

DotGadu Tutorial v2008_08_25

Dominik Cebula

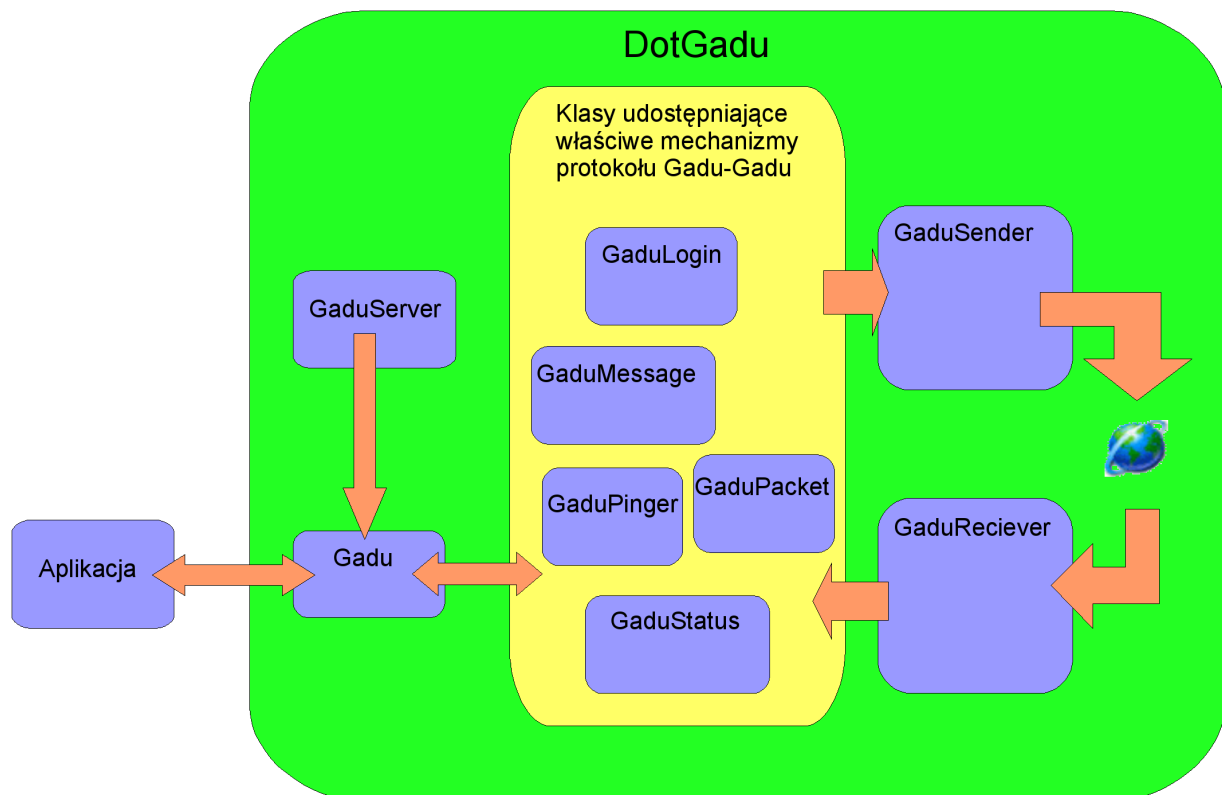
- 1. Organizacja biblioteki i przyjęte koncepcje.**
- 2. Ustawienia IDE:**
 - a) Visual Studio 2005
 - b) MonoDevelop
- 3. Łączenie z serwerem Gadu-Gadu i logowanie.**
- 4. Wysyłanie i odbieranie wiadomości.**
- 5. Zmiana statusu.**
- 6. Pobieranie tokenu.**
- 7. Rejestrowanie konta.**
- 8. Usuwanie konta.**
- 9. Zmiana hasła.**
- 10. Przypomnienie hasła na e-mail.**
- 11. Obsługa eventu OnGaduCriticalError.**
- 12. Powiadomienia o statusie użytkowników.**
- 13. Lista kontaktów.**
- 14. Katalog publiczny.**
- 15. Autor.**

1. Organizacja biblioteki i przyjęte koncepcje.

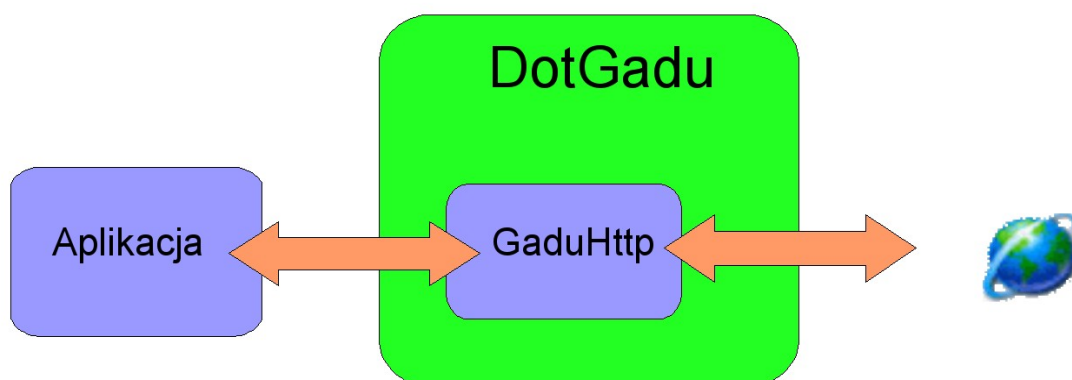
Z punktu widzenia użytkownika najważniejszą klasą jest klasa Gadu. Udostępnia ona metody, oraz właściwości, które zarządzają resztą klas stworzonych na potrzeby projektu. Podczas wywołania jakiejś funkcji, np. `sendMessage()` służącej do wysyłania wiadomości, tworzony jest nowy obiekt `GaduMessage`, który odpowiada za faktyczną obsługę mechanizmu wysyłania wiadomości.

Wysyłaniem i odbieraniem danych z gniazda zajmują się klasy `GaduSender` oraz `GaduReceiver`, tworzone wewnątrz klasy `Gadu`. Obie te klasy posiadają swój własny wątek, który jest niszczone razem z zakończeniem połączenia z Gadu-Gadu, poprzez wywołanie metody `Disconnect()` z klasy `Gadu`. `GaduSender` kolejkuje wszystkie pakiety jakie mają zostać wysłane do Gadu-Gadu. Obie klasy udostępniają eventy, które używa się do odbierania powiadomień z biblioteki. Dostać się do klas `GaduSender` oraz `GaduReceiver` możemy z poziomu klasy `Gadu`, referencję do tych obiektów zwracają właściwości `Gadu.Receiver` oraz `Gadu.Sender`.

