Björn Persson

Använda Visual Studio.NET 2003 och SharpDevelop 1.0

Applikationsutveckling

juni 2007

Om denna sammanfattning

Avsikten med denna sammanfattning är att beskriva hur man använder Visual Studio.NET 2003 (VS.NET) samt vilka olika typer av projekt det finns i VS.NET. I sammanfattningen beskrivs även hur man använder gratismiljön SharpDevelop 1.0 (eller #develop).

Koden i denna sammanfattning har skrivits i Microsofts *Visual Studio.NET 2003* (i Windows XP) och sedan kopierats in i dokumentet. Fel kan dock ha införts av misstag vid redigering av dokumentet. Övrig programvara från Microsoft som använts är Access 2003.

Konventioner i sammanfattning

Kod har skrivits med typsnitt av fast bredd (`Courier New`) för att göras mer lättläst samt längre exempel har inneslutits i en ram (se exempel nedan).
I Visual Basic kan programsatser (*statements*) skrivas på flera rader genom att använda understrykningstecknet (”_”). Detta underlättar läsning av kod då man slipper skrolla i sidled för att läsa längre programsatser. Viktigt är att placera ett mellanslag innan understrykningstecknet som i detta exempel:

```
enVariabel = 1 _
             + 2    'Kommentarer i kod skriv med fet stil
```

Spara papper! Skriv inte ut sammanfattningen utan ladda ner PDF.

Jag är givetvis tacksam för alla konstruktiva synpunkter på sammanfattningens utformning och innehåll.

Eskilstuna, juni 2007

Björn Persson

E-post: (se startsida på min webbplats)
Personlig hemsida: `http://www.kiltedviking.net/`

Innehållsförteckning

1 INLEDNING 4

2 ANVÄNDA VISUAL STUDIO.NET 5
2.1 Fönster och delar i Visual Studio.NET 5
2.2 Lösningar och projekt 6
2.3 Kompilera, bygga och köra 7
2.4 Funktioner och inställningar i VS.NET 7
2.4.1 Visa radnummer i kodfönster [ATT GÖRA] 7
2.4.2 Dölja/visa kod och regioner 7
2.4.3 Avancerat: Egna delade *assemblies* i dialogrutan Add Reference i VS.NET 8

3 PROJEKT I VISUAL STUDIO.NET 9
3.1 Att tänkta på… 10
3.1.1 Använd inte svenska tecken (å, ä eller ö) i projektnamn 10
3.1.2 Fundera på projektnamn (och andra namn) innan projekt skapas 10
3.1.3 Säkerhetsproblem med placering av projektfiler 11
3.2 Windows Application 11
3.2.1 Filer i projekt och referenser 11
3.2.2 VB6 kontra VS.NET 11
3.2.3 Lägga till fler projekt i lösning 11
3.3 Class Library 12
3.3.1 Filer i projekt och referenser 12
3.3.2 Lägga till fler projekt i lösning 12
3.3.3 Använda klassbibliotek i andra projekt 12
3.3.4 Privata och delade *assemblies* 12
3.4 Console Application [ATT GÖRA] 13
3.5 ASP.NET Web Application [ATT GÖRA] 13
3.6 ASP.NET Web Service [ATT GÖRA] 13

4 ANVÄNDA SHARPDEVELOP 14
4.1 Lösningar och projekt 15
4.2 Kompilera, bygga och köra 15

5 PROJEKT I SHARPDEVELOP 17
5.1 Att tänkta på… 18
5.1.1 Använd inte svenska tecken (å, ä eller ö) i projektnamn 18
5.1.2 Fundera på projektnamn (och andra namn) innan projekt skapas 18
5.1.3 Säkerhetsproblem med placering av projektfiler 18
5.2 *Console Application [ATT GÖRA]* 19
5.2.1 Filer i projekt och referenser 19
5.2.2 En enkel konsolapplikation 19
5.3 *Windows Application* 19
5.3.1 Filer i projekt och referenser [GÖR OM BILDER] 19
5.3.2 Lägga till fler projekt i lösning 20
5.4 *Class Library* 20
5.4.1 Filer i projekt och referenser 20
5.4.2 Lägga till fler projekt i lösning 20
5.4.3 Använda klassbibliotek i andra projekt 20
5.4.4 Privata och delade *assemblies* 21

6 FLER VERKTYG AV INTRESSE 21

Att göra:

* Förklara bin-mapp och dess undermappar.

* Förklara skillnad mellan Debug och Release.

* Förklara relationer mellan projekt, assemblies och namnutrymmen.

1 Inledning

Visual Studio.NET (VS.NET) skiljer sig en aning från Visual Studio 6 (förutom utseendemässigt ☺). En av dom största skillnaderna är att alla programmeringsspråk i .NET använder samma utvecklingsmiljö (*integrated development enviroment*, IDE). I denna sammanfattning kommer projekt i VS.NET beskrivas samt även jämförelse göras mellan projekt i Visual Basic 6 (VB6) och VS.NET.

Ett gratisalternativ till Visual Studio.NET är #develop (eller *SharpDevelop*). Som namnet visar så är SharpDevelop främst tänkt att användas för att utveckla projekt i C#, men även andra .NET-språk kan användas (dock inte med lika bra funktion i WYSIWYG[1]-editor för grafiska gränssnitt i övriga språk).

För att utveckla och exekvera .NET-applikationer så måste *.NET Framework* vara installerad. Detta har beskrivits i sammanfattningen *Installera IIS, .NET Framework 1.1, databasklienter och verktyg*. För att kunna skapa webbprojekt så behöver vi tillgång till en webbserver – antingen på samma dator som vi utvecklar projektet eller på en annan dator (en server). Detta finns också beskrivet i sammanfattningen *Installera IIS, .NET Framework 1.1, databasklienter och verktyg*. För en beskrivning av språket Visual Basic.NET (VB.NET), se sammanfattningen *Visual Basic.NET för komponenter*.

I nästa kapitel beskrivs miljön i VS.NET och i kapitel därefter de vanligaste typerna av projekt i VS.NET. Miljön i SharpDevelop beskrivs i det fjärde kapitlet.

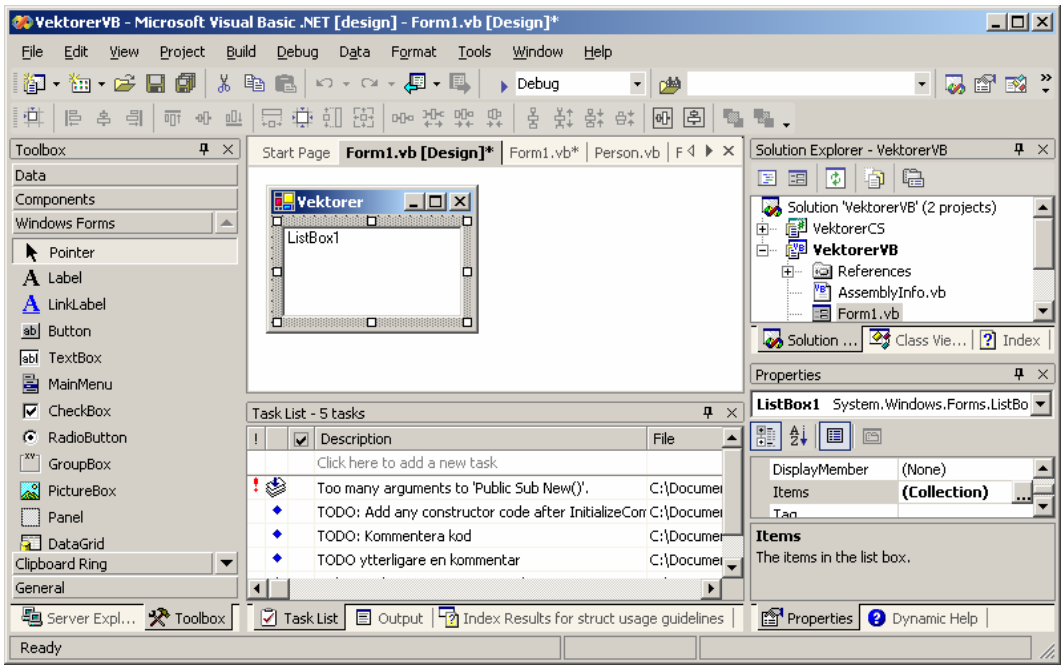
[1] *What-You-See-Is-What-You-Get*.

