

# **Komponenter med DCOM och MTS (COM+)**

## Om denna sammanfattning

Denna sammanfattning avser att ge en inblick i komponentteknologier med Microsofts DCOM och MTS/COM+. Exempel i denna sammanfattning bygger på DCOM (VB och VC++) och MTS/COM+ (VB). Sammanfattningen är en "fortsättning" på sammanfattningen *Komponenter med COM (och COM+)*.

Observera att detta **inte är en ersättning för eventuell kurslitteratur** och att en del exempel kan vara svåra att få att fungera med en gång. Exempel har utvecklats i (och beskrivits utifrån) Windows NT4 (med NT4 Option Pack installerat) och/eller Windows 2000/XP. Som verktyg för att utveckla komponenterna har Visual Studio 6 används (d.v.s. C++-exempel har inte testats i Visual Studio.NET! – se eventuell separat sammanfattning).

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

Eskilstuna, oktober 2003

Björn Persson, e-post: [bjorn.persson@mdh.se](mailto:bjorn.persson@mdh.se)  
Institutionen för ekonomi och informatik  
Mälardalens högskola  
Personlig hemsida: <http://www.eki.mdh.se/personal/bpn01/>

## Innehållsförteckning

|          |                                                       |           |
|----------|-------------------------------------------------------|-----------|
| <b>6</b> | <b>DISTRIBUTED COM.....</b>                           | <b>5</b>  |
| 6.1      | VISUAL BASIC .....                                    | 5         |
| 6.1.1    | Projektet och komponentens namn samt referenser ..... | 5         |
| 6.1.2    | Implementation av metoderna .....                     | 6         |
| 6.1.3    | Testa komponent lokalt .....                          | 7         |
| 6.1.4    | Skapa filer att placera på klient .....               | 9         |
| 6.1.5    | Skapa installationspaket till klienten.....           | 10        |
| 6.1.6    | Skapa installationsprogrammet.....                    | 14        |
| 6.1.7    | Konfigurera server för DCOM i NT4 .....               | 19        |
| 6.1.8    | Konfigurera server för DCOM i 2000/XP.....            | 20        |
| 6.1.9    | Installera på klienten.....                           | 21        |
| 6.1.10   | Skapa klientprogram.....                              | 22        |
| 6.1.11   | Provkör programmet.....                               | 23        |
| 6.2      | VISUAL C++ .....                                      | 23        |
| 6.2.1    | Skapa komponenten.....                                | 23        |
| 6.2.2    | Skapa DLL för proxy/stub.....                         | 24        |
| 6.2.3    | Registrera DLL för proxy/stub .....                   | 24        |
| 6.2.4    | Konfigurera komponent med DCOMCNFG.EXE .....          | 24        |
| 6.2.5    | Skapa klientprogram.....                              | 24        |
| 6.2.6    | Konfigurerar klient för var komponent finns.....      | 26        |
| 6.2.7    | Att tänka på om DCOM .....                            | 26        |
| <b>7</b> | <b>MICROSOFT TRANSACTION SERVER .....</b>             | <b>27</b> |
| 7.1      | TRANSAKTIONER I MTS.....                              | 27        |
| 7.1.1    | Visual Basic.....                                     | 27        |
| 7.1.2    | Visual C++ .....                                      | 28        |
| 7.2      | MTS SOM ORB OCH SURROGATPROCESS .....                 | 29        |
| 7.3      | SKAPA KOMPONENTER FÖR EXEKVERING I MTS .....          | 30        |
| 7.3.1    | Implementera gränssnittet <i>IObjectControl</i> ..... | 31        |
| 7.3.2    | Använda metoder i <i>IObjectContext</i> .....         | 32        |
| 7.4      | SKAPA OBJEKT SOM SKA EXEKVERA I MTS.....              | 34        |
| 7.5      | PAKET I MTS/KOMPONENTTJÄNSTER.....                    | 35        |
| 7.5.1    | Skapa ett paket i MTS.....                            | 35        |
| 7.5.2    | Skapa en tillämpningar i COM+ .....                   | 36        |
| 7.5.3    | Lägga till komponenter i ett paket.....               | 36        |
| 7.6      | TIPS NÄR MAN SKAPAR KOMPONENTER FÖR MTS/COM+.....     | 37        |
| <b>8</b> | <b>EXEMPEL: EN N-LAGER APPLIKATION.....</b>           | <b>39</b> |
| 8.1      | BESKRIVNING AV BOKNINGSSYSTEMET.....                  | 39        |
| 8.2      | DATABASEN.....                                        | 39        |
| 8.2.1    | Tabellernas utformning.....                           | 39        |
| 8.3      | KOMPONENTER.....                                      | 40        |
| 8.3.1    | Datakomponenten Read.....                             | 41        |
| 8.3.2    | Datakomponenten Write.....                            | 48        |
| 8.3.3    | Affärskomponenten Las.....                            | 50        |
| 8.3.4    | Affärskomponenten Skriv.....                          | 51        |
| 8.4      | FORMULÄR.....                                         | 53        |
| 8.4.1    | BokaDator.....                                        | 53        |
| 8.4.2    | VisaBokningar.....                                    | 54        |
| 8.4.3    | Main.....                                             | 55        |

(Denna sida har avsiktligt lämnats blank.)

## 6 Distributed COM

Komponentteknologier för distribuerade komponenter har någon form av teknologi för att använda komponenter på andra datorer/processer (*remote method invocation*, RMI). Dessa teknologier bygger ofta på att ett objekt placeras hos klienten (så att den gör ett lokalt anrop) och ett motsvarande objekt på server (som gör det faktiska anropet mot komponenten). I Microsofts miljö kallas dessa objekt för *proxy*- respektive *stub*-objekt (i Java RMI kallas de för *stub*- respektive *skeleton*-objekt!).

*Distributed COM* (DCOM), Microsofts version på RMI, kan ses som en förlängning av COM – när komponenten exekverar på en annan dator än klienten (d.v.s. komponenten är distribuerad) så griper DCOM in. DCOM sköter kommunikationen mellan klienten och komponent över nätverket utan att programmeraren behöver skriva någon speciell kod för denna kommunikation. Det ”enda” som krävs är att adressen till datorn med komponenten registreras på klienten och eventuellt att komponenten konfigurerats som distribuerad.

