Drukuj Zamknij

Bardzo prosty klient poczty na Smartphone'a...

2003-11-30 19:17:24
Artur Gniadzik - Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

Wprowadzenie

Tworzenie aplikacji na urządzenia przenośne nieco różni się od tworzenia aplikacji, do jakich jesteśmy przyzwyczajeni, choćby przez ograniczenia, jakie urządzenia te narzucają na programistę. W poniższym artykule, na przykładzie bardzo prostego klienta poczty, przedstawię sposób tworzenia typowej aplikacji na urządzenia przenośne – typowej, tzn. dającej możliwość korzystania z sieci, pobierania z niej danych, przetwarzania ich i zapisywania w pamięci telefonu.

Co będzie nam potrzebne?

Z pewnością przyda się Visual Studio .NET – znacząco ułatwia tworzenie aplikacji w technologii .NET, również na urządzenia przenośne. Dodatkowo potrzebujemy „MSASYNC.EXE" i „Microsoft SMARTPHONE 2003 SDK.msi". Najpierw należy zainstalować ActiveSync. I tu mała uwaga: po zainstalowaniu ActiveSync-a nastąpi wyszukiwanie urządzenia podłączonego do jednego z portów komputera. Jest to spowodowane tym, że mamy możliwość testowania naszej aplikacji na rzeczywistym urządzeniu podłączonym do komputera. W naszym przypadku skorzystamy jednak z emulatora, więc możemy pominąć ten krok przerywając wyszukiwanie („CANCEL").

Po zainstalowaniu ActivSync-a pozostaje nam jeszcze zainstalowanie SDK i jesteśmy gotowi do tworzenia aplikacji na urządzenia przenośne.

Tworzenie aplikacji

W Visual Studio tworzymy nowy projekt, wybieramy *Smart Device Application*, a następnie *Smartphone* i *Windows Application* i jesteśmy gotowi do tworzenia aplikacji.

Komunikacja Sieciowa

Chcemy, aby nasz klient łączył się z serwerem POP3 i odbierał pocztę ze wskazanego konta. W tym celu tworzymy nową klasę (Komunikator), która się tym zajmie.

Komunikacja przy użyciu protokołu TCP/IP została już opisana w innym artykule i nie będę jej tu omawiać, gdyż w przypadku urządzeń przenośnych jest właściwie identyczna.

Komunikacja z serwerem POP3 odbywa się według schematu zapytanie-odpowiedź. Protokół POP3 jest oparty na protokole znakowym telnet, więc zarówno zapytania, jak i odpowiedzi są zwykłym tekstem.

Przykładowa sesja POP3 wygląda tak:

```
+OK ArGoSoft Mail Server Freeware, Version 1.8 (1.8.3.2)
user a
+OK Password required for a
pass a
+OK Mailbox locked and ready
quit
+OK
```

Odpowiedzi, jakie generuje serwer zawsze zaczynają się od wiersza zaczynającego się od „+OK.” lub „-ERR” (wierszy może być więcej, np. gdy zażądamy wiadomości). Do sprawdzenia, czy polecenie zostało przyjęte wystarczyłoby więc odczytanie pierwszych 3 znaków ze strumienia, jednak musimy odebrać całą odpowiedź, gdyż w przeciwnym przypadku „zalegałaby” nam ona w strumieniu danych, a to znacząco utrudniłoby wydanie innych poleceń. Najprościej byłoby użyć do tego po prostu polecenia Read(), z ustawioną ilością czytanych znaków na maksymalny rozmiar bufora, ale chcemy mieć możliwość przetwarzania wyniku wiersz po wierszu, więc musimy wymyślić coś innego. Utworzymy metodę readLine(), która się tym zajmie:

```
private string readLine()
{
     /**
      * Metoda pomocnicza - czyta ciąg znaków z wejścia, aż
      * do napotkania znaku końca linii
      */
     int i = 0;
     bool koniec = false;
     while (!koniec)
     {
          stream.Read(bufor, i, 1);
          if ((char)bufor[i]==´\n´) koniec=true;
          i++;
     }
     return Encoding.ASCII.GetString(bufor,0,i);
}
```

Nie możemy do czytania pojedynczych bajtów użyć metody ReadByte, bo nie jest blokująca (ReadByte() czyta ze strumienia i zwraca –1 po natrafieniu na jego koniec, a może się okazać, że zaczniemy czytać, zanim dane się pojawią lub skończymy, zanim na strumieniu pojawia się wszystkie dane, które chcemy odczytać).

