążone w od 33% do 50%
1	0	0	X	X	X	Obciążone w od 50% do 67%
1	0	1	X	X	X	Obciążone w od 67% do 83%
1	1	0	X	X	X	Obciążone w od 83% do 99%
1	1	1	X	X	X	Aktualnie niedostępne
Klasa główna: urządzenie audio/wideo						
0	0	0	0	0	0	Nieskategoryzowane
0	0	0	0	0	1	Zestaw słuchawkowy z mikrofonem
0	0	0	0	1	0	Zestaw głośnomówiący
0	0	0	0	1	1	Zarezerwowane
0	0	0	1	0	0	Mikrofon
0	0	0	1	0	1	Głośnik
0	0	0	1	1	0	Słuchawki
0	0	0	1	1	1	Przenośne audio
0	0	1	0	0	0	Audio samochodowe
0	0	1	0	0	1	Dekoder TV
0	0	1	0	1	0	Sprzęt Hi-Fi audio
0	0	1	0	1	1	Magnetowid
0	0	1	1	0	0	Aparat fotograficzny
0	0	1	1	0	1	Kamera
0	0	1	1	1	0	Wyświetlacz wideo
0	0	1	1	1	1	Wyświetlacz wideo z głośnikiem
0	1	0	0	0	0	Urządzenie do wideokonferencji
0	1	0	0	0	1	Zarezerwowane
0	1	0	0	1	0	Gra wideo
Klasa główna: urządzenie peryferyjne						
0	0	X	X	X	X	Ani klawiatura ani inne urządzenie wskazujące
0	1	X	X	X	X	Klawiatura
1	0	X	X	X	X	Urządzenie wskazujące
1	1	X	X	X	X	Urządzenie wskazujące z klawiaturą
X	X	0	0	0	0	Nieskategoryzowane
X	X	0	0	0	1	Dżojstik
X	X	0	0	1	0	Pad do gier
X	X	0	0	1	1	Pilot zdalnego sterowania
X	X	0	1	0	0	Urządzenie dotykowe
X	X	0	1	0	1	Tablet graficzny
X	X	0	1	1	0	Czytnik kart (np. SIM)
X	X	0	1	1	1	Cyfrowe pióro
X	X	1	0	0	0	Ręczny skaner kodów
X	X	1	0	0	1	Urządzenie do detekcji gestów (np. różdżka)
Klasa główna: urządzenie do obrazowania						
X	X	X	1	0	0	Wyświetlacz
X	X	1	X	0	0	Aparat fotograficzny
X	1	X	X	0	0	Skaner
1	X	X	X	0	0	Drukarka
Klasa główna: urządzenie noszone						
0	0	0	0	0	1	Zegarek

Tabela 3. c.d.

Bity						Podklasa urządzenia Bluetooth
7	6	5	4	3	2	
0	0	0	0	1	0	Pager
0	0	0	0	1	1	Kurtka
0	0	0	1	0	0	Hełm
0	0	0	1	0	1	Okulary
Klasa główna: zabawka						
0	0	0	0	0	1	Robot
0	0	0	0	1	0	Pojazd
0	0	0	0	1	1	Lalka
0	0	0	1	0	0	Kontroler
0	0	0	1	0	1	Gra
Klasa główna: urządzenie medyczne						
0	0	0	0	0	0	Niezdefiniowane
0	0	0	0	0	1	Miernik ciśnienia
0	0	0	0	1	0	Termometr
0	0	0	0	1	1	Waga
0	0	0	1	0	0	Miernik poziomu glukozy
0	0	0	1	0	1	Miernik stężenia CO2 we krwi
0	0	0	1	1	0	Ciśnieniomierz lub pulsomierz
0	0	0	1	1	1	Urządzenie do prezentacji informacji o zdrowiu
0	0	1	0	0	0	Krokomierz
0	0	1	0	0	1	Analizator składu ciała
0	0	1	0	1	0	Pikflometr
0	0	1	0	1	1	Urządzenie do kontroli przyjmowania leków
0	0	1	1	0	0	Proteza kolana
0	0	1	1	0	1	Proteza kostki
0	0	1	1	1	0	Urządzenie ogólnomedyczne
0	0	1	1	1	1	Urządzenie usprawniające poruszanie się