För att registrera komponenter på klienter och konfigurerar komponenter som distribuerade används verktyget **Distributed COM Configuration** (DCOMCNFG.EXE). I Windows 2000/XP har DCOMCNFG integrerats med MMC<sup>1</sup> *snap-in*<sup>2</sup> Komponenttjänster

För exemplen i detta kapitel så krävs tillgång till minst två datorer (helst med Windows NT/2000/XP<sup>3</sup>). COM-servern är utvecklad på en dator med Windows 2000 och testad även på Windows NT4. Klienten har utvecklats på en dator med Windows NT4. Enklast är också om dessa två datorer ingår i samma NT-domän<sup>4</sup> och/eller att samma användarkonto finns på båda datorer om endast arbetsgrupper (i Windows) används. För att (garantera att?) detta exempel ska fungera kommer vi att minska på säkerheten på datorn med COM-servern på, d.v.s. introducera en säkerhetsrisk!

---

### 6.1 Visual Basic

I detta exempel kommer vi att titta på hur man utvecklar en komponent i Visual Basic som vi sen ska skapa med hjälp av DCOM från en annan dator. Själva utvecklingen och testen av komponenten skiljer sig inte så mycket från utveckling av en lokal komponent. Men vi behöver hjälp av Visual Basics IDE att skapa två filer vi ska placera på klientens dator.

Komponenten innehåller två metoder som läser från en databas (*skivor.mdb*) och returnerar poster i Recordset-objekt. Nedan följer en beskrivning av hur vi skapar komponenten och konfigurerar DCOM så att klienten kan nå komponenten från en annan dator.

#### 6.1.1 Projektet och komponentens namn samt referenser

1. Skapa ett projekt av typen **ActiveX EXE**.
2. Ändra projektets namn till *SkivorDCOM* genom att markera projektet (Project1) i projektfönstret och ändra egenskapen (Name) i egenskapsfönstret. Markera sen klassen (Class1) i projektfönstret och ändra namnet till *Read* i egenskapsfönstret. Spara klassen och projektet i en mapp (t.ex. *SkivorDCOM*).

---

<sup>1</sup> Microsoft Management Console – ett enhetligt verktyg för att konfigurera applikationer, m.m. i Windows 2000/XP.

<sup>2</sup> En *snap-in* (eller *plug-in*) är modul som kan användas för att utöka något, t.ex. MMC (eller Internet Explorer).

<sup>3</sup> Det fungerar även med två datorer med Windows 9x, men det krävs att man laddar ner DCOM för Windows 95 resp. 98 från Microsofts hemsidor – <http://www.microsoft.com/com/>.

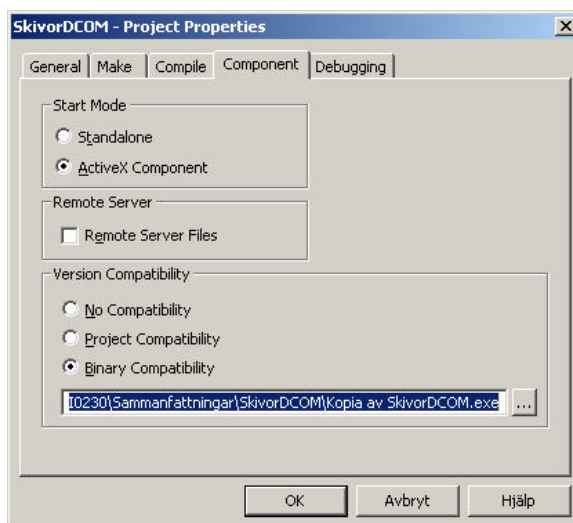
<sup>4</sup> EkI använder inte NT-domän! Men alla studentdatorer ingår i samma arbetsgrupp.

3. För att kunna jobba mot databaser sätter vi referenser till **Microsoft ActiveX Data Object 2.x Library**<sup>5</sup> för projektet.
4. Vi skapar skalkod för de två metoderna som komponenten ska ha – `GetAlbum()` och `GetAlbumForArtist()`. Båda två returnerar poster i ett Recordset-objekt.

```
Public Function GetAlbum() As ADODB.Recordset  
End Function  
  
Public Function GetAlbumForArtist(ByVal Artist As String) As ADODB.Recordset  
End Function
```

Sen skapar vi EXE-filen så att vi kan utnyttja *Binary Compatibility* så att vi inte av misstag ändrar komponentens gränssnitt.

5. För att skapa EXE-filen väljer vi **Make SkivorDCOM.exe...** från File-menyn.
6. Kopiera komponentens EXE-fil i Utforskaren och klistra in den direkt (vilket kommer ge den ett namn i stil med kopia av SkivorDCOM.exe). Öppna dialogrutan för projektets egenskaper genom att välja **SkivorDCOM Properties...** från Project-menyn. Klicka på fliken **Component**, markera alternativet **Binary Compatibility** samt klicka på knappen för att bläddra och välj en kopian av EXE-filen (vi precis skapade). Markera filen och klicka på knappen Öppna.



Klicka på knappen OK för att stänga dialogrutan och spara sen projektet igen.

## 6.1.2 Implementation av metoderna

Eftersom datakällan kommer vara den samma för alla metoder i komponenten så deklarerar vi vår `ConnectionString` som en konstant längst upp i vår klass. (För att visa att det är en konstant använder vi prefixet "c", c som i Const.)

```
Option Explicit  
Const cstrConn As String = "Provider=Microsoft.Jet.OLEDB.4.0;" _
```

<sup>5</sup> Använd helst version 2.5 eller senare av ADO.

& "Data Source=C:\Student\Skivor.mdb"

