



Katedra Systemów Geoinformatycznych

System GPS i jego zastosowania

Laboratorium

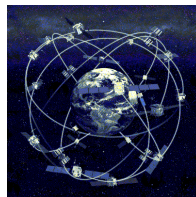
Komunikacja z odbiornikiem

GPS w technologii .NET

Compact Framework z

wykorzystaniem biblioteki

GPS Intermediate Device



Wstęp

W niniejszym ćwiczeniu Student zapoznaje się ze sposobem wykorzystania biblioteki GPS Intermediate Driver (GPS ID) przeznaczonej do komunikowania się z wbudowanym odbiornikiem GPS w urządzeniach mobilnych wyposażonych w system Windows Mobile w wersji 6.0 lub wyższej. W przeciwieństwie do poprzedniego ćwiczenia, użyte zostanie gotowe API dostarczone przez projektantów systemu Windows, które pozwala, bez konieczności kontrolowania szczegółów komunikacji, odbierać podstawowe dane nawigacyjne z odbiornika GPS.

GPS Intermediate Driver

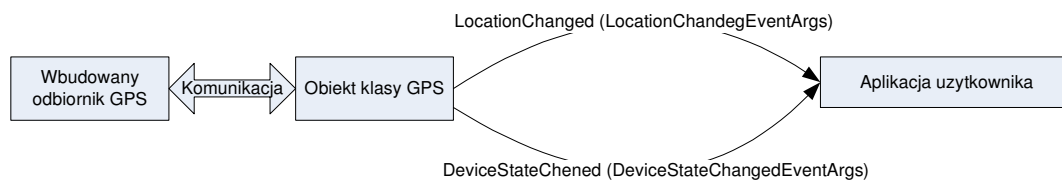
The GPS Intermediate Driver (GPS ID) jest warstwą programistyczną odpowiadającą za komunikację pomiędzy urządzeniem odbiorczym GPS a aplikacją deweloperską stworzoną w technologii .NET Compact Framework. Pozwala ona na zunifikowanie sposobu dostępu do różnych urządzeń GPS zapewniając pełną przenośność programów wykorzystujących GPS ID pomiędzy poszczególnymi implementacjami systemu Windows Mobile. Obsługa GPS jest możliwa dzięki udostępnieniu, poprzez platformę Platform Invocation Service (była przedstawiana na poprzednich ćwiczeniach), kodu biblioteki natywnej napisanej w języku C++ implementującej komunikację na poziomie portów szeregowych pomiędzy urządzeniem GPS a zasobami systemowymi. W przypadku wykorzystania omawianej funkcjonalności znajomość kodu C++ nie jest konieczna, wystarczy rozumienie podstawowych zasad projektowania obiektowego oraz mechanizmu przekazywania zdarzeń w języku C#. Poniżej opisane zostały najważniejsze klasy GPS ID API, które będą wykorzystywane w ćwiczeniu:

- Klasa **GPS** – klasa będąca programistyczną warstwą abstrakcyjną reprezentującą wbudowany odbiornik GPS. Obiekt klasy generuje dwa podstawowe zdarzenia:
 - `deviceStateChanged` – przekazywane przez delegat `DeviceStateChangedEventHandler` w momencie kiedy zmienia się stan urządzenia. Stan urządzenia zdefiniowany jest w klasie `GpsDeviceState`, która także definiuje szczegóły komunikacji z odbiornikiem GPS. Argumenty zdarzenia przechowywane są w klasie `DeviceStateChangedEventArgs`.
 - `locationChanged` – zdarzenie generowane w momencie gdy odbiornik wyznaczy nową pozycję, jest przekazywane poprzez delegat

`LocationChangedEventHandler`. Informacje o lokalizacji odbiornika przechowywane jest w obiekcie klasy `GpsPosition`. Argumenty zdarzenia przekazywane są w obiekcie klasy `LocationChangedEventArgs`.

- Klasa **Utils** – klasa pomocnicza ułatwiająca komunikację z wykorzystaniem platformy P/Invoke.
- Klasa **DegreesMinutesSeconds** – klasa ułatwiająca konwersję pozycji geograficznej z formatu dziesiętnego do formatu stopień, minuta, sekunda.

Ogólny schemat przepływu danych w omawianym rozwiązaniu można przedstawić schematycznie tak jak to pokazane zostało na Rys. 1.



Rys. 1. Uproszony schemat przepływu danych w omawianym rozwiązaniu.