Do ściągania wiadomości używamy metody pobierzWiadomosc(), która łączy się z serwerem, loguje użytkownika i po sprawdzeniu, o którą wiadomość tak naprawdę chodzi, ściąga ją, przetwarza i zwraca jako obiekt klasy Wiadomość. Obiekt ten będzie nam służył do przechowywania istotnych informacji o wiadomościach, tzn. tematu, treści oraz autora, a także znacznika wskazującego, czy użytkownik chce daną wiadomość usunąć. Poniżej przedstawiamy pola klasy wiadomość:

```
public class Wiadomosc
{
     string od;               //nadawca wiadomosci
     string temat;          //temat wiadomosci
     string tekst;          //tresc wiadomosci
     bool usunieta = false;     /*w tej chwili nie uzywana: jesli true, to wiadomosc zostala
´usunieta´ przez uzytkownika (w razie, gdybysmy zdecydowali sie na jej uzycie, nie bedziemy
musieli zmieniac formatu pliku) */
     ...
}
```

Przechowywanie danych w telefonie

Umiemy już pobrać wiadomości z serwera, chcielibyśmy jednak mieć do nich dostęp również offline, bez potrzeby ponownego pobrania już raz ściągniętych wiadomości. Najprostszym sposobem przechowywania danych jest zapisanie ich do pliku. W naszym przypadku będziemy zapisywać obiekty klasy Wiadomosc, więc dodajemy w tej klasie metodę zapisz(string nazwa):

```
public void zapisz(BinaryWriter w) {
     w.Write(usunieta);
     w.Write(od);
     w.Write(temat);
     w.Write(tekst);
}
public void zapisz(string nazwa)
{
     FileStream fs = new FileStream(nazwa, FileMode.Append, FileAccess.Write);
     BinaryWriter wr = new BinaryWriter(fs);
     zapisz(wr);
     wr.Close();
     fs.Close();
}
```

Otwieramy strumień, korzystając z klasy FileStream, podając nazwę pliku, tryb otwarcia (Append – utwórz, jeśli nie istnieje i dopisuj na końcu) oraz prawa dostępu. Następnie tworzymy BinaryWriter, który pozwoli nam pisać do tego pliku i zapisujemy interesujące nas pola obiektu Wiadomosc.

W analogiczny sposób dodajemy konstruktor, który utworzy nowy obiekt Wiadomosc – tym razem dostajemy jednak jako parametr nie nazwę pliku, ale od razu BinaryReader do odczytu z pliku. Dzięki temu BinaryReader sam wie, z którego miejsca pliku czytać. Teraz jeśli chcemy pokazać użytkownikowi wiadomość zapisaną w telefonie, wystarczy stworzyć obiekt klasy Wiadomosc(BinaryReader r).

Wyświetlanie wiadomości użytkownikowi

Umiemy już pobierać wiadomości, zapisywać je i odczytywać z pamięci telefonu. Pozostaje nam jeszcze przedstawienie tych możliwości użytkownikowi.

W naszej bardzo ograniczonej aplikacji chcemy mieć właściwie tylko dwie funkcje – odczytywanie wiadomości i pobieranie nowych. Musimy dodać takie opcje do menu. W Visual Studio możemy to wyklikać, ale należy uważać. Na jednym ekranie możemy mieć tylko dwa przyciski menu (co jest dość intuicyjne – mamy tylko dwa klawisze telefonu do tego celu), natomiast tylko jedno z tych menu może być rozwijalne – to z prawej strony. Visual Studio nie narzuci nam tych ograniczeń, ale nie stosując się do nich otrzymamy błąd podczas działania aplikacji.

Skoro mamy już gotowe menu, należy zastanowić się nad sposobem pokazywania wiadomości użytkownikowi. Zdecydowałem się na użycie ListView do wyświetlania listy tematów i MessageBox do prezentowania wiadomości.