2 Använda Visual Studio.NET

I detta kapitel beskrivs grundläggande saker om programmeringsmiljön (*Integrated Development Enviroment*, IDE:n) Visual Studio.NET (VS.NET) från Microsoft. I senare kapitel beskrivs gratismiljön SharpDevelop (inte från Microsoft ☺).

2.1 Fönster och delar i Visual Studio.NET

Denna beskrivning har gjorts med VS.NET 2003[2], med vilken .NET Framework 1.1[3] installeras. Nedan visas en bild med VS.NET:s ”huvudfönster”, d.v.s. själva miljön.



Nedan beskrivs de vanligaste delarna av (fönstren i) VS.NET (från vänster till höger och uppifrån och ner). De flesta fönstren kan innehålla flera saker, vilket löses genom att placera innehållet på flikar (oftast längst ner i fönstret). Vissa fönster visas inte alltid och fönster kan flyttas runt (dockade eller odockade), vilket gör att IDE:ns utseende (layout) kan variera. Men i bild ovan visas hur layout brukar ta sig ut efter installation med en lösning öppen.

- **Verktygsfönster** (*Toolbox*) – innehåller kontroller för att designa grafiska gränssnitt. Kontrollerna har placerats i ”rullgardiner” för att kunna ordna dem i mindre (logiska) grupper. Detta fönster innehåller även **serverutforskaren** (*Server Explorer*, vilket inte förklaras i denna sammanfattning).
- **Designfönstret** – där formulär för grafiska gränssnitt byggs upp, ett fönster där flikar av någon anledning placerats längst upp i fönstret. En andra funktion för detta fönster är **kodfönstret** där kod redigeras. Även sidor från hjälpen, m.m. visas här.
- **Lösningsfönster** (*Solution Explorer*) – här visas projekt som ingår i lösning, filer i projekt samt referenser till andra *assemblies*. En andra funktion för fönstret är att visa **klassvyn** (*Class View*) där klasser i lösning visas hierarkiskt efter projekt och namnutrymme (men **inte** efter arv!). Ytterligare två funktioner för fönstret är för

[2] … men beskrivning bör även fungera med version 1.0, eller 2002 som den också kallas.
[3] Med VS.NET 1.0 följer .NET Framework 1.0.

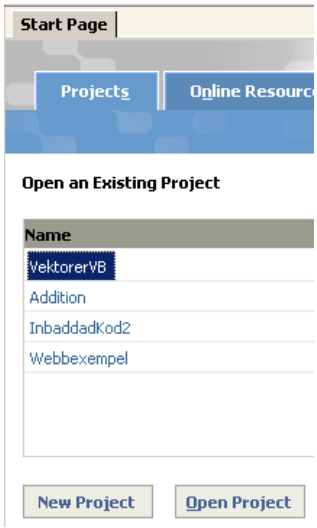
hjälpen där sökning kan ske via indexord och fritextsökning. Om sökning resulterar i flera alternativ så visas dessa i normalt i den nedre delen av VS.NET (se nästa punkt).

- I ”nedre raden” med fönster hittas **aktivitetsfönstret** (*Task List*) där kompileringsfel visas (röda utropstecknet i bild ovan) liksom att-göra-kommentarer (kommentarer som börjar med texten ”TODO”[4]). Här hittas även **resultatfönstret** (*Output*) om visar resultatet från kompileringar och exekveringar. En tredje funktion för fönstret är att visa **resultatet från sökningar** (*Index Results*) i hjälpen om sökning ger fler än ett resultat (om bara ett resultat så visas det direkt i design-/kodfönstret ☺).
- Det sista fönstret är **egenskapsfönstret** (*Properties*) som används för att ändra egenskaper för kontroller i ett formulär. Här finns även det **dynamiska hjälpfönstret** som visar olika innehåll (i form av länkar) beroende på vilket av ovan nämna fönster som har fokus. (Ett försök till intelligent hjälp. ☺)

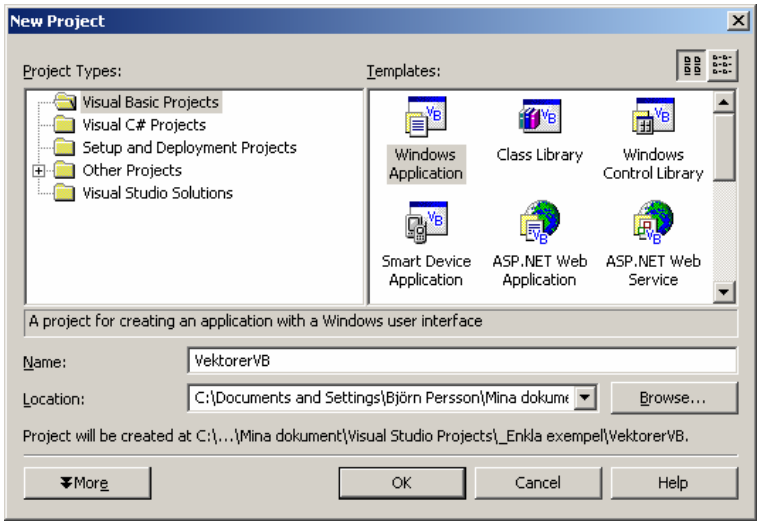
2.2 Lösningar och projekt

Ett **projekt** är en samling av filer som (oftast) kompileras till en binärfil (EXE- eller DLL-fil).

Lösningar (*solutions*) används för att samla flera projekt, t.ex. för att utveckla ett grafiskt gränssnitt (i EXE-fil) och komponenter (i DLL-filer). Detta innebär dock **inte** att en lösning måste innehålla flera projekt. När vi skapar ett projekt så skapas även en lösning (om vi inte lägger till ett nytt projekt i en existerande lösning).



För att skapa ett projekt kan vi antingen klicka på knappen New Project på Start Page (se bild till höger) eller välja File→New→Project…. Detta visar dialogrutan New Project (se bild nedan).



Om en lösning redan är öppen så visas även två radioknappar i dialogrutan New Projekt: Add to Solution och Close Solution. Markera den första för att lägga till det nya projektet i redan existerande (öppna) lösning och den andra för att skapa en ny lösning med det nya projektet i.

Som standard skapas nya projekt under mappen `Mina dokument\Visual Studio Projects`[5], vilket kan ändras varje gång vi skapar ett projekt eller sökväg ändras i VS.NET-inställningar[6]. En mapp – projektmappen – med samma namn som projektet skapas och alla

[4] Texten ”TODO” kan skrivas med gemener eller versaler, men genom att använda versaler så hittas den lättare i koden.
[5] I Ekonomihögskolans datorsalar så motsvaras `Mina dokument` av studenters hemkatalog, d.v.s. `H:\`.
[6] Välj Tools→Options samt markera mappen Enviroment och sen alternativet Projects and Solutions.

filer för projekt placeras i mappen (eller någon av dess undermappar). Nedan beskrivs typer av filer och vad de innehåller.