Koncepcyjny schemat biblioteki można przedstawić w następujący sposób:

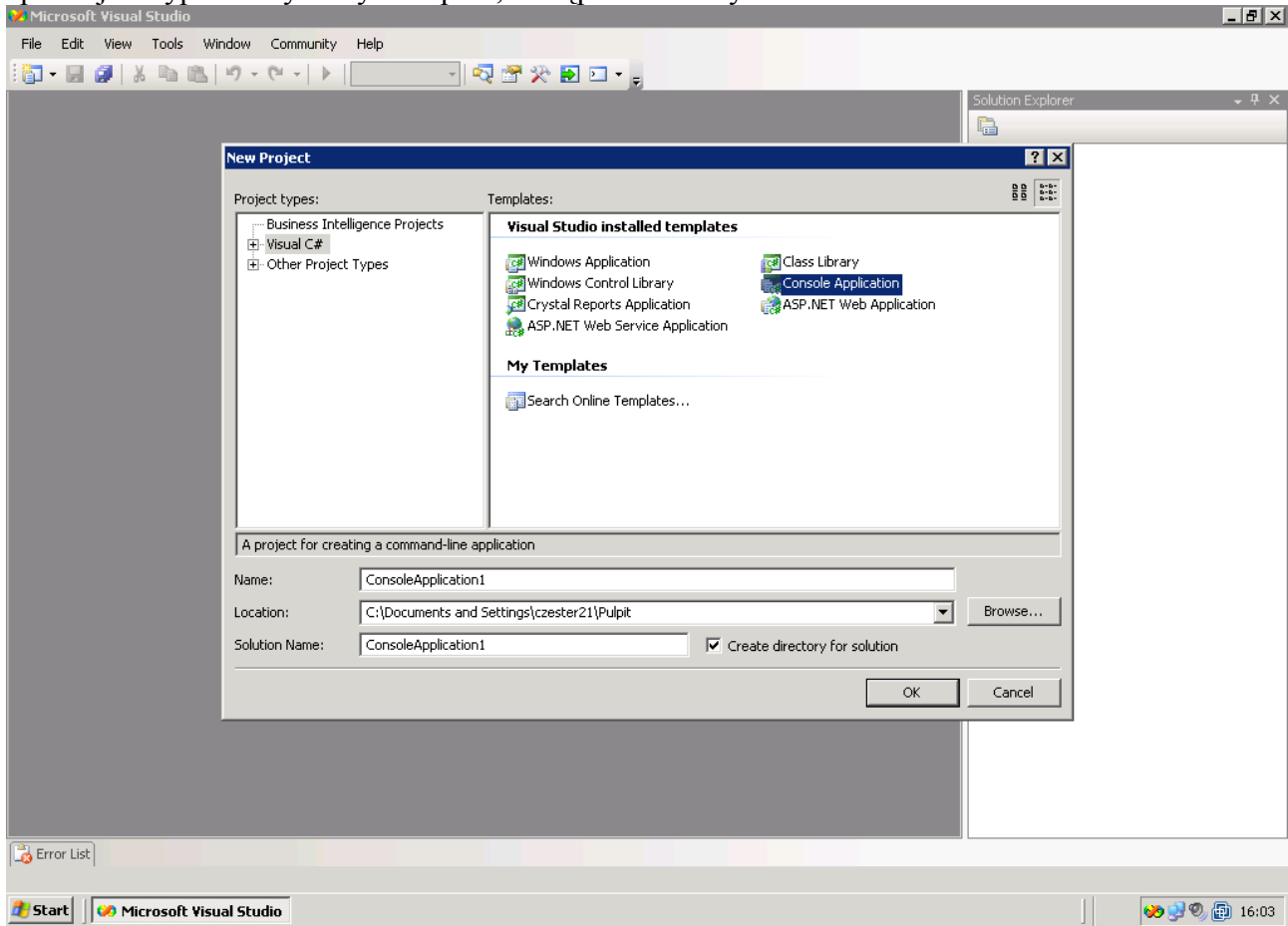


W przypadku usług HTTP:

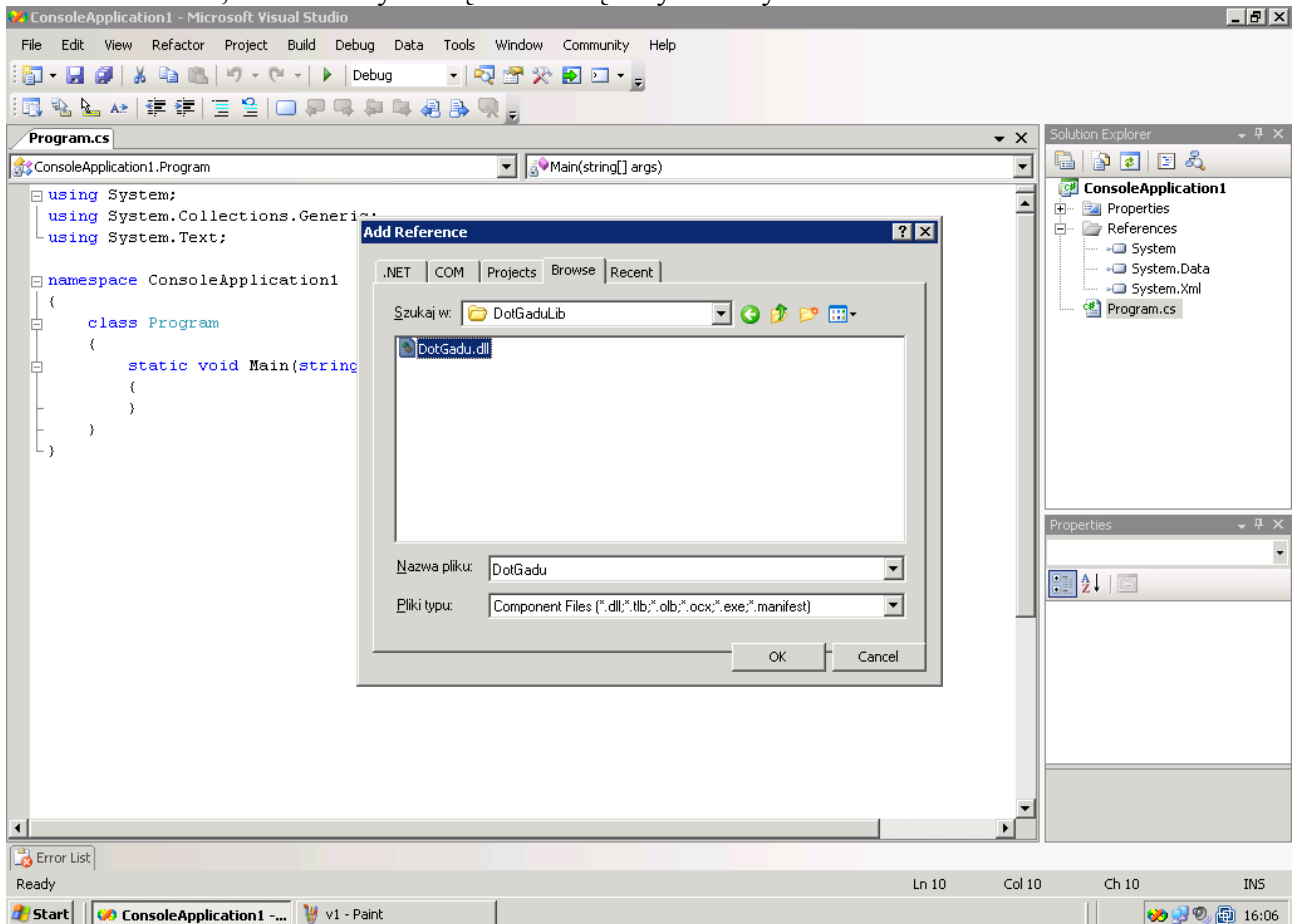


2a. Ustawienia IDE: Visual Studio 2005

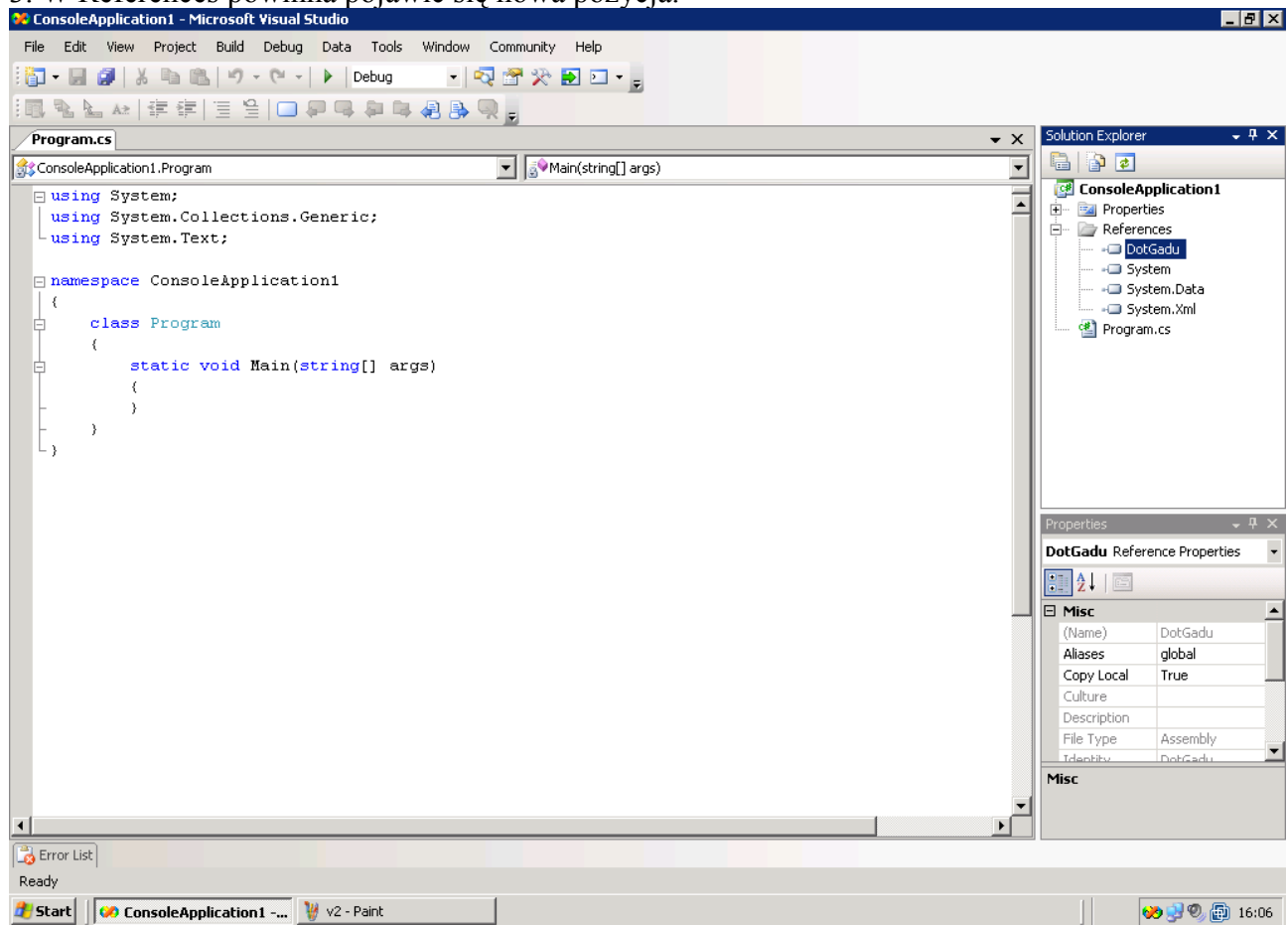
1. W celu utworzenia nowego projektu wybieramy File->New->Project..., wybieramy język, typ aplikacji i wypełniamy wszystkie pola, następnie klikamy ok.



2. Musimy dodać referencję do DotGadu, klikamy Project->Add Referencje..., przechodzimy do zakładki Browse, zaznaczamy naszą bibliotekę i wybieramy OK.