- **read(sukces, porażka)** – metoda pobiera z buforu odczytu wszystkie dostępne dane i przekazuje je jako parametr funkcji sukces. Jeśli w buforze nie było żadnych danych, wciąż wywoływana jest funkcja sukces tyle, że jej parametrem będzie pusty ciąg znaków;
- **readUntil(ogranicznik, sukces, porażka)** – funkcja ułatwiająca odczyt ciągów znaków, zakończonych określonym ogranicznikiem. Pozwala np. na ładowanie danych linia po linii (ogranicznik '\n'), które są przekazywane jako parametr funkcji sukces. Jeśli bufor jest pusty, lub nie zawiera znaku ogranicznika, funkcja sukces wywoływana jest z pustym ciągiem znaków;
- **subscribe(ogranicznik, sukces, porażka)** – metoda ułatwiająca cykliczne odczytywanie ciągów znaków z interfejsu Bluetooth. Stale monitoruje zawartość bufora i wywołuje funkcję sukces, gdy tylko w buforze pojawi się określony ogranicznik. Przekazywane parametry są identyczne, jak w **readUntil()**. Funkcja działa w pętli, aż do momentu wywołania polecenia **unsubscribe()**;
- **unscubscribe(sukces, porażka)** – metoda przerywająca działanie funkcji **subscribe()**;
- **subscribeRawData(sukces, porażka)** – metoda analogiczna do funkcji **subscribe()**, ale przeznaczona do danych binarnych. Jest wywoływana, gdy do bufora Bluetooth trafią jakiekolwiek dane. Odczytane dane są przekazywane do funkcji sukces w postaci obiektu klasy **ArrayBuffer**;
- **unsubscribeRawData(sukces, porażka)** – metoda przerywająca działanie funkcji **subscribeRawData()**;
- **clear(sukces, porażka)** – metoda ta czyści bufor interfejsu Bluetooth;
- **write(dane, sukces, porażka)** – jedyna funkcja służąca do przesyłu, a nie odbioru danych. Pozwala na wysłanie przez szeregowy interfejs Bluetooth pakiety danych, którym może być ciąg znaków, tablica liczb całkowitych (również podana w postaci wartości szesnastkowych) lub obiekt klasy **ArrayBuffer**.

Wymiana danych, po pomyślnym nawiązaniu połączenia, może polegać np. na wywołaniu funkcji **subscribe()**, której drugim parametrem będzie funkcja obejmująca wywołanie metody **write()**. Dzięki temu urządzenie mobilne będzie mogło stale monitorować dane nadchodzące przez Bluetooth i odpowiadać na nadchodzące żądania.

Bluetooth w przykładzie

W naszym przypadku użyjemy interfejsu Bluetooth do otwierania i zamykania bramy poprzez przesyłanie jednoliterowych komend z pilota do napędu. Na program składać się będzie funkcja inicjalizacji połączenia Bluetooth, uruchamiana przy włączeniu pilota oraz przypisane do przycisków polecenia do przesyłania informacji interfejsem Bluetooth.

Inicjalizację połączenia należy rozpocząć od wykrycia niesparowanego urządzenia poleceniem **bluetoothSerial.discoverUnpaired()**. Jeśli polecenie uda się pomyślnie wykonać, przechodzimy przez listę znalezionych urządzeń, w poszukiwaniu tego jednego, które chcemy obsługiwać, a adres tego jednego urządzenia, zapisujemy – **listing 1**.

We fragmencie programu na list. 1, do iteracyjnego przeglądania listy znalezionych urządzeń używamy funkcji **forEach**. Kluczowe jest natomiast dokonanie połączenia z użyciem polecenia **bluetoothSerial.connect()**, dzięki któremu wszystkie przesyłane następnie dane Bluetoothem będą kierowane odpowiedniego urządzenia.