Filtyp	**Innehåll**
`*.vb` (eller `*.cs`)	Kodfiler. Filändelse visar, praktiskt nog, på språk i fil. ☺
`<projektnamn>.vbproj` (eller `<projektnamn>.csproj`)	Innehåller information om projekt, bl.a. filer som ingår i projekt.
`<projektnamn>.sln` och `<projektnamn>.suo`	Innehåller information om lösning, bl.a. projekt som ingår.
`AssemblyInfo.vb`	Fil med inställningar för projekt, bl.a. för komponenttjänster.

Programfiler (EXE- och DLL-filer) placeras i mappen `bin` under projektmappen.

2.3 Kompilera, bygga och köra

Att kompilera och bygga var tidigare två separata steg – idag i VS.NET så gör båda tillsammans. Att **kompilera** innebär bl.a. att syntax kontrolleras, d.v.s. hur kod ska skrivas, och att **bygga** (*build*) innebär att exekverbar kod skapas. För att bygga kod kan vi välja Build→Build Solution (eller trycka Ctrl-Shift-B på tangentbord). Resultatet av byggande placeras i bin-mappen under projektmappen.

När vi kontrollerat att syntax är felfri så är det dags att börja buggtesta koden, d.v.s. köra applikation och försöka åtgärda fel som kan uppstå. För att göra detta så kan vi antingen välja Debug→Start eller klicka på spelaknappen i verktygsfältet (eller F5 på tangentbordet ☺).

När vi har buggtestat applikation så är det dags att skapa den slutgiltiga versionen av applikation, den s.k. *release*-versionen. Detta kan göras genom att välja Release (istället för Debug) i listruta till höger om körknappen i verktygsfältet samt bygga applikationen igen. Det vi behöver distribuera är alla EXE- och DLL-filer i projektmappens bin-mapp.

2.4 Funktioner och inställningar i VS.NET

Nedan beskrivs några användbara finesser/funktioner i VS.NET (inte så många idag…☺) och några praktiska inställningar som kan vara av intresse.

2.4.1 Visa radnummer i kodfönster [ATT GÖRA]

När vi kompilerar kod eller felsöker (*debug*) så visas oftast radnummer som fel finns respektive uppstod på. Men av någon anledning så visas inte radnummer som standard i kodfönstret. Nedan beskrivs hur vi ”plockar fram” radnumren.

2.4.2 Dölja/visa kod och regioner

I VS.NET kan vi dölja och visa olika delar av koden, t.ex. en hel klass eller en metod, genom att klicka på minus- respektive plustecknet längst till vänster om klassen respektive metoden.

Vi kan även skapa s.k. **regioner** av kod som kan döljas eller visas. För att skapa en region så använder vi direktiven `#Region` och `#End Region` för att tala om var regionen börjar respektive slutar. I exemplet nedan skapas en region med ”namnet” Namn på region och i bilden nedanför visas hur det tar sig ut i VS.NETs IDE när regionen visas respektive döljs.

```
#Region "Namn på region"
  'Kod som ska kunna döljas eller visas
```

```
    #End Region
```

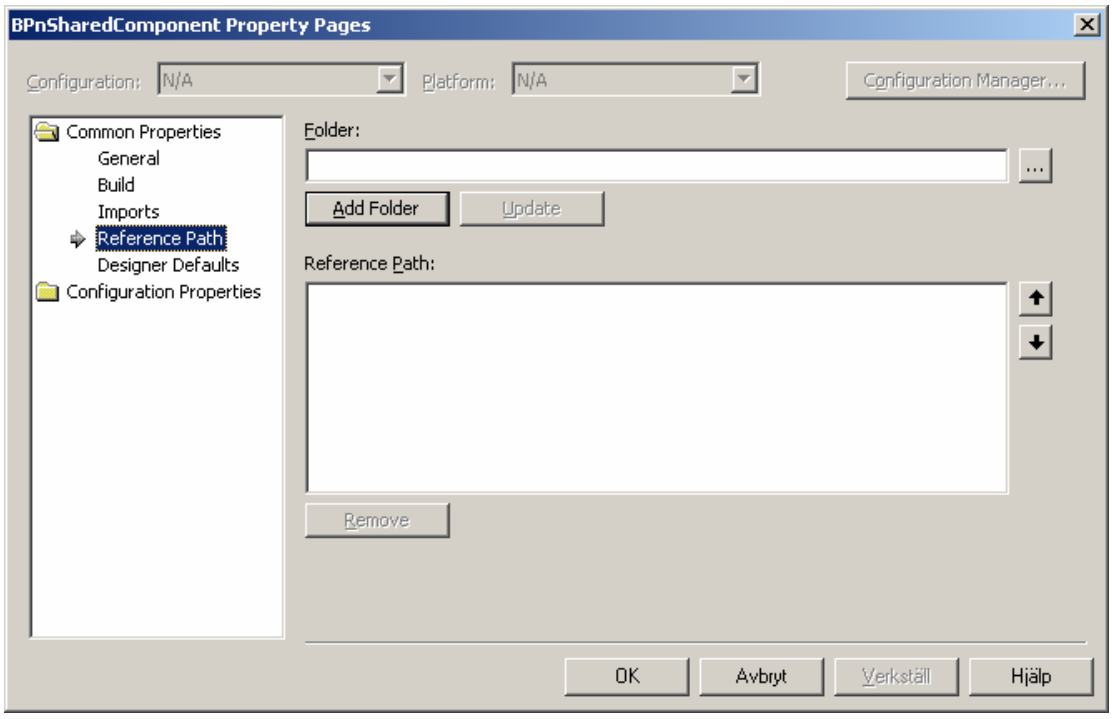
```
#Region "Namn på region"
    'Kod som ska kunna döljas eller visas
#End Region

Namn på region
```

Figur 1 - Regioner i VS.NET – expanderade och kollapsade.

2.4.3 Avancerat: Egna delade *assemblies* i dialogrutan Add Reference i VS.NET

Om vi skapar egna delade *assemblies* så hamnar dessa **inte** i dialogrutan Add Reference bara för att dom registreras i GAC. Vill vi, på ett relativt enkelt sätt, kunna använda dialogrutan Add Reference för att ange en referens till våra delade *assemblies* kan vi konfigurera en (eller flera) sökvägar i VS.NET.

1. Skapa en mapp (t.ex. `C:\Student\DeladeAssemblies\`) att placera våra DLL:er i.
2. Högerklicka på projektet där vi vill använda egen utvecklade *assemblies* i och välj **Properties**. Dialogrutan <Projektnamn> Property Pages visas (se bild nedan).



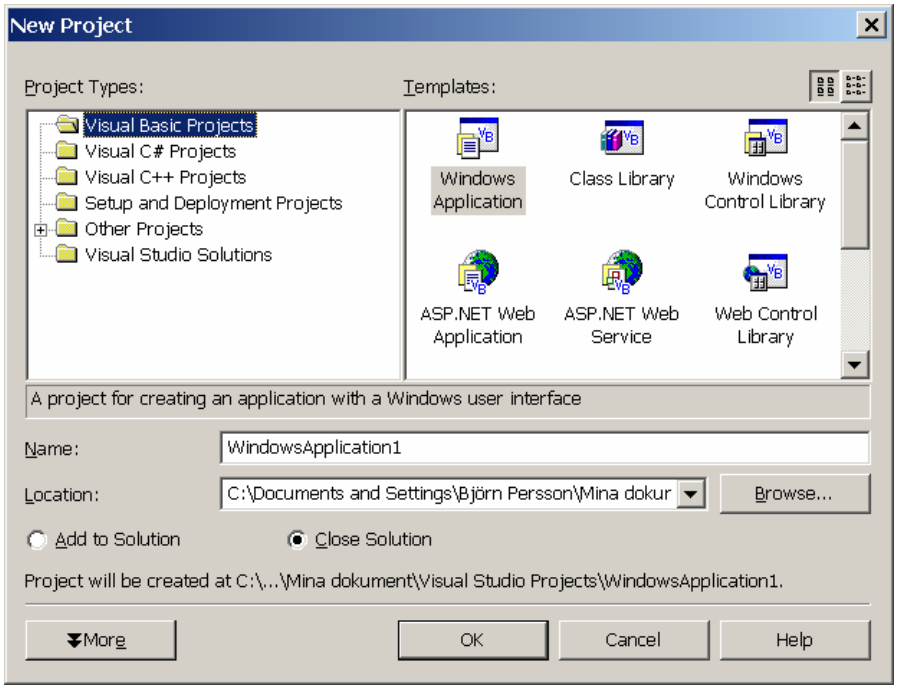
3. Expandera **Common Properties** och klicka på **Reference Path**.
4. I textrutan Folder – fyll i sökvägen till mapp (eller bläddra till mapp genom att klicka på knappen med tre punkter till höger om textruta) som skapades i punkt 1 och klicka på **Add Folder**.
5. Klicka på **OK** för att stänga dialogrutan <Projektnamn> Property Pages.