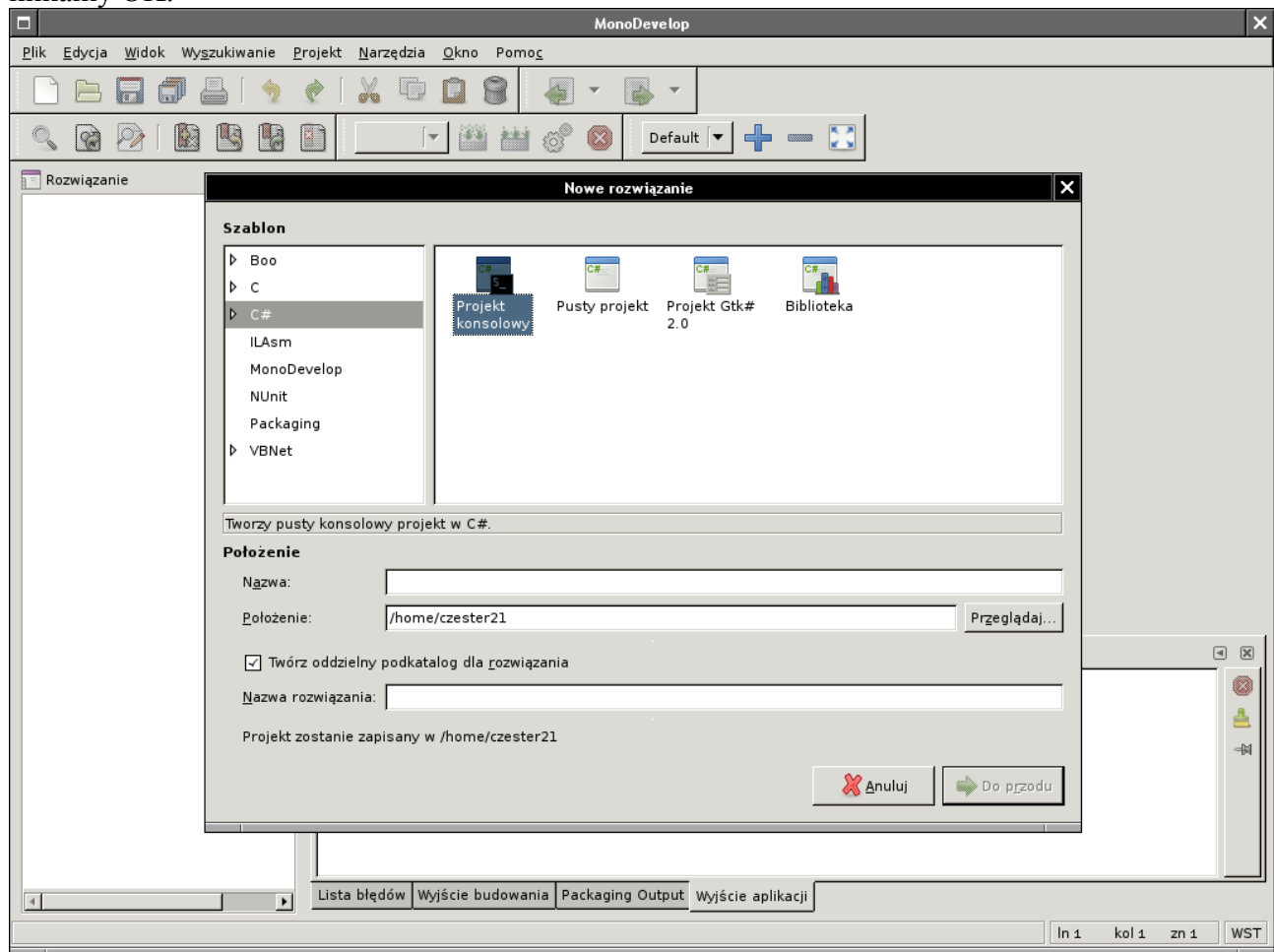
3. W References powinna pojawić się nowa pozycja.



4. Projekt jest już ustawiony do pracy z biblioteką, proponuję przekompilować plik `example1.cs` dla sprawdzenia.

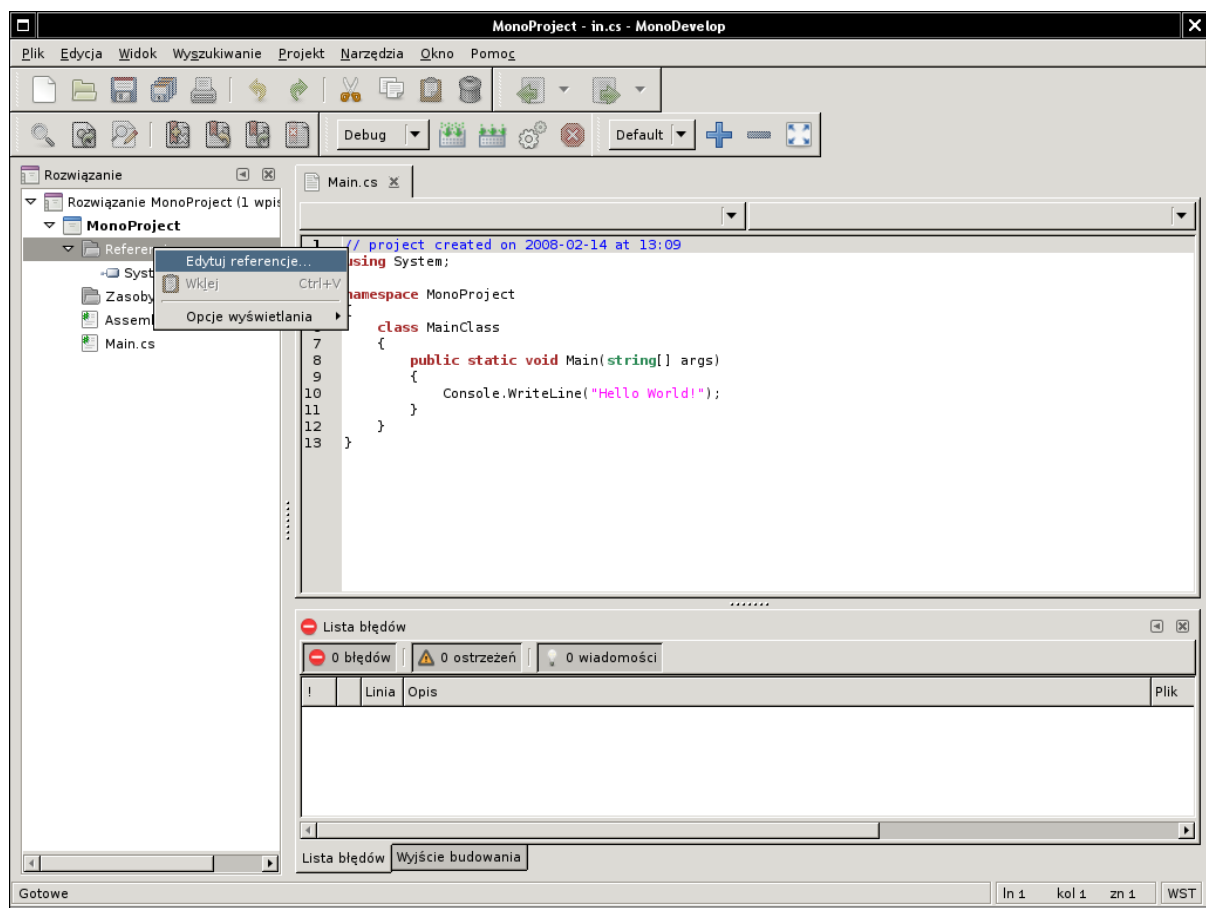
2b. Ustawienia IDE: MonoDevelop

1. Wybieramy opcję Plik->Nowy Projekt, następnie zaznaczamy język C#, wybieramy projekt konsolowy lub Gtk#. Wypełniamy nazwę projektu, klikamy „Do Przodu”, w następnym oknie klikamy OK.