Implementationen av metoderna är bara "vanlig" kod för att hämta poster i databasen med Connection-objektets metod `Execute()`. Eftersom klienten kan finnas på en annan dator (vilket ju är halva meningen med DCOM ☺) så måste vi använda *disconnected* Recordset-objekt. För detta måste vi ange att klienten ska hantera postpekaren (*cursor*) och sen sätta Recordset-objektets egenskap `ActiveConnection` till `Nothing`.

```
Public Function GetAlbum() As ADODB.Recordset
    Dim adoConn As ADODB.Connection
    Dim adoRS As ADODB.Recordset
    Dim strSQL As String

    strSQL = "SELECT * FROM tblAlbum"
    Set adoConn = New ADODB.Connection

    adoConn.CursorLocation = adUseClient      'Ange att klient ska hantera postpekare
    adoConn.Open cstrConn
    Set adoRS = adoConn.Execute(strSQL)
    Set adoRS.ActiveConnection = Nothing      'Koppla loss Recordset-objekt från DB

    Set GetAlbum = adoRS                      'Returnera Recordset-objekt

    adoConn.Close                            'Stäng objekt och rensa upp
    Set adoRS = Nothing
    Set adoConn = Nothing
End Function

Public Function GetAlbumForArtist(ByVal Artist As String) As ADODB.Recordset
    Dim adoConn As ADODB.Connection
    Dim adoRS As ADODB.Recordset
    Dim strSQL As String

    strSQL = "SELECT * FROM tblAlbum WHERE Artist='" & Artist & "'"
    Set adoConn = New ADODB.Connection

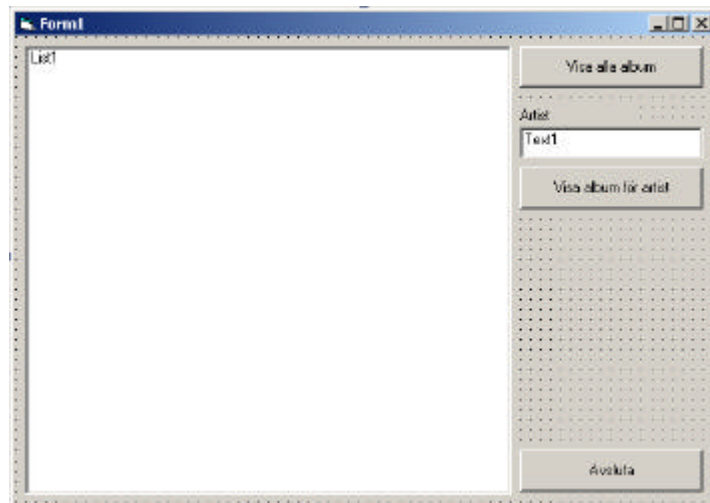
    adoConn.CursorLocation = adUseClient      'Ange att klient ska hantera postpekare
    adoConn.Open cstrConn
    Set adoRS = adoConn.Execute(strSQL)
    Set adoRS.ActiveConnection = Nothing      'Koppla loss Recordset-objekt från DB

    Set GetAlbumForArtist = adoRS            'Returnera Recordset-objekt

    adoConn.Close                            'Stäng objekt och rensa upp
    Set adoRS = Nothing
    Set adoConn = Nothing
End Function
```

### 6.1.3 Testa komponent lokalt

En bra idé är alltid att testa distribuerade komponenter lokalt innan vi börjar blanda in fler saker (som nätverkskommunikation) som kan fallera. Vi skapar därför ett formulär med en listruta (`List1`), en textruta (`Text1`) och tre kommandoknappar (`Command1`, `Command2` och `cmdAvsluta`) (se bild nedan). Som vanligt så är det praktiskt att använda etiketter för att tala om vad textrutor (och egentligen också listrutor) innehåller (ska innehålla). ☺



1. Skapa ett nytt projekt av typen **Standard EXE**. Döp projektet till t.ex. SkivorDCOMKlientLokalt.<sup>6</sup> Lägg till kontroller i formulär enligt bild ovan.
2. Sätt en referens till COM-servern, d.v.s. **SkivorDCOM**, och till **Microsoft ActiveX Data Object 2.x Library** (eftersom vi kommer få Recordset-objekt i retur från metoder).
3. Deklarera en global variabel för vår komponent, t.ex. objSkivor. Variabeln deklareras så globalt som möjligt då den (teoretiskt sätt) ska skapas på en annan dator. D.v.s. vi vill hålla en referens till komponenten så länge programmet exekverar.
4. Dubbelklicka på formulärets bakgrund samt respektive knappar för att skapa "skalkod" för metoder. Fyll sen i koden nedan som saknas.

Nedan visas all kod för testformuläret.

```
Option Explicit
Private objSkivor As SkivorDCOM.Read

Private Sub cmdAvsluta_Click()
    Unload Me
End Sub

Private Sub Command1_Click()
    Dim adoRS As ADODB.Recordset

    Set adoRS = objSkivor.GetAlbum

    List1.Clear      'Rensa listruta

    While Not adoRS.EOF
        List1.AddItem adoRS.Fields("artist") & " - " & adoRS.Fields("titel")
        adoRS.MoveNext
    Wend
End Sub

Private Sub Command2_Click()
    Dim adoRS As ADODB.Recordset
    Dim strArtist As String

    strArtist = Text1.Text

    List1.Clear      'Rensa listruta
```

<sup>6</sup> Jag brukar skapa testprojekt för komponenter som heter samma sak som komponentens projekt samt där jag lägger till ordet "Klient" som suffix. I detta fall så är klienten lokal och kommer inte fungera utan ändringar om det ska användas på en annan dator (därför suffixet "Lokalt" också ☺).

```
If strArtist <> "" Then
    Set adoRS = objSkivor.GetAlbumForArtist(strArtist)

    If adoRS.BOF And adoRS.EOF Then
        List1.AddItem "(album för artist [" & strArtist & "] saknas)"
    End If

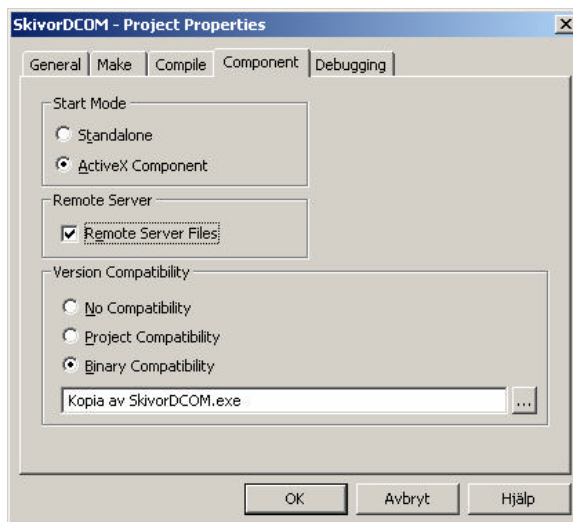
    While Not adoRS.EOF
        List1.AddItem adoRS.Fields("artist") & " - " & adoRS.Fields("titel")
        adoRS.MoveNext
    Wend
Else
    MsgBox "FEL: Du måste fylla i namnet på en artist!"
    List1.AddItem "(album för artist [" & strArtist & "] saknas)"
End If
End Sub

Private Sub Form_Load()
    Set objSkivor = New SkivorDCOM.Read 'Skapa komponent
End Sub
```

### 6.1.4 Skapa filer att placera på klient

För att kunna hitta en distribuerad VB-klient så måste vi skapa (en form av) *proxy*-objekt samt *type library*-fil (TLB-fil) som ska placeras på klientdatorn. Vi skapa därför två filer, en VBR-fil och en TLB-fil, för varje COM-server som sen ska placeras på klientdatorn (där klienten ska exekvera).

1. Visa dialogrutan med egenskaper för projekt (välj SkivorDCOM Properties... från Project-menyn). På fliken **Component** – bocka för kryssrutan **Remote Server Files**.