3 Projekt i Visual Studio.NET

Innan vi kan börja programmerar i Visual Studio.NET så måste vi skapa ett nytt projekt (beskrivs nedan). Ett **projekt** motsvarar oftast en binärfil – EXE eller DLL – där källkoden främst består av ett programmeringsspråk. Ett projekt kallas även för ***assembly*** eller **sammansättning**.[7] [8] Första projektet vi skapar skapar även en lösning (*solution*). En **lösning** kan innehålla flera projekt (som projektgrupper i Visual Basic 6). [FLYTTA stycke tidigare?]

När Visual Studio.NET startar så visas startsidan – har projekt redan skapats så brukar lista med de fyra senaste visas. För att skapa ett nytt projekt kan vi klicka på knappen New Project (på fliken Get Started) eller välja New→Project… från File-menyn (Ctrl-Shift-N). Detta visar dialogrutan New Projekt där vi kan välja

- programmeringsspråk
- typ av projekt
- namn på projekt
- samt var projekt ska sparas (gäller ej webbprojekt).



Typerna av projekt är de samma för Visual Basic.NET (VB.NET) och C#, men skiljer sig en del när det gäller Visual C++ (skillnader som inte kommer behandlas i denna sammanfattning). Nedan beskrivs typerna i VB.NET och C# samt vilken typ av filer som projekt resulterar i.

---

[7] En *assembly* kan bestå av en eller flera filer – binär- eller resursfiler. Men i enklare projekt så kan man oftast betrakta ett projekt som en binärfil, d.v.s. en *assembly*.

[8] *Assembly* översätts till sammansättning, men det är ett ord jag inte kan smälta (ännu i.a.f. ☺) så jag använder det engelska ordet i resten av sammanfattningen.

Typ av projekt	**Beskrivning**	**Filtyp**
Windows Application	Applikation baserat på grafiskt gränssnitt (*Windows Forms*).	`EXE`
Class Library	Klassbibliotek för t.ex. komponenter.	`DLL`
Windows Control Library	Grafisk kontroll för Windows-formulär.	`DLL`
ASP.NET Web Application	Applikation baserat på webbgränssnitt (*Web Forms*).	`ASPX`
ASP.NET Web Service	Klassbibliotek för webbtjänster som t.ex. anropas via SOAP.	`DLL`
Web Control Library	Grafisk kontroll för webbformulär.	`DLL`
Console Application	Applikation baserat på textbaserat gränssnitt.	`EXE`
Windows Service	Applikation utan gränssnitt, d.v.s. exekverar utan att någon är inloggad på dator och utan interaktion med användare.	`DLL`
Empty Project	Tomt projekt ☺.	-
Empty Web Project	Tomt webbprojekt ☺ – kräver en webbserver.	-
New Project In Existing Folder		-

Denna sammanfattning kommer titta på projekttyperna 1, 2, 4, 5 och 7 i tabellen ovan. Exempel nedan bygger på Visual Basic.NET som språk, men projekt för C# (och i viss mån Visual C++) fungerar på liknande sätt. I de flesta fall kan ”vb” ersättas med ”cs” (i t.ex. filnamn) om vi använder C# istället för VB.NET.

3.1 Att tänkta på…

Innan vi går vidare och skapar projekt så finns det en del saker vi bör veta eller tänka på.

3.1.1 Använd inte svenska tecken (å, ä eller ö) i projektnamn

Ibland uppstår problem (bl.a. med .NET:s säkerhetsinställningar) om vi använder svenska tecken i projektnamn, d.v.s. felmeddelanden visar inte alltid på varför felet uppstår (svårt att felsöka med andra ord). Detta gäller även mappar i sökväg som vi lägger projekt. Vi bör även undvika att använda svenska tecken i klasser, attribut/egenskaper, metodnamn, m.m..

Använd därför endast svenska tecken i strängar. Detta är något som gäller för de flesta produkter som utvecklats av icke-svenskar (läs amerikaner[9] ☺), d.v.s. inte bara VS.NET.

(För att slippa fler potentiella problem så bör vi undvika mellanslag i fil- och mappnamn, men gärna även fil- eller mappnamn som är längre än 8+3[10] tecken.)

3.1.2 Fundera på projektnamn (och andra namn) innan projekt skapas

Att byta namn på ett projekt innebär att man måste ändra på flera ställen. Fundera därför på ett lämpligt projekt namn **innan** du skapar ett projekt.

Om projektnamn innehåller mellanslag så kommer VS.NET automatiskt att ersätta dessa med understrykningstecken (”_”) på en del ställen (t.ex. i *assembly*-namn).

[9] Amerikaner och andra engelsk talande människor använder oftast (antagligen) inte svenska tecken i filnamn, vilket antagligen innebär att programmen inte avlusats (*debugged*) med dem.
[10] 8 tecken i filnamn och 3 tecken i filextension med en punkt mellan, t.ex. `Filnamn.txt`.

3.1.3 Säkerhetsproblem med placering av projektfiler

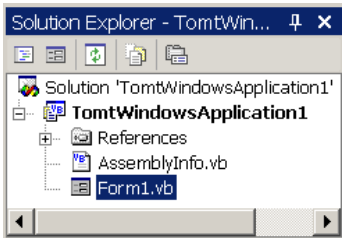
Med .NET Framework (som .NET-program exekveras i) så kommer även säkerhets-inställningar som kan ställa till problem. Projekt (och dess filer) bör därför placeras på en lokal hårddisk och inte på en filserver (s.k. hemkatalog).

På Ekonomihögskolan har datorer i datorsalar konfigurerats för att tillåta att utveckla och exekvera .NET-program från studenters hemkataloger.[11] Detta är också (bortsett från eventuella USB-minnen ☺) det enda stället som studenter kan spara filer permanent.

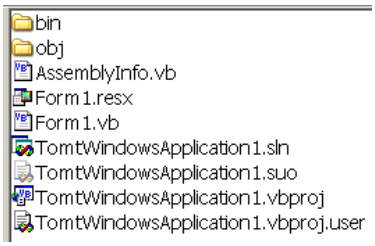
3.2 Windows Application

Projekt av typen *Windows Application* bygger på grafiska gränssnitt, d.v.s. formulär av typen *Windows Forms* (*WinForms*), och kompileras till en EXE-fil (d.v.s. ett program). Principen för grafiska gränssnitt är att något exekveras när användare gör något i applikation, t.ex. klickar på en knapp eller väljer ett alternativ i en meny. Detta kallas händelsestyrd exekvering.

3.2.1 Filer i projekt och referenser



När vi skapat ett ”tomt” projekt så visas två filer i *Solution Explorer*, eller lösningsfönstret (se bild till höger): `Form1.vb`[12], källkodsfil med bl.a. klass för formulär, och `AssemblyInfo.vb`[13], innehållande information om *assembly* (projekt).