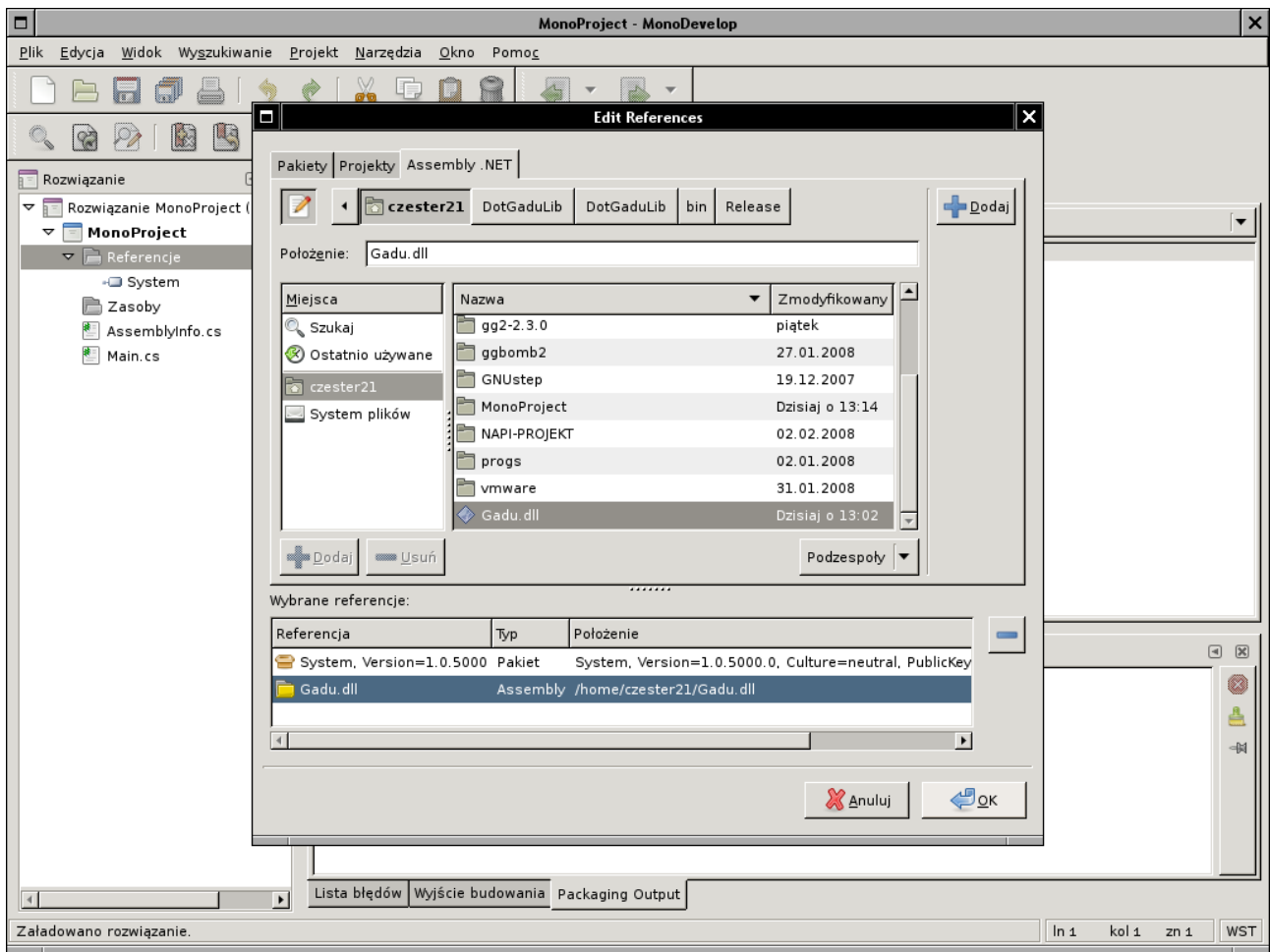


2. Wybieramy Widok->Rozwiązanie.

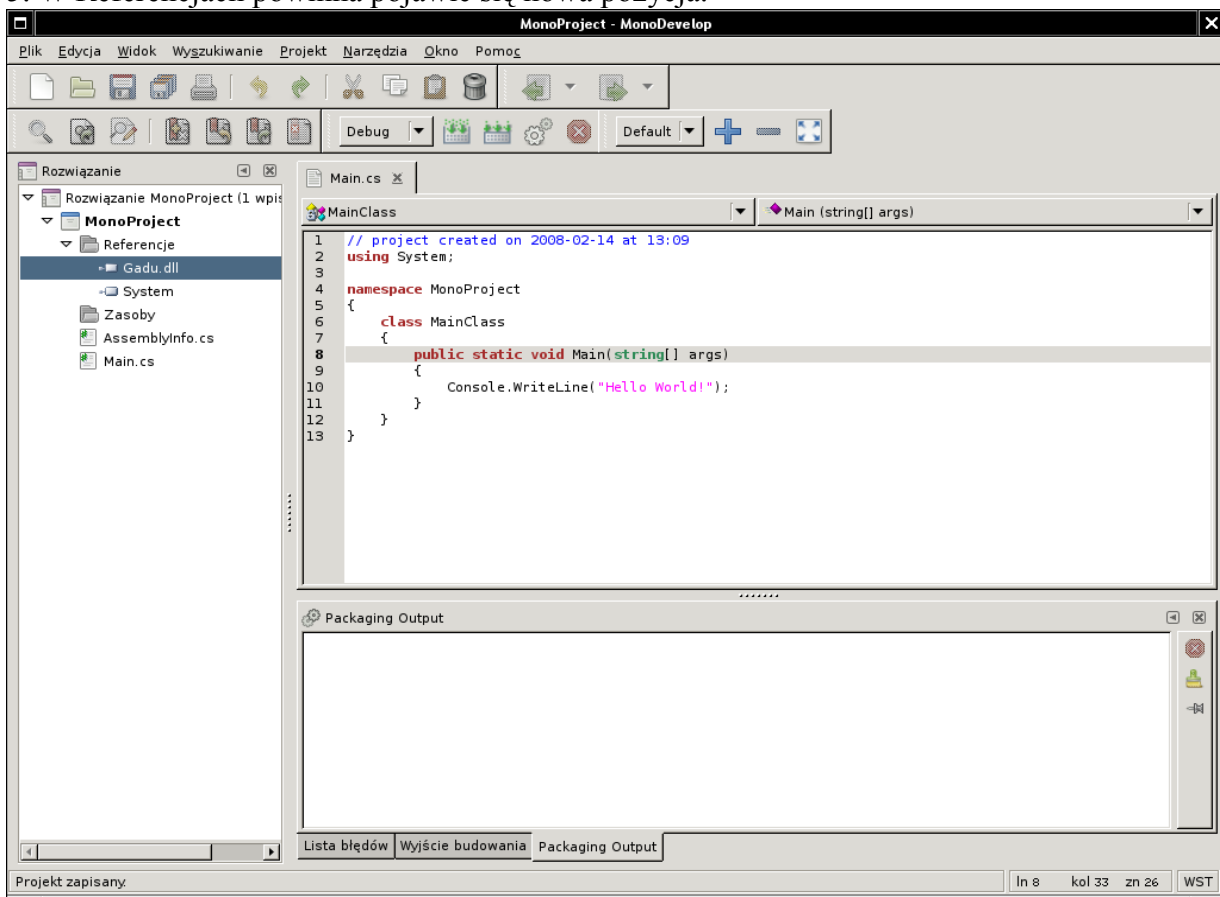
3. Klikamy prawym przyciskiem na Referencje, wybieramy „Edytuj Referencje”.