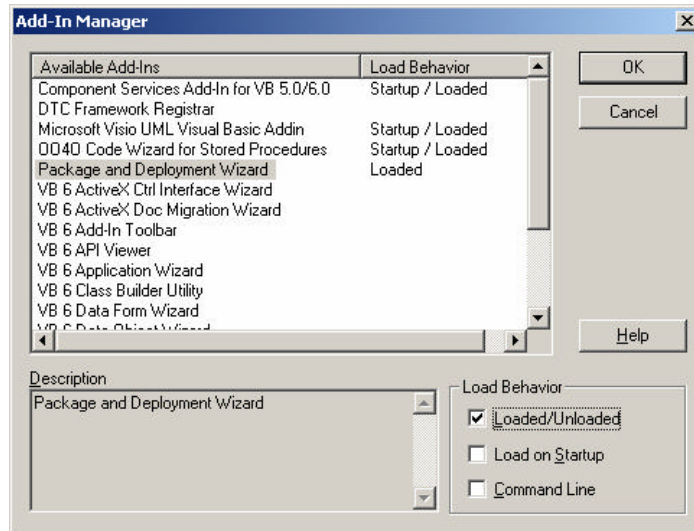


2. Skapa sen EXE-filen, d.v.s. välj **Make SkivorDCOM.EXE...** från File-menyn, för att skapa VBR- och TLB-filer.

Nu är det dags att skapa installationsprogram för klienten (för enkelhetens skull). Detta är en två stegsprocess: skapa ett installationspaket och ett installationsprogram.

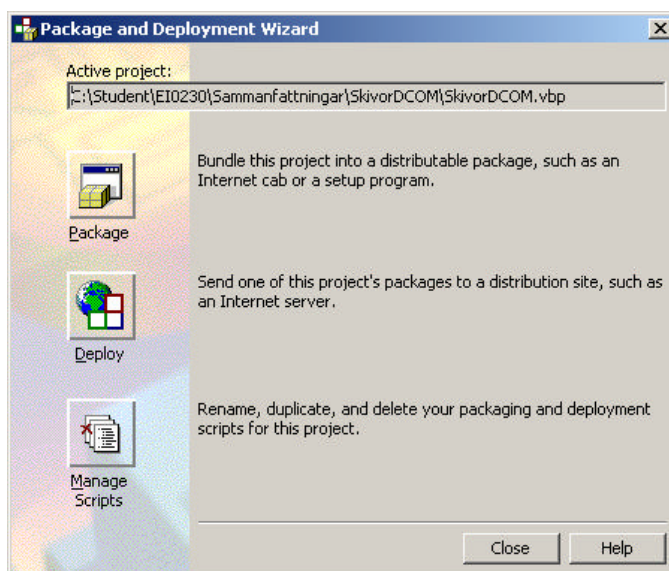
### 6.1.5 Skapa installationspaket till klienten

Vi måste först ladda Visual Basics *Package and Deployment Wizard* för att skapa installationsprogrammet.<sup>7</sup> Välja **Add-In Manager...** från menyn Add-In. Markerar där **Package and Deployment Wizard** i listrutan, bockar för **Loaded/Unloaded** och klicka på **OK** (se bild nedan).



Vi kan nu starta **Package and Deployment Wizard** från Add-Ins-menyn.

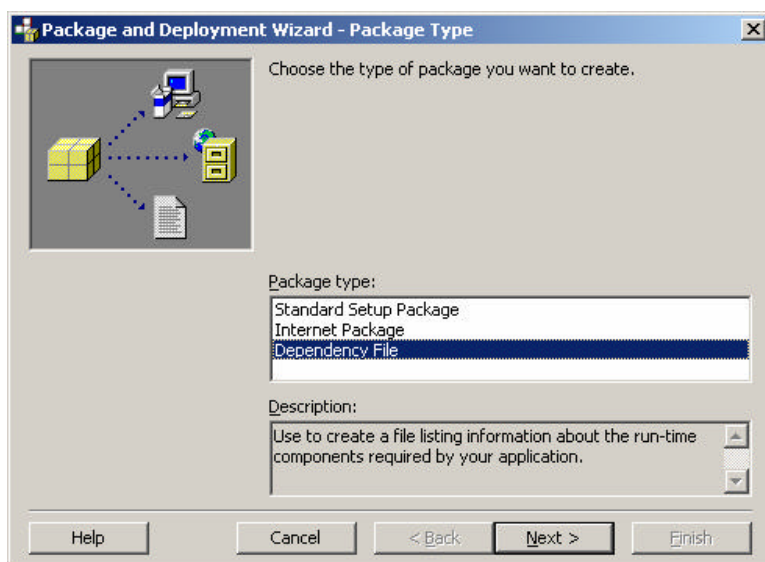
1. Starta **Package and Deployment Wizard** på Add-Ins-menyn. Om dialogruta med frågan om att spara projekt dyker upp – svara Ja.
2. I första dialogrutan (se bild nedan) klickar vi på knappen **Package**, vilket gör att projektets filer kontrolleras. Svara Ja på eventuell fråga om att kompilera om projekt.



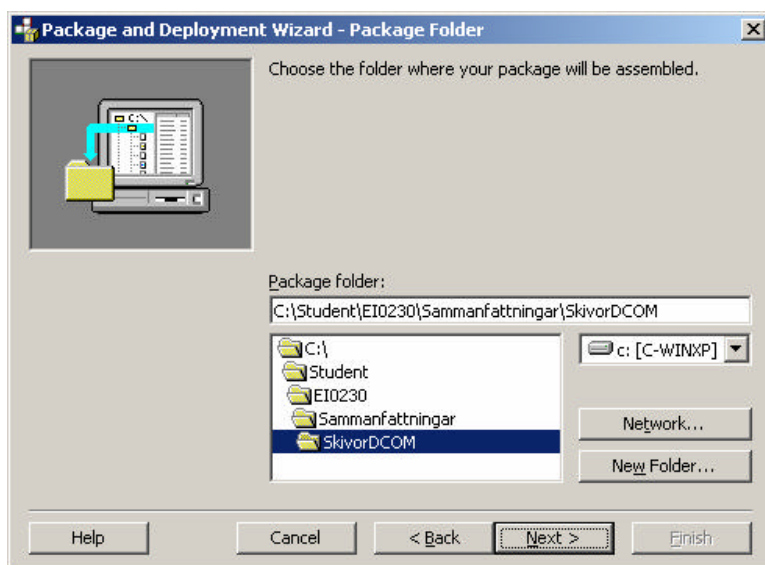
3. I nästa dialogruta (se bild nedan) markerar vi alternativet **Dependency File** och klicka på **Next>** (se bild nedan).