Fast tittar vi i Utforskaren (se bild till höger) så ser vi en hel del filer till: två för lösning (`.SLN` och `.SUO`) och två för projektet (`.VBPROJ` och `.VBPROJ.USER`).[14] Några av dessa filer kan vi se i lösningsfönstret genom att klicka på knappen *Show All Files* i längst upp i lösningsfönstret.

Noden *References* (i lösningsfönstret) är *assemblies* som refereras till av projektet, d.v.s. vilka *assemblies* som projektet är beroende av. Som standard är de ”vanligaste” *assemblies* refererade till (5 stycken, bl.a. för *Windows Forms* och databasåtkomst).

3.2.2 VB6 kontra VS.NET

En stor skillnad mellan formulär i VB6 och VB.NET är att formulär i VB.NET numera är klasser. Alla kontroller som placeras i ett formulär får en motsvarande instansvariabel i klassen.

3.2.3 Lägga till fler projekt i lösning

Eftersom en lösning kan bestå av flera projekt så kan vi lägga till ett projekt i aktuell lösning genom att t.ex. högerklicka på lösning (inte existerande projekt!) i lösningsfönstret samt välja **Add…** och sen **New Project…**. Lägger vi till ett projekt till i lösningen så skapas som ”standard” en ny mapp för det nya projektet (på samma nivå i filsystem som existerande

---

[11] Detta gällde innan 2006… Från 2006 kommer inte Ekonomihögskolan ha egna datatekniker, d.v.s. jag vet inte hur administration av datorsalar kommer att fungera…
[12] Eller `Form1.cs` om C#-projekt.
[13] Filen `AssemblyInfo.vb`, eller `AssemblyInfo.cs` om C#-projekt, behöver vi endast använda om vi t.ex. vill lägga till namn på programmerare, företag som skapat *assembly* eller om vi skapar komponenter för COM+.
[14] Om projekt är för C# så innehåller filnamnen ”CS” istället för ”VB”.

projekt). Filerna för lösningen (bl.a. SLN-filen) finns dock bara i den första projektmappen. Detta är främst intressant, i detta fall, om vi lägger till klassbibliotek.

3.3 Class Library

Klassbibliotek används bl.a. för att samla klasser i en modul (*assembly*) och för att kunna dela kod mellan olika Windows-applikationer. Om vi ska skapa komponenter, som t.ex. ska exekvera i komponenttjänster (COM+), så är det ett klassbibliotek vi ska skapa. Några av skillnaderna mellan projekt för en Windows-applikation och ett klassbibliotek är att formulärfilen är ersatt med en "ren" klassfil (när projektet skapas) och att klassbibliotek kompileras till en DLL-fil. ☺

3.3.1 Filer i projekt och referenser

Precis som med Windows-applikationer så hittar vi två filer i projektet, men formulärfilen har som sagt ersatts med en "vanlig" klassfil, `Class1.vb`. Antalet referenser har även minskat till 3, då vi (oftast) inte har behov av *assemblies* för gränssnitt.

3.3.2 Lägga till fler projekt i lösning

Om vi vill testa ett klassbibliotek, må det vara i det faktiska användargränssnittet eller i ett testgränssnitt, så är det praktiskt att skapa ett Windows- eller konsolapplikationsprojekt först och lägga till klassbiblioteket efteråt. Därmed kan vi köra applikationen genom att välja Start från Debug-menyn (eller klicka på Start-knappen i verktygsfält. …eller trycka på F5-tangenten). Eller så kan vi högerklicka på användargränssnittets projekt och välja *Set as StartUp Project* om vi lägger till användargränssnittet efteråt. Det viktiga är att vi tänker på namnet på projekts namn innan vi skapar det första projektet då detta även kommer namnet på namnutrymmen (eller att vi ser till att ändra namn på namnutrymmen då vi lägger till filer i projekt).

3.3.3 Använda klassbibliotek i andra projekt

För att använda klassbiblioteket i ett annat projekt (t.ex. Windows-gränssnitt) så måste vi lägga till en referens till DLL-filen (*assembly*) med klassbiblioteket. Det gör vi genom att högerklicka på Reference-grenen i lösningsfönstret och välja **Add Reference** i menyn som visas. Klicka sen på Browse-knappen, bläddra till DLL-fil och markera den samt klicka på **Open**. Sist klickar vi på **OK** för att stänga dialogrutan *Add Reference*. Detta kopierar DLL-filen till "binärmappen"[15] (för användande projekt). Detta sätt används då vi använder privata *assemblies* (se nästa avsnitt).

3.3.4 Privata och delade *assemblies*

När vi skapar klassbibliotek så kan de antingen vara privata eller delade *assemblies*. En privat *assembly*, d.v.s. DLL-filen den finns i, används endast av en applikation. Privata *assemblies* är lättast att utveckla men innebär (oftast) att samma DLL-fil kan finnas i flera kopior (och eventuellt versioner). Delade *assemblies* däremot används av flera applikationer, d.v.s. det finns endast en kopia av filen.[16] För att göra delade assemblies tillgängliga så installeras de i

[15] "Binärmappen" är mappen som resulterande binärfil, EXE eller DLL, kompileras till. När vi avlusar (*debug*) så kan binärfilen hamna i en mapp medan slutversionen (*release version*) kan hamna i en annan. Båda typer av filer brukar hamna i projektets BIN-mapp eller en mapp därunder.

[16] Nåja… detta är inte riktigt sant. Det kan finnas flera version av filen men endast en kopia av varje version.

Global Assembly Cache (GAC[17]). Se sammanfattningen *Komponenter med MTS/COM+ i .NET* för hur delade *assemblies* skapas och installeras i GAC.

3.4 Console Application [ATT GÖRA]

3.5 ASP.NET Web Application [ATT GÖRA]

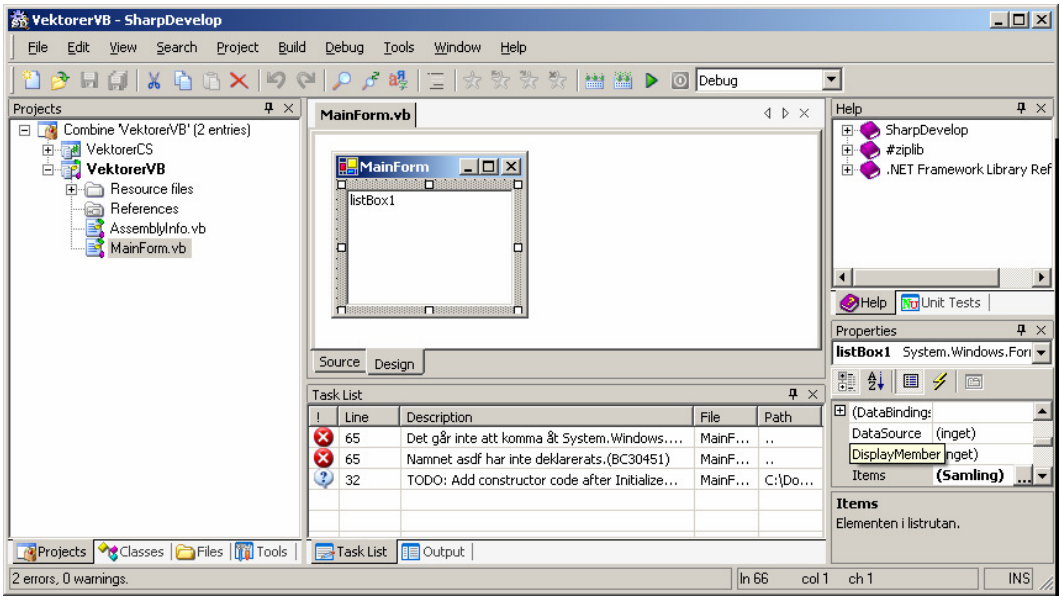
Om vi inte använder en egen dator, eller s.k. *roaming profiles*[18] i nätverk, så bör vi först ändra sökvägen till projekten för att vi ska kunna öppna projektet nästa gång vi loggar på. (Projekt sparas som standard i inloggad användares profil – profiler som endast sparas lokalt på dator i Ekonomihögskolans datorsalar och som raderas när studenter loggar ut.) Välj Options… på Tools-menyn, expandera grenen *Enviroment* och markera *Projects and Solutions*. Ändra sökvägen i textrutan *Visual Studio projects location* och klicka på OK för att stänga dialogrutan.