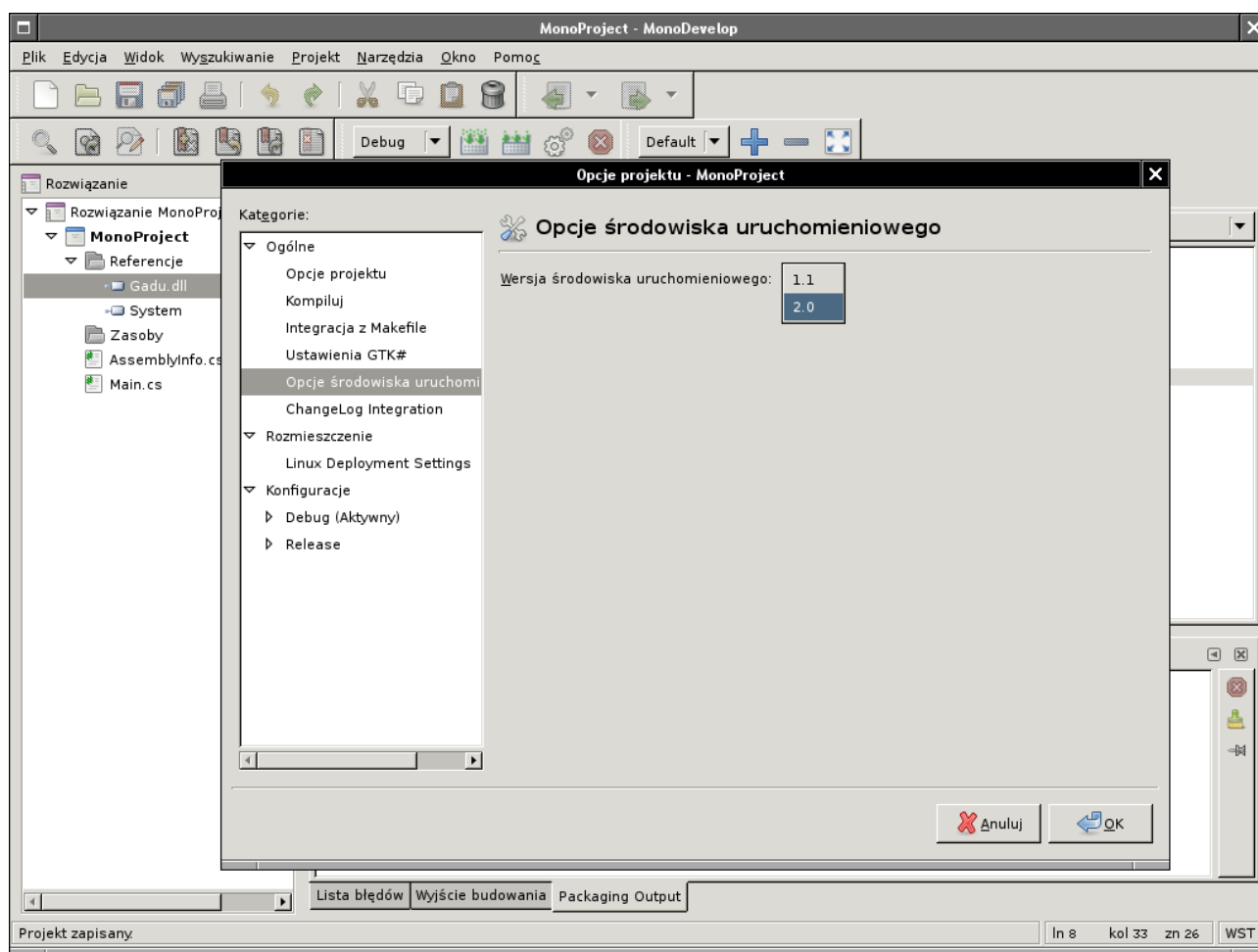
4. Przechodzimy do zakładki „Assembly .NET”, wybieramy naszą bibliotekę DLL i klikamy Dodaj. W dolnej części okna „Wybrane Referencje” powinna znajdować się nasza biblioteka DotGadu.



5. W Referencjach powinna pojawić się nowa pozycja.



6. Ponieważ część biblioteki DotGadu jest napisana w .NET 2.0 musimy zmienić środowisko uruchomieniowe na .NET 2.0, wybieramy Projekt->Opcje, w „Kategorie” wybieramy „Opcje środowiska uruchomieniowego” i zmieniamy „Wersja środowiska uruchomieniowego” na 2.0 i klikamy OK.



7. Projekt jest już ustawiony do pracy z biblioteką, proponuję przekompilować plik example1.cs dla sprawdzenia.

3. Łączenie z serwerem Gadu-Gadu i logowanie.

Jeżeli już dodaliśmy referencję do pliku DLL w naszym projekcie możemy przystąpić do pisania kodu. Pierwszą rzeczą, którą trzeba zrobić to dodanie przestrzeni nazw DotGadu. Wpisujemy więc:

```
using DotGadu;
```

Teraz już możemy używać wszystkich obiektów udostępnionych do obsługi Gadu-Gadu. Zanim połączymy się z serwerem trzeba pobrać jego adres ip oraz port z appmsg.gadu-gadu.pl, możemy również ustawić adres ip oraz port serwera ręcznie. Do obu tych operacji posłuży klasa GaduServer. Pobieranie adresu automatycznie realizujemy w ten sposób:

```
GaduServer gaduServer = new GaduServer(179824);
```

GaduServer posiada dwa konstruktory, pierwszy przyjmuje jako parametr numer gg, na który zamierzamy się zalogować, ten konstruktor również automatycznie pobierze adres serwera gg. Drugi konstruktor przyjmuje jako parametry adres ip oraz port:

```
GaduServer gaduServer = new GaduServer("217.17.45.147", 8074);
```