---

<sup>7</sup> Skulle inte Package and Deployment Wizard finnas som alternativ så kan ni installera om alla filer för VB/Visual Studio. Det finns ett alternativ i Visual Studios installationsprogram för att installera om alla filer.



4. Som standard tycker guiden att man ska placera installationsfilerna (som vi håller på att skapa) ska placeras i samma mapp som EXE-filen. Det gör vi (för enkelhetens skull<sup>8</sup>) och klickar därför på **Next->**.



5. På frågan om vi vill ha en **Remote Automation (RA) server** svarar vi **Nej**. Detta var sättet som distribuerade komponenter kunde användas **innan** DCOM, d.v.s. vi är **inte** intresserade av att använda det.

Skulle<sup>9</sup> guiden vid något tillfälle klaga på att den inte kan hitta någon fil (t.ex. AUTMGR32.EXE, AUTPRX32.DLL, CLIREG32.EXE eller RACMGR32.EXE) så finns (några av) dessa filer i mappen C:\Program\Microsoft Visual Studio\Vfp98\Distrib.src\System\<sup>10</sup> (eller i %WinDir%\System32).

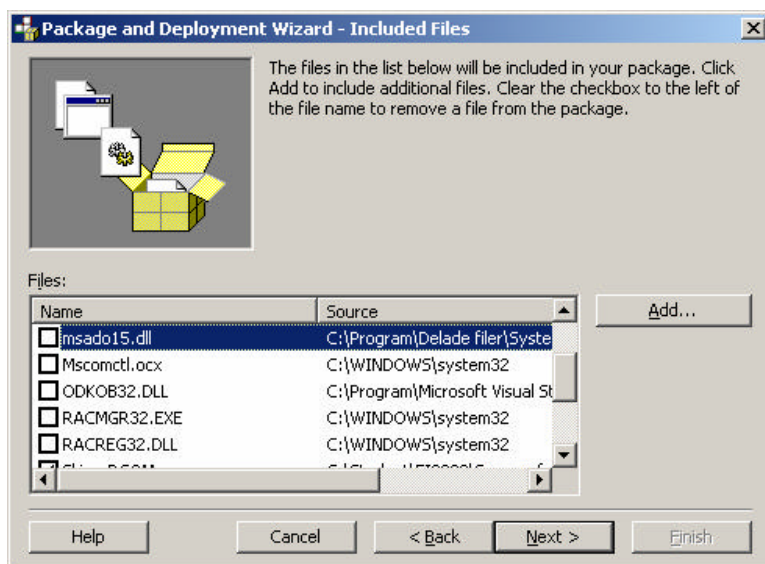
<sup>8</sup> Om vi inte skapar installationsfilerna i samma mapp som den vi utvecklar EXE-filen i så måste vi kopiera filen till en mapp där vi vill att installationsfilerna ska placeras.

<sup>9</sup> Vilket den lär göra när vi **inte** använder Visual Basic Enterprise Edition.

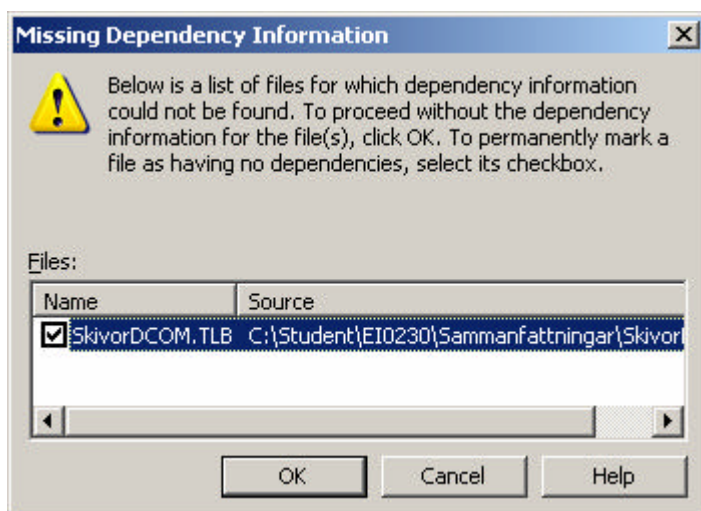
<sup>10</sup> ... eller i N:\Student\Komponenter\.

Guiden analyserar nu vilka filer vi använder (har referenser till) i projektet och föreslår att vi ska inkludera dem med installationsfilerna. Om vi t.ex. vet att ADO (med samma version) redan är installerad på klientdatorn så kan vi ta bort boken framför filen MSADO15.DLL.<sup>11</sup> Eftersom vi inte ska distribuera EXE-filen (eftersom vi ska använda DCOM) så måste vi lägga till COM-serverns VBR- och TLB-filer.

6. Ta bort boken framför msado15.dll samt klicka på knappen **Add** och bläddrar och väljer filerna SKIVORDCOM.VBR och SKIVORDCOM.TLB (välj All Files (\*.\*) i listrutan **Files of typ:** för att filerna ska synas).

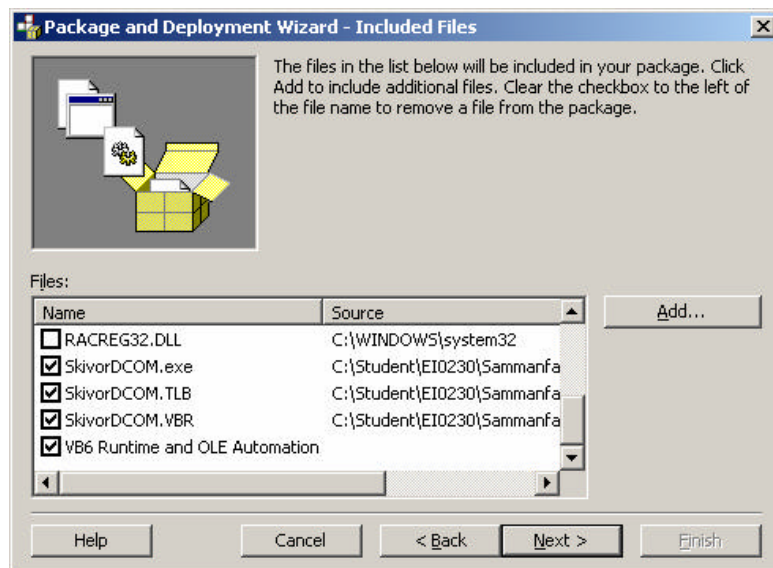


När vi valt filerna (lämpligen genom att hålla ner Ctrl-knappen när vi markerar filerna) och klickar OK så visas en dialogruta som varnar att vi saknar en *dependency* fil (DEP-fil), vilket är korrekt. Bocka för kryssrutan och klicka på OK (se bild nedan).