3.6 ASP.NET Web Service [ATT GÖRA]

[17] GAC finns i fysiskt mappen `C:\Windows\assembly` i Windows XP, d.v.s. under systemmappen `%WINDIR%`.
[18] *Roaming profiles* används för att skrivbordet i Windows (och alla inställningar, d.v.s. profilen) ska sparas (i hemkatalog i nätverket) när användaren loggar ut samt ska kopieras till lokal dator när användaren loggar på.

4 Använda SharpDevelop

SharpDevelop (SD) är en IDE som installeras ”ovan på” .NET Framework, d.v.s. .NET Framework måste vara installerad. .NET Framework kan laddas ner gratis från Microsofts hemsidor och senaste versionen av SharpDevelop kräver (idag) .NET Framework 1.1. SharpDevelop påminner mycket om VS.NET (liksom många andra IDE:er)[19] men skiljer sig en aning i placering av fönster samt vad saker och ting kallas.



Nedan beskrivs de vanligaste delarna av (fönstren i) SharpDevelop (från vänster till höger och uppifrån och ner). De flesta fönstren kan innehålla flera saker, vilket löses genom att placera innehållet på flikar (oftast längst ner i fönstret). Vissa fönster visas inte alltid och fönster kan flyttas runt (dockade eller odockade), vilket gör att IDE:ns utseende (layout) kan variera. Men i bild ovan visas hur layout brukar ta sig ut efter installation med en lösning öppen.

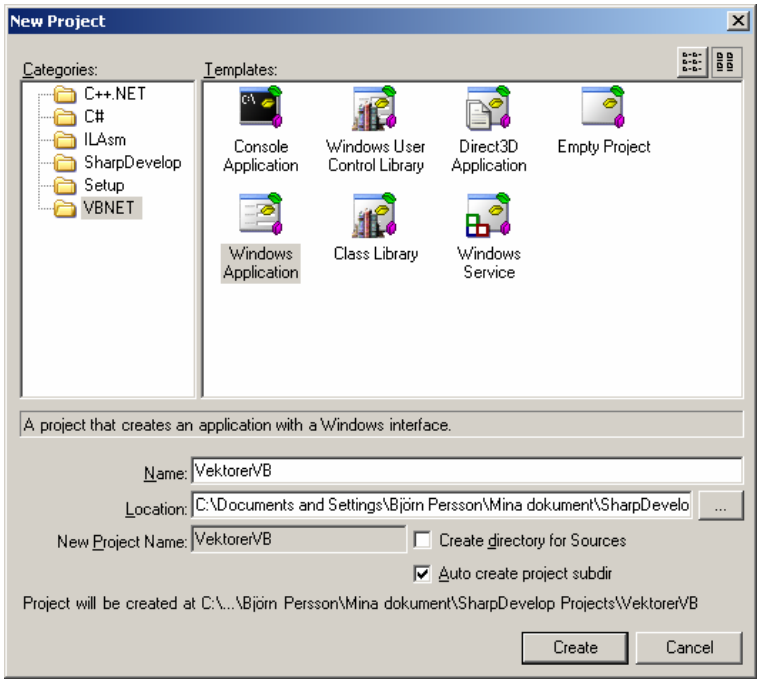
- Längst till vänster finns **lösningsfönstret** (*Projects*) där projekt och filer i lösning visas. För att öppna koden för ett formulär (eller annan kodfil) så dubbelklickar vi på fil i fönstret. I detta fönster kan även **klassvyn** (*Classes*) visas med alla klasser ordnade hierarkiskt i projekt och namnutrymmen (inte enligt arv!). Även **verktygsfönstret** (*Tools*) med kontroller för att designa formulär visas här.
- I mitten hittar vi **kod-/designfönstret** för att designa formulär och skriva kod för logiken i formulär. En skillnad mot VS.NET är att det endast finns en flik per formulär längst upp – i nederkanten hittar vi dock två flikar för att skifta mellan kod och design av formulär. Här visas även hjälpsidor som söks fram i hjälpen.
- För att bläddra i hjälpen använder vi **hjälpfönstret** (*Help*). Hjälpen i SharpDevelop är inte lika utvecklad som i VS.NET… Men SharpDevelop är gratis och vi kan alltid hoppas på bättre funktion i framtida versioner.
- Längst ner hittar vi **aktivitetslistan** (*Task List*) där bl.a. kompileringsfel (röda kryss i bild ovan) visas samt TODO-kommentarer (kommentarer som börjar med TODO). Vid kompilering (och exekvering) visas resultatet i **resultatfönstret** (*Output*).
- För att ändra egenskaper för bl.a. kontroller i ett formulär används **egenskapsfönstret** (*Properties*).

[19] Microsoft är duktiga på att utveckla andras idéer, d.v.s. alla deras produkter är inte deras egna idéer.

4.1 Lösningar och projekt

En **lösning** är ett sätt att samla ett eller flera projekt vid utveckling av applikation. Lösningar (*solutions*) i SD kallas för *combines* (kombination eller sammansättning). Ett **projekt** motsvarar oftast en assembly (d.v.s. en EXE- eller DLL-fil) och kallas för projekt även i SharpDevelop. ☺ Jag kommer dock använda ordet lösning för *combine* även i samband med beskrivning av SD.

För att skapa ett nytt projekt välj File→New→Combine (eller klicka på knappen New Combine på Start Page som visas när SD startas). Detta visar dialogrutan New Project (se bild till höger). Välj språk (VBNET för VB.NET) som projekt ska innehålla samt typ av projekt att skapa: *Windows Application* för en klientapplikation (EXE) eller *Class Library* för en DLL.



Projekt sparas som standard i en mapp, med samma namn som projektet, under `Mina dokument\SharpDevelop Projects` (något som kan ändras varje gång vi skapar ett projekt eller med inställningar i SharpDevelop[20]). Nedan beskrivs typer av filer i projekt och vad de innehåller.

Filtyp	**Innehåll**
`*.vb` (eller `*.cs`)	Kodfiler. Filändelse visar, praktiskt nog, på språk i fil. ☺
`<projektnamn>.prjx`	Innehåller information om projekt, bl.a. filer som ingår i projekt.
`<projektnamn>.cmbx`	Innehåller information om lösning, bl.a. projekt som ingår.
`AssemblyInfo.vb`	Fil med inställningar för projekt, bl.a. för komponenttjänster.

Programfiler (EXE- och DLL-filer) placeras i mappen `bin` under projektmappen.

4.2 Kompilera, bygga och köra

Att kompilera och bygga var tidigare två separata steg – i SharpDevelop så gör båda tillsammans. Att **kompilera** innebär bl.a. att syntax kontrolleras, d.v.s. hur kod ska skrivas, och att **bygga** (*build*) innebär att exekverbar kod skapas. För att bygga kod kan vi välja Build→Build Combine (eller trycka F8 på tangentbord). Resultatet av byggande placeras i en mapp `BIN` (oftast[21]) under projektets mapp.

[20] Välj Tools→Options samt markera mappen SharpDevelop options och sen alternativet Projects and Combines.
[21] Med oftast menas att ibland händer det att bin-mappen hamnar under mappen `SharpDevelop Projects`. Detta gäller främst när vi lägger till ett andra projekt till en lösning.