Kiedy mamy już adres naszego serwera, możemy połączyć się z Gadu-Gadu. Użyjemy klasy Gadu oraz metody Connect(), do niej podamy referencje do GaduServer:

```
Gadu gadu=new Gadu();  
gadu.Connect(gaduServer);
```

W tym momencie musimy wiedzieć, że metoda Connect() utworzy klasy GaduSender, GaduReceiver, GaduPinger, a w każdej z nich wątek, dlatego bardzo ważne jest aby wywołać metodę Disconnect() podczas, kończenia działania programu.

```
gadu.Disconnect();
```

Mamy już nawiązane połączenie, możemy przejść do logowania się do serwera Gadu-Gadu. W klasie Gadu mamy funkcję Login(), która przyjmuje jako parametry numer gg, oraz nasze hasło:

```
gadu.Login(179824, "password");
```

Pamiętajmy, również o tym aby się wylogować:

```
gadu.Logout();
```

Teraz zajmiemy się odbieraniem informacji z samej biblioteki DotGadu, takich jak powiadomienie o wysłanym pakiecie, poprawnym lub niepoprawnym logowaniu. W klasie Gadu mamy trzy właściwości, zwracające referencje do obiektów, które posiadają ważne dla nas evenety. Są to:

- gadu.Sender
- gadu.Receiver
- gadu.Pinger

GaduSender - klasa zajmująca się wysyłaniem pakietów, GaduReceiver - klasa zajmująca się odbieraniem pakietów, GaduPinger klasa zajmująca się pingowaniem serwera Gadu-Gadu. Najpierw podłożymy delegację pod event informujący nas o tym, że pakiet został wysłany do serwera Gadu-Gadu. Tworzymy w tym celu funkcję informującą o wysłaniu pakietu:

```
public static void packet(IGaduPacket packet)  
{  
    Console.WriteLine("Packet sended...");  
}
```

Funkcja musi mieć parametr IGaduPacket, aby pasowała do delegacji. Teraz przypiszemy funkcję do eventu w GaduSender:

```
gadu.Sender.OnPacketSended+=new OnPacketSendedHandler(packet);
```


Za każdym razem kiedy GaduSender wyśle jakiś pakiet nasza funkcje packet() zostanie wywołana. Następnie stworzymy funkcje informujące o tym czy logowanie zakończyło się sukcesem czy porażką:

```
public static void loginok()
{
    Console.WriteLine("Logowanie OK");
}

public static void loginfailed()
{
    Console.WriteLine("Logowanie zakonczylo sie niepowodzeniem");
}

public static void loginneedemail()
{
    Console.WriteLine("Trzeba uzupełnic email na koncie gg");
}
```

Musimy jeszcze zadbać o to aby funkcje były wywoływane przez GaduReciever:

```
gadu.Reciever.OnLoginOK+=new OnLoginOKHandler(loginok);
gadu.Reciever.OnLoginFailed+=new OnLoginFailedHandler(loginfailed);
gadu.Reciever.OnLoginNeedEmail+=new OnLoginNeedEmailHandler(loginneedemail);
```

Proponuję jeszcze dodać przed wylogowaniem się z konta i rozłączeniem z Gadu-Gadu:

```
System.Threading.Thread.Sleep(5000);
```

Dzięki temu program poczeka zanim wyloguje się i zakończy połączenie z Gadu-Gadu. Musisz wiedzieć, że DotGadu działa całkowicie asynchronicznie, oznacza to, że wywołanie jakiejś metody, która np. wysyła wiadomość nie powoduje „przetrzymywania” programu aż wiadomość na pewno zostanie wysłana, pakiet z wiadomością zostanie dodany do GaduSender i zajmie się nią odpowiedzialny za to wątek. Dlatego bez tej liniki, która „poczeka” trochę przed wywołaniem gadu.Disconnect(), wszystkie wątki zostaną bezwzględnie zakończone, przez co biblioteka nie zdąży się załogować do Gadu-Gadu.

Zobacz przykładowy program example1.cs

4. Wysyłanie i odbieranie wiadomości.

Wiadomości wysyłane są przy użyciu metody sendMessage() z klasy Gadu, a odbierane są przez dopisanie funkcji obsługującej odpowiedni event w GaduReciever. Jeżeli jesteśmy już podłączeni do Gadu-Gadu oraz jesteśmy załogowani, wysyłamy do kogoś wiadomość w następujący sposób:

```
gadu.sendMessage(3837462,"Witam, wlasnie testuje DotGadu :-)");
```

Linika ta w przykładzie drugim została umieszczona w funkcji loginok() aby wysłać wiadomość tylko kiedy jesteśmy poprawnie załogowani.

Odbieranie wiadomości będziemy realizować za pomocą funkcji RecvMsg() podpiętej pod odpowiedni event w GaduReciever. Do funkcji dodamy również kod, które spowoduje zakończenie działania naszej aplikacji w momencie kiedy, ktoś nam przysła wiadomość o treści „exit”:

```
public static void RecvMsg(GaduPacketRecieveMessage msg)
{
    Console.WriteLine("Wiadomosc od "+msg.Sender+": "+msg.Message);
    if (msg.Message=="exit\0")
        exit=true;
}
```

Musimy jeszcze podpiąć ją do odpowiedniego eventu:

```
gadu.Reciever.OnRecieveMessage+=new OnRecieveMessageHandler(RecvMsg);
```

Stworzymy również funkcję, która poinformuje nas o odebraniu pakietu potwierdzającego doręczenie wiadomości.

```
public static void MsgAck(GaduPacketMessageAck ack)
{
    Console.WriteLine("Odebralem pakiet mowiacy o statusie wiadmosci");
    if (ack.Status==GaduPacketConstans.GG_ACK_BLOCKED)
        Console.WriteLine("Wiadomosc zablokowana");
    else if (ack.Status==GaduPacketConstans.GG_ACK_DELIVERED)
        Console.WriteLine("Wiadomosc dostarczono");
    else if (ack.Status==GaduPacketConstans.GG_ACK_QUEUED)
        Console.WriteLine("Wiadomosc zakoljekowano");
    else if (ack.Status==GaduPacketConstans.GG_ACK_MBOXFULL)
        Console.WriteLine("Odbiorca ma pelna skrzynke odbiorcza");
    else if (ack.Status==GaduPacketConstans.GG_ACK_NOT_DELIVERED)
        Console.WriteLine("Wiadomosc nie dostarczona");
}
```

Pozostaje podpiąć ją pod event:

```
gadu.Reciever.OnRecieveMessageAck+=new OnRecieveMessageAckHandler(MsgAck);
```

W przykładzie wiadomość zaimplementowana jest również funkcja loginfailed(), która ustawia exit na true, tym samym kończy działanie programu.