Zadanie 1

Zadanie 1 polega na prawidłowej konfiguracji urządzenia mobilnego umożliwiającej korzystanie z wbudowanego urządzenia GPS oraz stworzeniu podstawowej aplikacji odczytującej pozycję. W pierwszym etapie ćwiczenia laboratoryjnego praca będzie odbywała się jedynie na emulatorach, natomiast w końcowym etapie będzie możliwość sprawdzenia działania programu na przykładzie telefonu komórkowego Samsung Omnia i900 wyposażonego we wbudowany odbiornik GPS.

Wykonywanie ćwiczenia zaczynamy od uruchomienia rozwiązania (ang. solution) Visual Studio o nazwie *GPS.sln* znajdującego się w katalogu roboczym niniejszego ćwiczenia. W projekcie o nazwie *GpsSample* stworzony został podstawowy interfejs użytkownika (plik *Form1.cs*) wraz z zadeklarowanymi metodami, które w trakcie tego zadania będziemy wypełniać. Projekt *Microsoft.WindowsMobile.Samples.Location* stanowi natomiast API omawianej biblioteki.

Realizację zadania zaczynamy od otworzenia pliku *Form1.cs* i deklaracji w klasie *Form1* następujących zmiennych:

```
private EventHandler updateDataHandler;

GpsDeviceState device = null;

GpsPosition position = null;

Gps gps = new Gps();
```

Zmienna `updateDataHandler` będzie nam umożliwiała przekazanie informacji o zdarzeniach pochodzących z klasy GPS do GUI użytkownika. W zmiennych `device` oraz `position` będą przechowywane parametry działania urządzenia GPS reprezentowanego poprzez obiekt `gps`. Następnie przystępujemy do modyfikacji metody `Form1_Load` zdefiniowanej w linii 137 pliku `Form1.cs`. Metoda `Form1_Load` wywołana jest w momencie tworzenia okna formularza `Form1`, a jej zawartość powinna być uzupełniona o poniższą linię kodu:

```
updateDataHandler = new EventHandler(UpdateData);
```

Powyższe polecenie pozwala nam na zdefiniowanie `EventHandler`, czyli obiektu przekazującego informację o wystąpieniu zdarzenia, oraz akcji jaka ma po wystąpieniu zdarzenia wykonać. W naszym przypadku jest to wywołanie funkcji `UpdateData`, którą zdefiniujemy później – na tym etapie wystarczy założyć, iż funkcja ta będzie aktualizowała zawartość interfejsu w momencie wyznaczenia nowej pozycji przez odbiornik lub zmiany jego stanu.

Następnie, należy zdefiniować jaka metoda ma być wywoływana w momencie gdy obiekt `gps` wygeneruje zdarzenie o zmianie stanu lub zmianie położenia. Realizuje się to za pomocą „przywiązania” zdarzeń `DeviceStateChanged` oraz `LocationChanged` do odpowiednich delegatów zdefiniowanych w GPS ID API:

```
gps.DeviceStateChanged+=new DeviceStateChangedEventHandler(gps_DeviceStateChanged);
gps.LocationChanged+= new LocationChangedEventHandler(gps_LocationChanged);
```

Definicja powyższych linijek w metodzie `Form1_Load` spowoduje ich wywołanie przy uruchomieniu aplikacji oraz umożliwi przekazywanie zdarzeń generowanych przez klasę GPS do odpowiednich metod, w tym przypadku metod `gps_DeviceStateChanged` oraz `gps_LocationChanged`, których definicje znajdują się poniżej:

```

protected void gps_LocationChanged(object sender, LocationChangedEventArgs args)
{
    position = args.Position;
    Invoke(updateDataHandler);
}

protected void gps_DeviceStateChanged(object sender, DeviceStateChangedEventArgs args)
{
    device = args.DeviceState;
    Invoke(updateDataHandler);
}

```

Obie metody będą wywoływane w momencie wygenerowania zdarzenia przez klasę GPS, będą uaktualniały dane pochodzące z odbiornika przechowywane w obiektach *position* oraz w *device* będących atrybutami klasy Form1. Należy także zauważyć, iż korzystanie z metody Invoke jest tutaj konieczne gdyż zdarzenia przekazywane przez metodę updateDataHandler generowane są w innym wątku niż GUI użytkownika, zatem bezpośrednie wywołanie metody updateData mogłoby skutkować wygenerowaniem wyjątku *cross-trhead operation exception*. Więcej na temat unikania błędów związanych z programowaniem wielowątkowym można znaleźć w:

<http://msdn.microsoft.com/en-us/library/aa446540.aspx>

Następnie przystępujemy do implementacji zadeklarowanej wcześniej metody *UpdateData*. Metoda ta wywoływana jest w momencie wystąpienia jednego ze zdarzeń *gps_DeviceStateChanged* lub *gps_LocationChanged*, a jej zadaniem jest pobrać aktualne informacje dotyczące pozycji odbiornika i pokazać je w GUI użytkownika w komponencie graficznym Label o nazwie *status* (patrz Form1 Designer). Jej definicja wygląda następująco:

```

void UpdateData(object sender, System.EventArgs args)
{
    if (gps.Opened)
    {
        string str = "";
        if (position != null)
        {
            if (position.LatitudeValid)
                str += "Latitude (D,M,S):\n " +
                    position.LatitudeInDegreesMinutesSeconds + "\n";

            if (position.LongitudeValid)
                str += "Longitude (D,M,S):\n " +
                    position.LongitudeInDegreesMinutesSeconds + "\n";
        }
        status.Text = str;
    }
}

```

Należy także zauważyć, iż do metody UpdateData przekazywane są dwa argumenty (*object sender*, *System.EventArgs args*), dzieje się tak ponieważ metoda ta wywoływana jest za

pomocą metody `Invoke`, która powoduje automatyczne przekazanie ww. zmiennych do metody wywoływanej poprzez `updateDataHandler`.

Na koniec potrzebna jest jeszcze definicja metod uruchamiających i kończących działanie urządzenia GPS:

```
private void stopGpsMenuItem_Click(object sender, EventArgs e)
{
    if (gps.Opened)
    {
        gps.Close();
    }

    startGpsMenuItem.Enabled = true;
    stopGpsMenuItem.Enabled = false;
}
```


```
private void startGpsMenuItem_Click(object sender, EventArgs e)
{
    if (!gps.Opened)
    {
        gps.Open();
    }

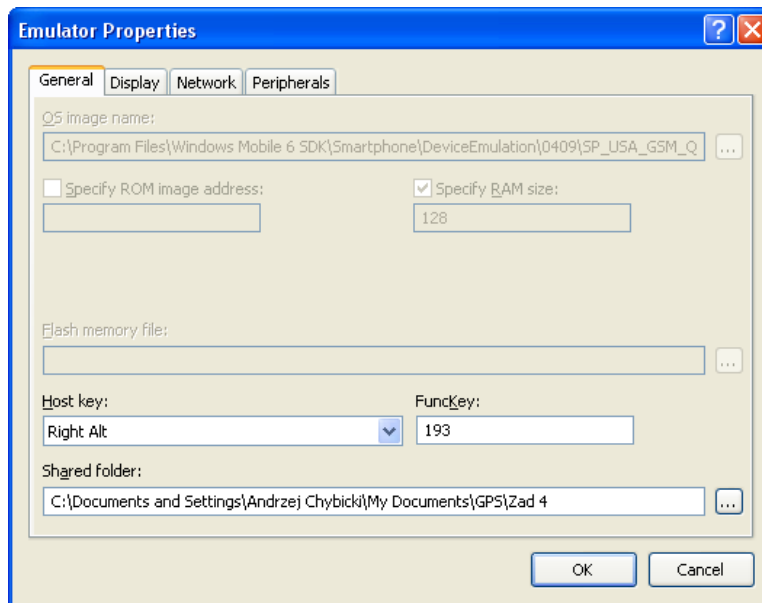
    startGpsMenuItem.Enabled = false;
    stopGpsMenuItem.Enabled = true;
}
```

oraz metody kończącej działanie programu:

```
private void exitMenuItem_Click(object sender, EventArgs e)
{
    if (gps.Opened)
    {
        gps.Close();
    }

    Close();
}
```

Aby móc testować działanie stworzonej aplikacji musimy na wykorzystywanym emulatorze urządzenia mobilnego zainstalować program *FakeGPS* (dostępny w katalogu roboczym ćwiczenia), który jest programistycznym symulatorem wbudowanego odbiornika GPS. Rozwiązanie to pozwoli nam w warunkach laboratoryjnych na testowanie działania wykorzystywanej biblioteki. Uruchomienie emulatora wywołujemy poprzez naciśnięcie przycisku  w oknie narzędzi Visual Studio. Następnie, aby w łatwy sposób uruchomić program instalacyjny programu *FakeGPS.cab* znajdujący się w katalogu roboczym komputera, musimy zmapować ten katalog na pamięć emulatora. Wykonujemy to odpowiednio definiując pole *Shared Folder* w *Menu->File>Configure* okna emulatora (patrz Rys.2). W wyniku tego działania, emulator urządzenia mobilnego będzie traktował uprzednio zdefiniowany *Shared Folder* jako pamięć *Storage Card* urządzenia mobilnego.