När vi kontrollerat att syntax är felfri så är det dags att börja buggtesta koden, d.v.s. köra applikation och försöka åtgärda fel som kan uppstå. För att göra detta så kan vi antingen välja Debug→Run eller klicka på spelaknappen i verktygsfältet (eller F5 på tangentbordet ☺).

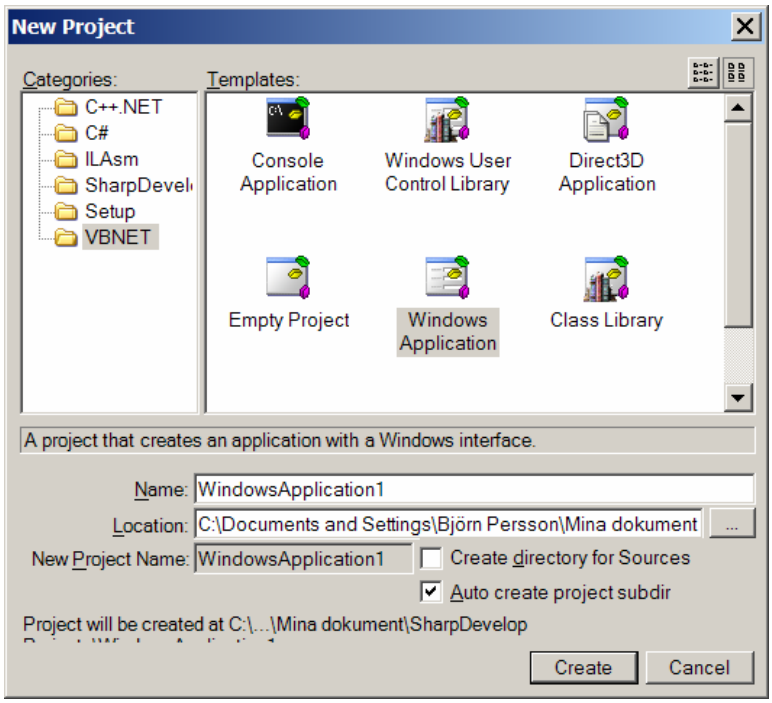
När vi har buggtestat applikation så är det dags att skapa den slutgiltiga versionen av applikation, den s.k. *release*-versionen. Detta kan göras genom att välja Release (istället för Debug) i listruta till höger om körknappen i verktygsfältet samt bygga applikationen igen. Det vi behöver distribuera är alla EXE- och DLL-filer i Release-mappen under projektmappens bin-mapp.

5 Projekt i SharpDevelop

Innan vi kan börja programmerar i SharpDevelop så måste vi skapa ett nytt projekt (beskrivs nedan). Ett **projekt** motsvarar oftast en binärfil – EXE eller DLL – där källkoden främst består av ett programmeringsspråk. Ett projekt kallas även för ***assembly*** eller **sammansättning**.[22] [23] Första projektet vi skapar skapar även en lösning (*solution*). En **lösning** kan innehålla flera projekt. [FLYTTA stycke tidigare?]

När SharpDevelop startar så visas startsidan – har projekt redan skapats så brukar lista med de senaste visas. För att skapa ett nytt projekt kan vi klicka på knappen New Combine (på fliken Start Page) eller välja New→Combine… från File-menyn (Ctrl-Shift-N). Detta visar dialogrutan New Projekt där vi kan välja

- programmeringsspråk
- typ av projekt
- namn på projekt
- samt var projekt ska sparas.



Typerna av projekt är de samma för Visual Basic.NET (VB.NET) och C#, men skiljer sig en del när det gäller Visual C++ (skillnader som inte kommer behandlas i denna sammanfattning). Nedan beskrivs typerna i VB.NET och C# samt vilken typ av filer som projekt resulterar i.

---

[22] En *assembly* kan bestå av en eller flera filer – binär- eller resursfiler. Men i enklare projekt så kan man oftast betrakta ett projekt som en binärfil, d.v.s. en *assembly*.

[23] *Assembly* översätts till sammansättning, men det är ett ord jag inte kan smälta (ännu i.a.f. ☺) så jag använder det engelska ordet i resten av sammanfattningen.

Typ av projekt	**Beskrivning**	**Filtyp**
Console Application	Applikation baserat på textbaserat gränssnitt.	`EXE`
Windows Control Library	Grafisk kontroll för Windows-formulär.	`DLL`
Direct3D Application	Applikation med ett Direct3D-gränssnitt.	`EXE`
Empty Project	Tomt projekt ☺.	-
Windows Application	Applikation baserat på grafiskt gränssnitt (*Windows Forms*).	`EXE`
Class Library	Klassbibliotek för t.ex. komponenter.	`DLL`
Windows Service	Applikation utan gränssnitt, d.v.s. exekverar utan att någon är inloggad på dator och utan interaktion med användare.	`DLL`

Denna sammanfattning kommer titta på projekttyperna 1, 5 och 6 i tabellen ovan. Exempel nedan bygger på Visual Basic.NET som språk, men projekt för C# (och i viss mån Visual C++) fungerar på liknande sätt. I de flesta fall kan ”vb” ersättas med ”cs” (i t.ex. filnamn) om vi använder C# istället för VB.NET.

5.1 Att tänkta på…

Innan vi går vidare och skapar projekt så finns det en del saker vi bör veta eller tänka på.

5.1.1 Använd inte svenska tecken (å, ä eller ö) i projektnamn

Ibland uppstår problem (bl.a. med .NET:s säkerhetsinställningar) om vi använder svenska tecken i projektnamn, d.v.s. felmeddelanden visar inte alltid på varför felet uppstår (svårt att felsöka med andra ord). Detta gäller även mappar i sökväg som vi lägger projekt. Vi bör även undvika att använda svenska tecken i klasser, attribut/egenskaper, metodnamn, m.m..

Använd därför endast svenska tecken i strängar. Detta är något som gäller för de flesta produkter som utvecklats av icke-svenskar, d.v.s. inte bara SharpDevelop.

(För att slippa fler potentiella problem så bör vi undvika mellanslag i fil- och mappnamn, men gärna även fil- eller mappnamn som är längre än 8+3[24] tecken.)

5.1.2 Fundera på projektnamn (och andra namn) innan projekt skapas

Att byta namn på ett projekt innebär att man måste ändra på flera ställen. Fundera därför på ett lämpligt projekt namn **innan** du skapar ett projekt.

Om projektnamn innehåller mellanslag så kommer SharpDevelop automatiskt att ersätta dessa med understrykningstecken (”_”) på en del ställen.

5.1.3 Säkerhetsproblem med placering av projektfiler

Med .NET Framework (som .NET-program exekveras i) så kommer även säkerhets-inställningar som kan ställa till problem. Projekt (och dess filer) bör därför placeras på en lokal hårddisk och inte på en filserver (s.k. hemkatalog).

På Ekonomihögskolan har datorer i datorsalar konfigurerats för att tillåta att utveckla och exekvera .NET-program från studenters hemkataloger.[25] Detta är också (bortsett från eventuella USB-minnen ☺) det enda stället som studenter kan spara filer permanent.

[24] 8 tecken i filnamn och 3 tecken i filextension med en punkt mellan, t.ex. `Filnamn.txt`.

5.2 *Console Application [ATT GÖRA]*

En konsolapplikation är en textbaserad applikation och kompileras till en EXE-fil (d.v.s. ett program). Denna typ av applikationer är sekventiella, d.v.s. ordningen för exekvering sker främst från början till slut (men kan göras mer interaktiva med en ”menyloop”).

5.2.1 Filer i projekt och referenser

När vi skapar en konsolapplikation så finns det en fil i *Projects*, eller lösningsfönstret: `Main.vb`[26], källkodsfil med modul[27]. I Utforskaren finns ytterligare två filer för lösning och projekt (`.CMBX` resp. `.PRJX`).