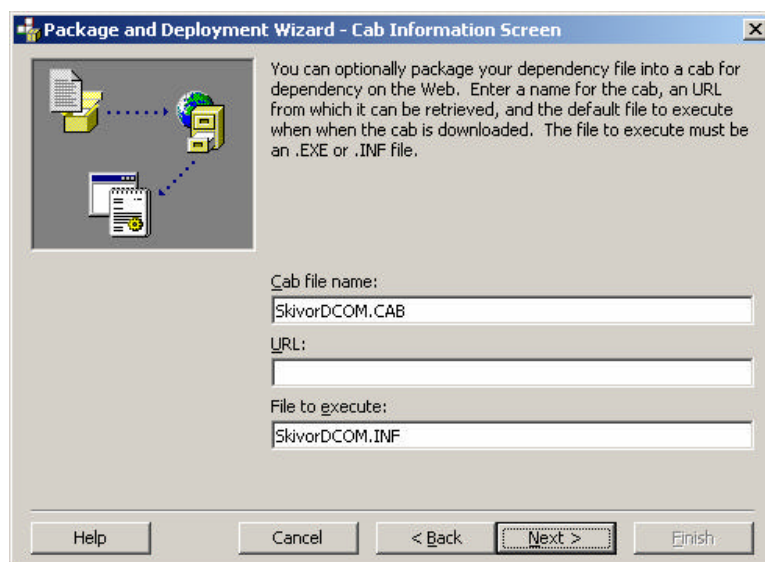


Guiden har nu lagt till ett antal filer till (utöver dom tre som fanns innan vi klickade på **Add...**). Bland filerna ser vi våra VBR- och TLB-filer och vi kan klicka på **Next>**.

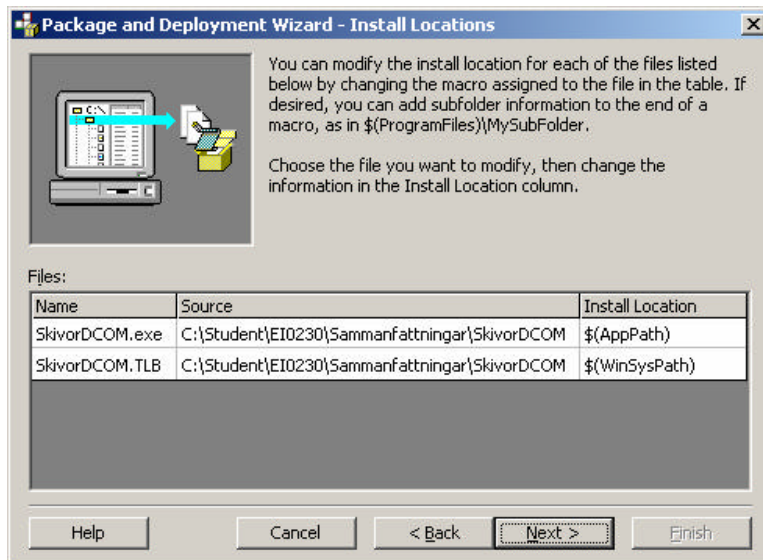
<sup>11</sup> Och för alla datorer i datorsalarna på EkI så vet vi att samma version av ADO är installerad och tar därmed bort just den boken för att undvika eventuella fel som kan uppstå!



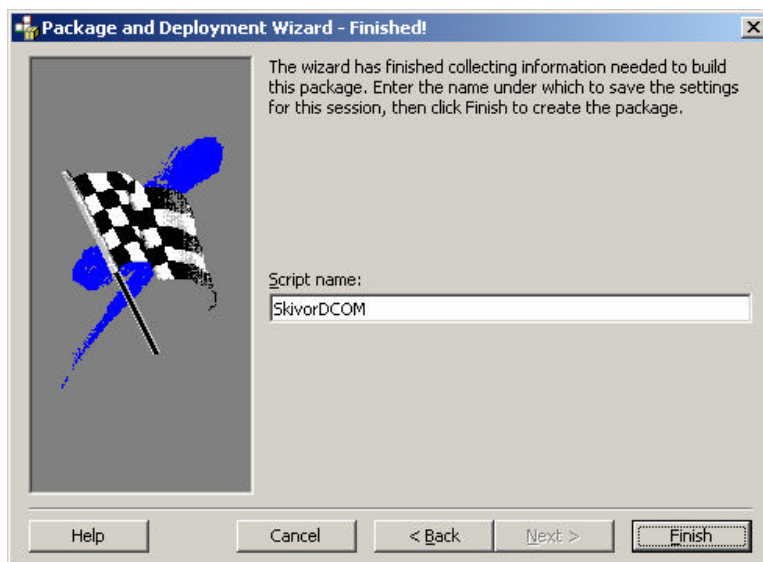
7. I nästa dialogruta (se bild nedan) låter vi namnen på filerna vara och klickar endast på **Next> ...**



8. ... liksom vi gör i dialogrutan med sökvägarna (se bild neda), d.v.s. vi klickar endast på **Next>**.



9. Vi ändrar dock namnet på skriptfilen (**Script name:**) till SkivorDCOM och klickar på **Finish**.

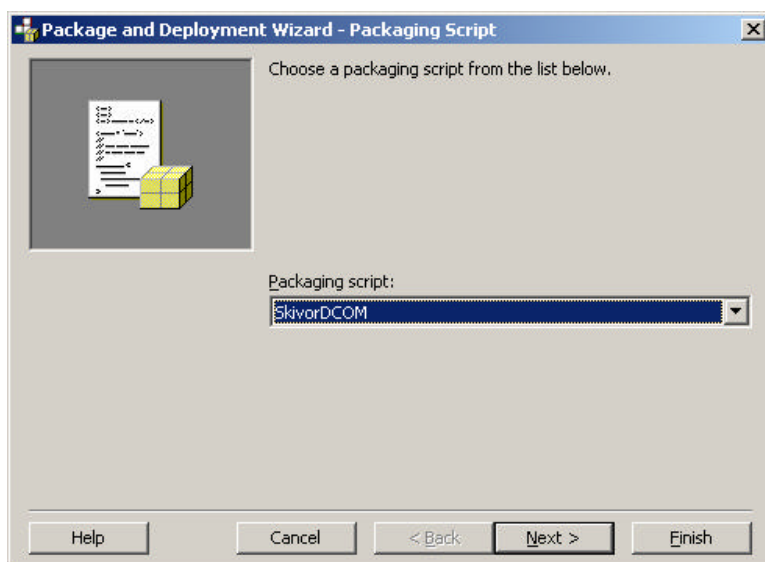


10. I dialogrutan Packaging Report klickar vi bara på **Close** för att stänga rapportfönstret och vi är tillbaka till dialogrutan vi startade med (som vi **inte** ska stänga riktigt än!).