Zobacz przykładowy program example2.cs

5. Zmiana statusu.

Zmiany statusu dokonujemy przez wywołanie funkcji changeStatus() z klasy Gadu. Funkcja przyjmuje dwa parametry, jest to statusid oraz opis, który ustawiamy na pusty string, jeżeli wybraliśmy status, który nie umożliwia ustawienie opisu. Poniżej znajdują się przykłady użycia funkcji changeStatus():

```
gadu.changeStatus(GaduPacketConstans.GG_STATUS_AVAIL,String.Empty);
gadu.changeStatus(GaduPacketConstans.GG_STATUS_AVAIL_DESCR,"Dostepny z opisem");
gadu.changeStatus(GaduPacketConstans.GG_STATUS_BUSY,String.Empty);
gadu.changeStatus(GaduPacketConstans.GG_STATUS_BUSY_DESCR,"Zajety z opisem");
gadu.changeStatus(GaduPacketConstans.GG_STATUS_INVISIBLE,String.Empty);
gadu.changeStatus(GaduPacketConstans.GG_STATUS_INVISIBLE_DESCR,"Nieiwdzialny z opisem");
```

W przykładzie trzecim, program łączy się z gg, jeżeli zaloguje się poprawnie co 5 sec. zmienia status:

Zobacz przykładowy program example3.cs

6. Pobieranie tokenu.

Wszystkie operacje dotyczące usług HTTP realizowane są za pomocą klasy GaduHttp. Do wszystkich zmian na naszym koncie potrzebny jest token, a więc przed każdą operacją musimy go pobrać, pokazać użytkownikowi, następnie użytkownik powinien podać odczytany ciąg znaków z tokenu. Pobieranie tokenu realizujemy za pomocą metody getToken() z klasy GaduHttp. Metoda ta przyjmuje dwa parametry, są to referencje wyjściowe, pierwsza wpisuje id tokena, druga obraz tokenu.

Najpierw musimy utworzyć sobie dwie zmienne:

```
String tokenId;
Image token;
```

Referencje te podamy do metody getToken() w następujący sposób:

```
gaduHttp.getToken(out tokenId, out token);
```

Po wykonaniu w zmiennej tokenId powinniśmy mieć id tokenu, w zmiennej token obraz reprezentujący token.

Zobacz przykładowy program example4.cs

7. Rejestrowanie konta.

W celu zarejestrowania konta należy użyć funkcji registerAccount() z klasy GaduHttp. Funkcja jako parametry przyjmuje kolejno: hasło, e-mail, id pobranego tokenu, wartość pobranego tokenu. Jeżeli rejestracja się powiedzie zostanie zwrócony numer gadu-gadu naszego nowego konta.

```
uin=gaduHttp.registerAccount("passwd", "mail@mail.com", tokenId, tokenval);
```

W przypadku niepowodzenia zostanie zwrócony wyjątek GaduRegisterException.
Pełny kod:

Zobacz przykładowy program example5.cs

8. Usuwanie konta.

Realizowane przez deleteAccount() z GaduHttp. Musimy podać numer konta do usunięcia, hasło, id tokenu oraz wartość tokenu. W przypadku niepowodzenia zostanie zwrócony GaduDeleteException.

Zobacz przykładowy program example6.cs

9. Zmiana hasła.

Zmianę hasła realizujemy poprzez metodę changePassword(), jako parametry przyjmuje: numer gg, e-mail, stare hasło, nowe hasło, id tokenu, wartość tokenu.

Zobacz przykładowy program example7.cs

10. Przypomnienie hasła na e-mail.

Wywołujemy funkcję remindPassword() podając: numer gg, id tokenu, wartość tokenu.

Zobacz przykładowy program example8.cs

11. Obsługa eventu OnGaduCriticalError.

Event OnGaduCriticalError jest kluczowy jeśli chcemy stworzyć stabilną aplikację opartą o DotGadu. Jest to statyczny Event znajdujący się w klasie Gadu, który zostaje wywołany przez GaduSender lub GaduReceiver w przypadku jakiegokolwiek błędu, np. związanego z samym gniazdem. W kodzie obsługującym event powinniśmy podać schemat działania na wypadek błędu, np. procedurę rozłączenia się i ponownego połączenia oraz logowania, pamiętamy przy tym o obsłudze wyjątków. Za przykład niech posłuży prosty bot napisany w DotGadu:

Zobacz przykładowy program example9.cs

12. Powiadomienia o statusie użytkowników.

Klasa GaduNotifier udostępnia mechanizmy dzięki, którym możemy sprawdzić status użytkowników naszej listy. Dziedziczy ona po klasie List, a więc możemy traktować ją jako listę. Klasa posiada metodę Add, która służy do kompletowania listy użytkowników, których statusem jesteśmy zainteresowani. Jeżeli skończyliśmy już kompletować listę użytkowników wywołujemy metodę sendNotify aby wysłać do gg stosowany pakiet. GaduReceiver powinien przechwycić odpowiedni pakiet i wywołać event. Możemy również dodawać i usuwać listę użytkowników o, których statusach będziemy informować po przesłaniu całej listy: metody addNotify i delNotify.

Zobacz przykładowy program example10.cs

13. Lista kontaktów.

GaduUserList umożliwia wysyłanie oraz pobieranie listy kontaktów z serwera gg. Dziedziczy ona po klasie List, a więc możemy dodawać i usuwać z niej użytkowników. Jeżeli mamy już komplet wywołujemy metodę sendAll. Jeżeli chcemy pobrać listę kontaktów z serwera wywołujemy metodę getListFromServer. Pobieranie listy realizujemy poprzez odpowiedni event GaduReceiver. Server zwróci pakiet, którego zawartość możemy interpretować za pomocą klasy GaduUser i metody ParseUserListString.

Zobacz przykładowy program `example11.cs`

14. Katalog publiczny.

GaduPubDir umożliwia odczyt naszych danych z katalogu publicznego, zapis naszych danych oraz szukanie osób w katalogu publicznym. W celu odczytania naszych danych trzeba utworzyć obiekt klasy GaduUser i wypełnić tylko pole Uin, następnie przekazać referencje do tego obiektu do metody Read z klasy GaduUser. Server prześle nam stosowny pakiet, który zostanie przechwycony przez GaduReceiver. Zapisywanie danych do katalogu wygląda bardzo podobnie, musimy jednak wypełnić klasę GaduUser. W celu wyszukania użytkowników klasa GaduUser pełni rolę filtra, podajemy ją do metody Search. Oto przykład prezentujący wyszukiwanie użytkowników o imieniu Tomek.

Zobacz przykładowy program `example12.cs`

15. Autor.

Dominik Cebula

gg: 3837462

email: dominikcebula@gmail.com