Rys. 2. Mapowanie folderu komputera PC na pamięć Storage Card emulatora.

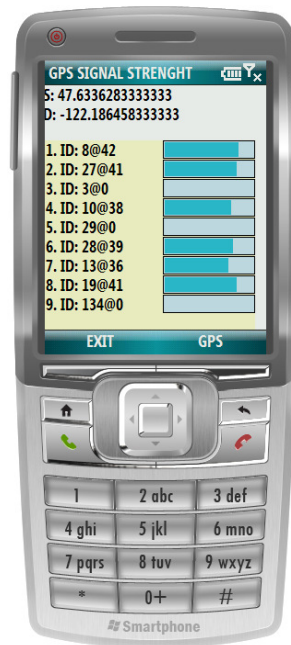
Pozostaje zatem uruchomienie na emulatorze eksploratora plików (*Start->More->FileExplorer*), uruchomienie w emulatorze pliku instalacyjnego *FakeGPS.cab* oraz uruchomienie samego symulatora FakeGPS (*Start->More->FakeGPS*). Po uruchomieniu symulator będzie chodził w tle. Aby przetestować działanie stworzonej przez nas aplikacji wystarczy przekompilować stworzony kod oraz nacisnąć przycisk *StartGPS* z menu stworzonego przez nas programu.

Uruchomienie aplikacji, która pokazuje pozycję otrzymywaną z symulatora GPS, jest ostatnim elementem zadania 1.

Zadanie 2

Zadanie 2 polega na modyfikacji programu w taki sposób, aby w GUI użytkownika widoczna była nie tylko pozycja odbiornika, ale także siła sygnału pochodzącego od widzianych przez odbiornik satelitów. Informację o sile sygnału można pobrać korzystając z obiektu *GPSPosition* wywołując metodę *GetSatellitesInView*, która zwraca tablicę obiektów klasy *Satellite*. Informacja o sile sygnału przechowywana jest w atrybucie *SignalStrength*. Wizualizację siły sygnału można wykonać z wykorzystaniem komponentu GUI o nazwie *ProgressBar*, który jest dostępny w *Toolbox Visual Studio*.

Należy pamiętać także o odpowiedniej modyfikacji metody *UpdateData* odpowiedzialnej za realizację GUI użytkownika. Wynik działania programu powinien być maksymalnie zbliżony do tego jak pokazano na Rys. 3.



Rys. 3. Przykładowy wynik działania programu z zadania 2.

Zadanie 3

W obiektach klasy *Satellite* oprócz siły sygnału dostępna jest także informacja mówiąca o położeniu satelity względem odbiornika, jest ona definiowana za pomocą dwóch parametrów:

- Elavation – definiuje kąt w płaszczyźnie pionowej pomiędzy kierunkiem patrzenia na satelitę przez odbiornik a kierunkiem stycznym do płaszczyzny geoidy w punkcie zdefiniowanym przez położenie odbiornika
- Azymut – kierunek w płaszczyźnie poziomej podany w stopniach, w jakim satelita jest widoczny z punktu położenia odbiornika.

Zadanie 3 polega na dwuwymiarowej wizualizacji położenia satelitów względem odbiornika.

Wynik działania programu powinien wyglądać możliwie podobnie do tego jak pokazano na Rys. 4.



Rys. 4. Wizualizacja położenia satelit.

WSKAZÓWKI

- W celu rysowania poszczególnych elementów grafiki reprezentujących satelity w przestrzeni 2D można się posłużyć biblioteką GDI zdefiniowaną w przestrzeni nazw System.Drawing. Zestaw klas GDI w .NET Compact Framework jest niemal identyczny ze standardowym zestawem GDI dostępnym w .NET Framework,
- rysowanie w GDI obiektów 2D najłatwiej zrealizować z wykorzystaniem obiektu Graphics lub Graphics2D poprzez nadpisanie (override) metody On_Paint na wybranym komponencie GUI. Szczegóły dotyczące tych zagadnień można znaleźć w:
<http://msdn.microsoft.com/en-us/library/ms172503.aspx>
- opisane programy można przetestować w warunkach terenowych na dostępnym przy stanowisku laboratoryjnym telefonie Samsung Omnia i900