Noden *References* (i lösningsfönstret) är *assemblies* som refereras till av projektet, d.v.s. vilka *assemblies* som projektet är beroende av.[28]

5.2.2 En enkel konsolapplikation

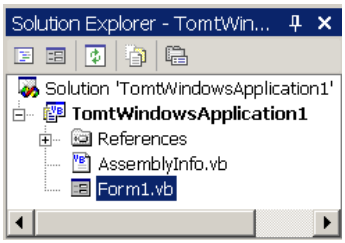
När vi skapar ett nytt konsolapplikationsprojekt så generar SharpDevelop följande kod i filen `Main.vb`.

```
Imports System
Module Main
  Sub Main()
    Console.WriteLine("Hello World!")
  End Sub
End Module
```

5.3 *Windows Application*

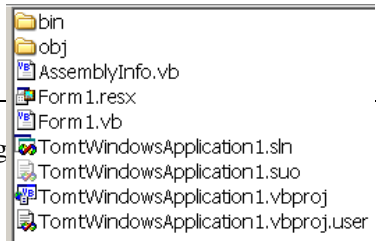
Projekt av typen *Windows Application* bygger på grafiska gränssnitt, d.v.s. formulär av typen *Windows Forms* (*WinForms*), och kompileras till en EXE-fil (d.v.s. ett program). Principen för grafiska gränssnitt är att något exekveras när användare gör något i applikation, t.ex. klickar på en knapp eller väljer ett alternativ i en meny. Detta kallas händelsestyrd exekvering.

5.3.1 Filer i projekt och referenser [GÖR OM BILDER]



När vi skapat ett ”tomt” projekt så visas två filer i *Projects*, eller lösningsfönstret (se bild till höger): `MainForm.vb`[29], källkodsfil med bl.a. klass för formulär, och `AssemblyInfo.vb`[30], innehållande information om *assembly* (projekt).

Fast tittar vi i Utforskaren (se bild till höger) så ser vi två filer till för lösning och projekt (`.CMBX` resp. `.PRJX`).



[25] Detta gällde innan 2006… Från 2006 kommer inte Ekonomihögskolan ha eg
hur administration av datorsalar kommer att fungera…
[26] Eller `Main.cs` om C#-projekt.
[27] I C# så är det en klass istället för modul – moduler är unika för VB.NET.
[28] I motsats till VS.NET så visas inte de ”vanliga” referenserna under fliken References i SharpDevelop.
[29] Eller `MainForm.cs` om C#-projekt.
[30] Filen `AssemblyInfo.vb`, eller `AssemblyInfo.cs` om C#-projekt, behöver vi endast använda om vi t.ex. vill lägga till namn på programmerare, företag som skapat *assembly* eller om vi skapar komponenter för COM+.

Noden *References* (i lösningsfönstret) är *assemblies* som refereras till av projektet, d.v.s. vilka *assemblies* som projektet är beroende av.[31]

5.3.2 Lägga till fler projekt i lösning

Eftersom en lösning kan bestå av flera projekt så kan vi lägga till ett projekt i aktuell lösning genom att t.ex. högerklicka på lösning (inte existerande projekt!) i lösningsfönstret samt välja **Add** och sen **Add new Project…**. Lägger vi till ett projekt till i lösningen så skapas som ”standard” en ny mapp för det nya projektet (på samma nivå i filsystem som existerande projekt). Ett problem är att bin-mapp för det nya projektet tenderar till hamna på fel ställe. Även det nya projektet har en fil för lösningen (CMBX-fil), men om vi använder den för att öppna projektet så laddas inte det första projektet.

Detta är främst intressant, i detta fall, om vi lägger till klassbibliotek.

5.4 *Class Library*

Klassbibliotek används bl.a. för att samla klasser i en modul (*assembly*) och för att kunna dela kod mellan olika Windows-applikationer. Om vi ska skapa komponenter, som t.ex. ska exekvera i komponenttjänster (COM+), så är det ett klassbibliotek vi ska skapa. Några av skillnaderna mellan projekt för en Windows-applikation och ett klassbibliotek är att formulärfilen är ersatt med en “ren” klassfil (när projektet skapas) och att klassbibliotek kompileras till en DLL-fil. ☺

5.4.1 Filer i projekt och referenser

Precis som med Windows-applikationer så hittar vi två filer i projektet, men formulärfilen har som sagt ersatts med en ”vanlig” klassfil, `MyClass.vb`.

5.4.2 Lägga till fler projekt i lösning

Om vi vill testa ett klassbibliotek, må det vara i det faktiska användargränssnittet eller i ett testgränssnitt, så är det praktiskt att skapa ett Windows- eller konsolapplikationsprojekt först och lägga till klassbiblioteket efteråt. Därmed kan vi köra applikationen genom att välja Start från Debug-menyn (eller klicka på Start-knappen i verktygsfält. …eller trycka på F5-tangenten). Eller så kan vi högerklicka på användargränssnittets projekt och välja *Set as StartUp Project* om vi lägger till användargränssnittet efteråt.

Det viktiga är att vi tänker på namnet på projekts namn innan vi skapar det första projektet då detta även kommer namnet på namnutrymmen (eller att vi ser till att ändra namn på namnutrymmen då vi lägger till filer i projekt).

5.4.3 Använda klassbibliotek i andra projekt

För att använda klassbiblioteket i ett annat projekt (t.ex. Windows-gränssnitt) så måste vi lägga till en referens till DLL-filen (*assembly*) med klassbiblioteket. Det gör vi genom att högerklicka på Reference-grenen i lösningsfönstret och välja **Add Reference** i menyn som visas. Klicka sen på fliken .NET Assembly Browser, Browse…-knappen, bläddra till DLL-fil och markera den samt klicka på **Open/Öppna**. Sist klickar vi på **OK** för att stänga dialogrutan *Add Reference*. DLL-filen kopieras till ”binärmappen”[32] (för användande projekt)

[31] I motsats till VS.NET så visas inte de ”vanliga” referenserna under fliken References i SharpDevelop.
[32] ”Binärmappen” är mappen som resulterande binärfil, EXE eller DLL, kompileras till. När vi avlusar (*debug*) så kan binärfilen hamna i en mapp medan slutversionen (*release version*) kan hamna i en annan. Båda typer av filer brukar hamna i projektets BIN-mapp eller en mapp därunder.

när projektet kompileras. Detta sätt används då vi använder privata *assemblies* (se nästa avsnitt).

5.4.4 Privata och delade *assemblies*

När vi skapar klassbibliotek så kan de antingen vara privata eller delade *assemblies*. En privat *assembly*, d.v.s. DLL-filen den finns i, används endast av en applikation. Privata *assemblies* är lättast att utveckla men innebär (oftast) att samma DLL-fil kan finnas i flera kopior (och eventuellt versioner). Delade *assemblies* däremot används av flera applikationer, d.v.s. det finns endast en kopia av filen.[33] För att göra delade assemblies tillgängliga så installeras de i *Global Assembly Cache* (GAC[34]). Se sammanfattningen *Komponenter med MTS/COM+ i .NET* för hur delade *assemblies* skapas och installeras i GAC.

6 Fler verktyg av intresse

För databaser så kan vi använda Access, SQL Server/MS Desktop Engine (MSDE), Oracle och/eller MySQL.

För enklare verktyg för webbutveckling med ASP.NET finns ASP.NET Web Matrix, ett verktyg som kan användas på egen dator (inte på en server över nätverk). Om man inte vill installera IIS så installeras en enkel webbserver med Web Matrix.

[33] Nåja… detta är inte riktigt sant. Det kan finnas flera version av filen men endast en kopia av varje version.
[34] GAC finns i fysiskt mappen `C:\Windows\assembly` i Windows XP, d.v.s. under systemmappen `%WINDIR%`.