Najbardziej przypadł mi do gustu widok „Details” :

```
lista.View= System.Windows.Forms.View.Details;
```

Żeby cokolwiek dało się tu jednak wyświetlić, należy dodać do listy co najmniej jedną kolumnę

```
ListView.Columns.Add
```

i dodać komponenty do wyświetlenia

```
ListView.Items.Add
```

Lista jest już gotowa do wyświetlenia. Po kliknięciu na pozycję z listy chcemy zobaczyć wiadomość. Dodajemy metodę obsługującą zdarzenie ItemActivate

```
lista.ItemActivate += new System.EventHandler(listView1_ItemActivate);
```

I sama metoda:

```
private void listView1_ItemActivate(object sender, System.EventArgs e)
{
    MessageBox.Show(
    ((Wiadomosc) wiad[listView1.SelectedIndices[0]]).ToString());
}
```

I jeszcze jedno – właściwie jesteśmy zwolnieni z tworzenia w menu opcji powrót – możemy po prostu posługiwać się klawiszem powrotu w telefonie.

Testowanie aplikacji

Tworzoną aplikację będziemy chcieli testować na emulatorze Smartphone'a. W tym celu w Visual Studio z menu *Debug* wybieramy opcję *Start* lub *Start Without Debugging*. Jeśli zostaniemy poproszeni o wybór urządzenia, na którym chcemy uruchomić aplikację, wskazujemy *Smartphone Emulator (Virtual Radio)*. Uruchamia się emulator i ładuje się nasza aplikacja. Proces ten zajmuje trochę czasu, należy więc cierpliwie poczekać. Po załadowaniu aplikacji możemy już z niej korzystać.

Podczas testowania natknąłem się na dwa problemy:

Po pierwsze, na słabszym komputerze uruchamianie aplikacji poprzez opcję *Debug -> Start* nie działa – nie zostaje ona załadowana. Prawdopodobnie jest to spowodowane zbyt małą ilością pamięci – wystarczy uruchamiać aplikację poprzez opcję *Start without debugging* i wszystko działa już dobrze.

Drugim problemem jest to, że emulator po wyłączeniu nie zapamiętuje utworzonych przez nas plików. Jeśli więc nie chcemy przy każdym uruchamianiu aplikacji łączyć się z serwerem w celu pobrania jakichkolwiek wiadomości, wystarczy zakończyć aplikację, ale nie wyłączać emulatora. Jest to również bardzo pomocne przy testowaniu, gdyż znacząco przyspiesza uruchamianie aplikacji (jednak poprzednie korzystanie z niej nie mogło zakończyć się błędem – w tym przypadku musimy ponownie uruchomić emulator).

Podsumowanie

Stworzona przeze mnie aplikacja nie jest szczególnie funkcjonalna. Działa na zaszytym w kodzie koncie, pozwala jedynie na pobieranie i zapisywanie wiadomości, itd. Założeniem nie było jednak stworzenie w pełni funkcjonalnego klienta poczty, ale pokazanie czytelnikowi podstaw tworzenia aplikacji na urządzenia przenośne. Z moich doświadczeń wynika, że tworzenia takiej aplikacji nie jest bardzo skomplikowane i nie różni się znacząco od tworzenia innego typu aplikacji w technologii .NET. Właściwie najwięcej ograniczeń narzuconych jest na interfejs użytkownika, należy również pamiętać o tym, że nie mamy dostępu do wszystkich klas. Prezentowany kod powinien jednak przekonać czytelnika, że nawet przy tych ograniczeniach tworzenie w miarę skomplikowanych aplikacji na urządzenia przenośne jest stosunkowo proste.

Aby uruchomić dołączoną do programu aplikację, należy w kodzie programu (w klasie Klient) przy wywołaniu *new Skrzynka*, zmienić nazwę konta pocztowego, z którego chcemy korzystać – taki wybór nie został uwzględniony w aplikacji, aby pozostawić ją możliwie nieskomplikowaną i dać czytelnikowi możliwość zabawy z nią w celach edukacyjnych (i innych) (.

Artur Gniadzik

Drukuj Zamknij