Należy przy tym zaznaczyć, że skanowanie otoczenia i nawiązywanie połączenia Bluetooth trwa trochę

Listing 1. Wykrycie niesparowanych urządzeń

```
devices.forEach(function(device)
{
    if (device.name==”Brama”)
    {
        app.btaddr=device.address;
        bluetoothSerial.connect(app.btaddr, function(){alert(„połączono”);}, function(){alert(„zakończono
połączenie”);});
    }
})
```

Listing 2. Kod JavaScript

```
var app = {
    initialize: function() {
    this.bindEvents();
    },
    bindEvents: function() {
        document.addEventListener(‚deviceready’, this.onDeviceReady, false);
    },
    onDeviceReady: function() {
        app.receivedEvent(‚deviceready’);
        app.bt1();
        $(„#openButton”).click(function(){
            bluetoothSerial.write(‘o’, function(){alert(„otwarto”);}, function(){alert(„wystąpił błąd”);});
        });
        $(„#closeButton”).click(function(){
        bluetoothSerial.write(‘z’, function(){alert(„zamknięto”);}, function(){alert(„wystąpił błąd”);});
        });
    },

    receivedEvent: function(id) {
        var parentElement = document.getElementById(id);
        var listeningElement = parentElement.querySelector(‚.listening’);
        var receivedElement = parentElement.querySelector(‚.received’);
        listeningElement.setAttribute(‚style’, ‚display:none;’);
        receivedElement.setAttribute(‚style’, ‚display:block;’);
        console.log(‚Received Event: ‚ + id);
    },
    btaddr:null,
    bt1 : function(){
        bluetoothSerial.discoverUnpaired(function(devices) {
            devices.forEach(function(device) {
                alert(device.address);
                if (device.name==”Brama”){
                    app.btaddr=device.address;
                    bluetoothSerial.connect(app.btaddr, function(){alert(„połączono”);}, function()
{alert(„zakończono połączenie”);});
                }
            })
        }, function(){
            alert(„error”);
        });
    },
};
app.initialize();
```

Listing 3. Kod HTML (tylko fragment BODY)

```
<body>
    <div class=”app”>
        <h1>Apache Cordova</h1>
        <div id=”deviceready” class=”blink”>
            <p class=”event listening”>Connecting to Device</p>
            <p class=”event received”>Device is Ready</p>
        </div>
    </div>
    <script type=”text/javascript” src=”cordova.js”></script>
    <script type=”text/javascript” src=”js/jquery-2.1.3.min.js”></script>
    <script type=”text/javascript” src=”js/index.js”></script>
    <div style=”display:table;width:100%;height:100px;background-color:green;font-size:xx-large;text-
align:center”>
        <div id=”openButton” style=”display:table-cell; vertical-align: middle;”>OTWÓRZ</div>
    </div>
    <div id=”closeButton” style=”display:table;width:100%;height:100px;background-color:red;font-size:xx-
large;text-align:center”>
        <div style=”display:table-cell; vertical-align: middle;”>ZAMKNIJ</div>
    </div>
</body>
```

czasu i trzeba poczekać kilka-kilkanaście sekund, zanim operacja się powiedzie. Oczywiście, całość dzieje się w tle, dzięki czemu reszta aplikacji działa w międzyczasie normalnie.

Wysyłanie polecenia otwarcia lub zamknięcia bramy przypiszemy do dwóch nowych przycisków. Naciśnięcie ich będzie powodowało przesłanie przez Bluetooth, poleceniem **bluetoothSerial.write()** pojedynczych znaków, które napęd będzie interpretował jako komendy otwarcia lub zamknięcia bramy.

```
$(„#openButton”).click(function(){
bluetoothSerial.write(‘o’, function()
{alert(„otwarto”);}, function()
{alert(„wystąpił błąd”);});
});
```

Cały dodatkowy kod JavaScript, potrzebny do realizacji funkcji otwierania i zamykania bramy Bluetoothem został zebrany na **listingu 2**, a na **listingu 3** umieszczono potrzebny dodatkowy kod HTML.

Marcin Karbowniczek, EP