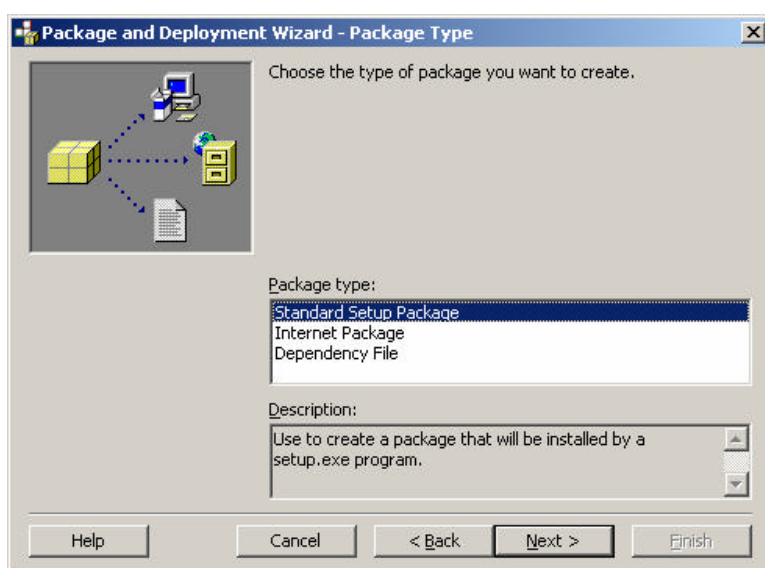
Vi är nu klara att skapa själva installationsprogrammet som vi kan köra för att installera på klienten.

### 6.1.6 Skapa installationsprogrammet

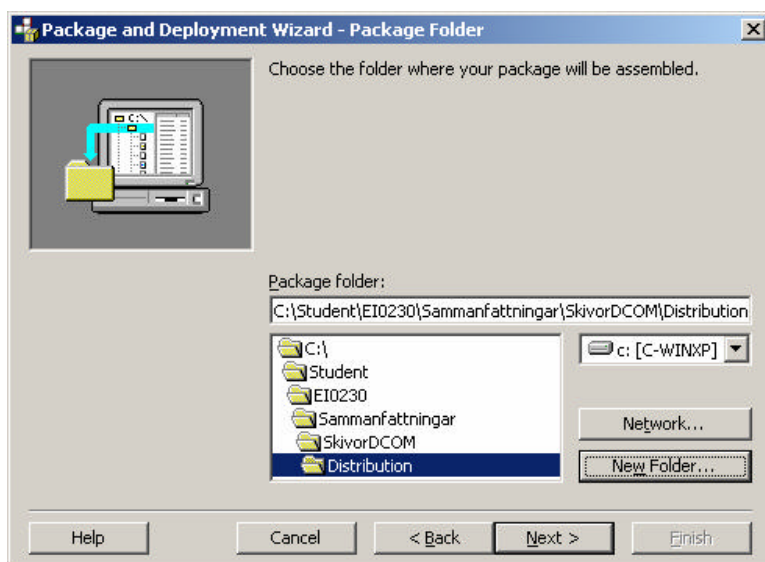
1. Klicka på knappen **Package** (igen), men nu visas en annan dialogruta (se bild nedan) än förra gången (vi klickade på knappen).



2. Välj det paket vi skapade i avsnittet ovan och klickar på **Next>** (eftersom vi antagligen inte har så många alternativa att välja från i listrutan ☺).
3. Denna gång väljer vi alternativet **Standard Setup Package** (istället för **Dependency File**) och klicka på **Next>**.

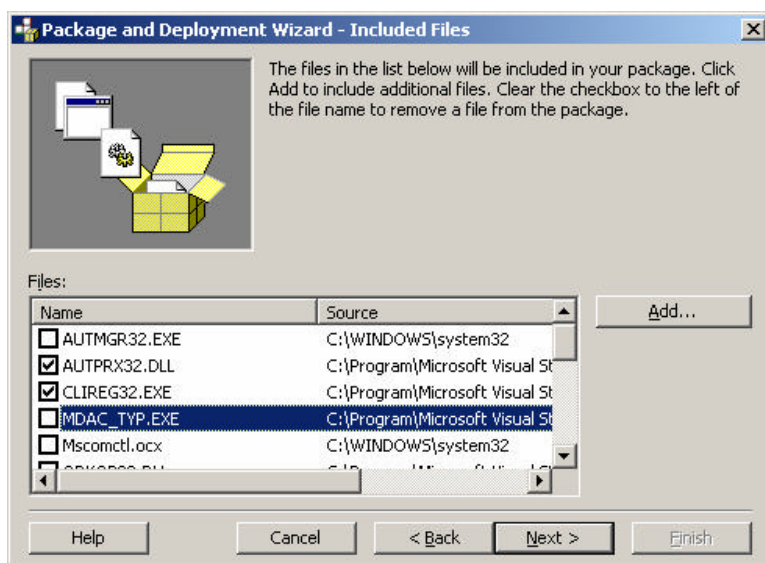


4. Guiden vill nu veta var vi ska placera installationsprogrammet. Så vi skapar en ny mapp **Distribution** (genom att klicka på **New Folder...** och fyller i namnet på den nya mappen i dialogrutan som visas) under projektets mapp.



Klickar sen på **Next>**.

5. Om det visas en dialogruta som säger att information om en fil är för gammal så klickar vi på OK för att gå vidare.
6. I nästa dialogruta visar guiden på filer som den vill distribuera med installationsprogrammet. Eftersom (eller om) vi redan har ADO installerat så väljer vi bort filen MDAC\_TYP.EXE samt eftersom vi inte behöver EXE-filen på klienten (den ska exekvera på servern) så väljer vi bort SkivorDCOM.exe.



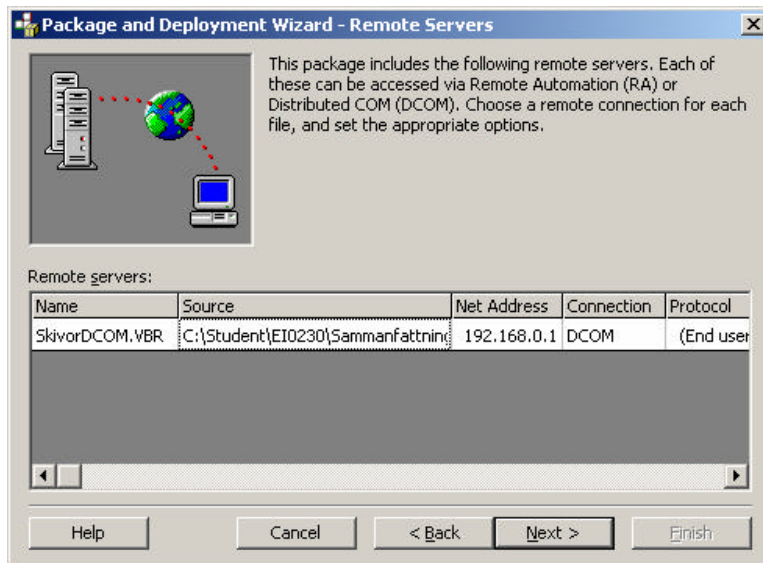
Kontrollera gärna att VBR- och TLB- filen är med och klicka på **Next>**.

7. I nästa dialogruta måste vi ange på vilken dator som COM-servern finns (kommer att finnas). Lämpligt kan vara att ange IP-adressen till datorn så att vår applikation fungerar även om DNS-server<sup>12</sup> (eller WINS-server<sup>13</sup>) saknas (eller är nere). (Denna

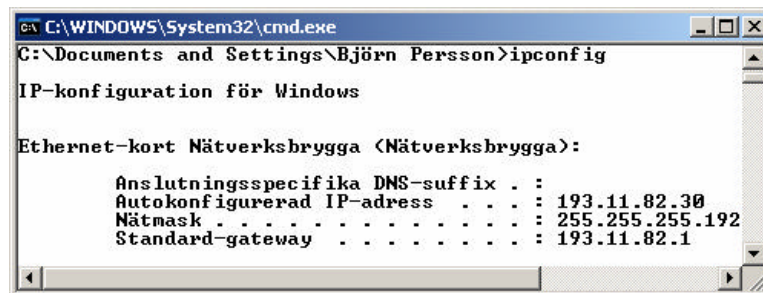
<sup>12</sup> En DNS-server används för att översätta datornamn på Internet (t.ex. bpm.eki.mdh.se) till IP-adresser (130.243.93.11).

<sup>13</sup> En WINS-server används främst i Windows NT4-domäner för att hitta datorer (t.ex. bpm) i domänen.

adress kan givetvis ändras vid ett senare tillfälle.) Fyll i adressen i rutan **Net Address** och klicka på **Next>**.

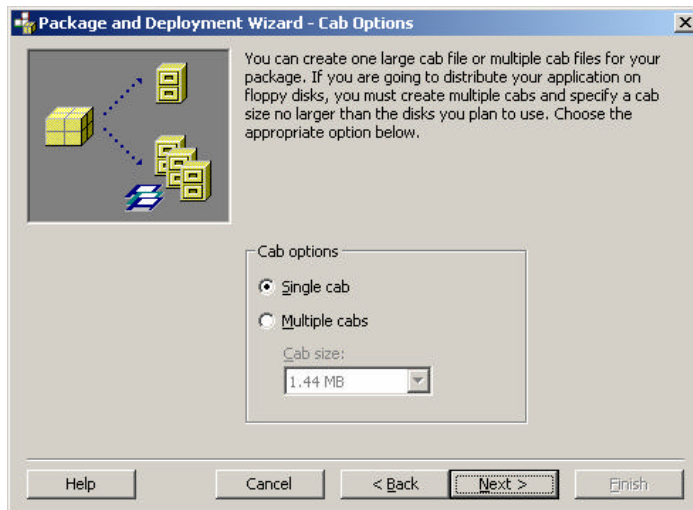


(Hemma har jag ett nätverk där servern använder IP-adressen 192.168.0.1, därav den adressen i bilden ovan. För att ta reda på IP-adressen för en dator kan man skriva t.ex. IPCONFIG i kommandotolken.

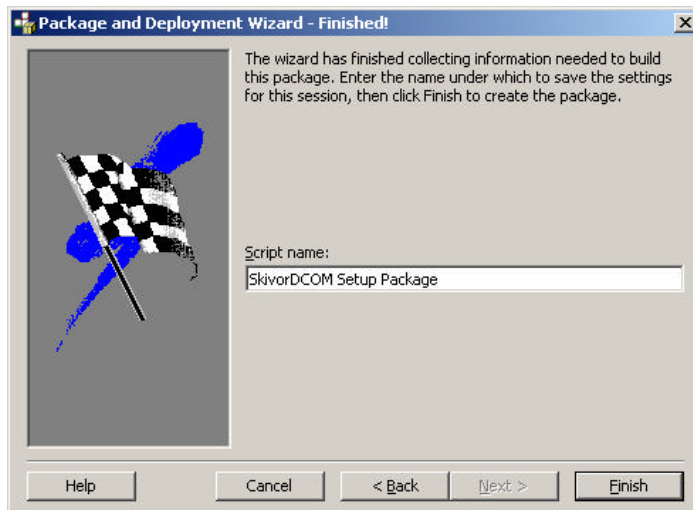


IP-adressen i bild ovan är t.ex. 193.11.82.30, d.v.s. min bärbara dator på MdH.)

8. I nästa dialogruta kan man välja om man vill ha filerna delade så att de får plats på disketter. Vi väljer **Single cab** för enkelhetens skull och klicka på **Next>**.



9. I nästa dialogruta (Installation title) godkänner vi namnet på installationens titel genom att bara klicka på **Next>**.
10. Och eftersom vi inte kommer skapa några tillägg till startmenyn så klickar vi på **Next>** i nästa dialogruta (Start Menu Items) också.
11. Vi ändrar inte heller på sökvägarna för filerna utan klicka bara på **Next>** i denna dialogruta (Install Locations).
12. Vi byter dock namnet på skriptet (i **Script name:**) till SkivordCOM Setup Package 1 och klickar på **Finish**.



### 13. Stäng rapportfönstret och **Package and Deployment Wizard**.

Om vi tittar i mappen `Distribution` (under projektets mapp) så ser vi att vi har fått ett installationsprogram (`setup.exe`) med tillhörande CAB-fil. I mappen `Support` finns även filer som krävs för att vårt VB-program ska kunna exekvera på en dator utan VB eller Visual Studio. Vi måste alltså köra `setup.exe` på vår klientdator som vi vill exekvera vår distribuerade klient ifrån. Observera att vi behöver även övriga filer när vi exekverar `setup.exe`!

Nu är det dags att konfigurera servern (den dator där vår komponent kommer exekvera).

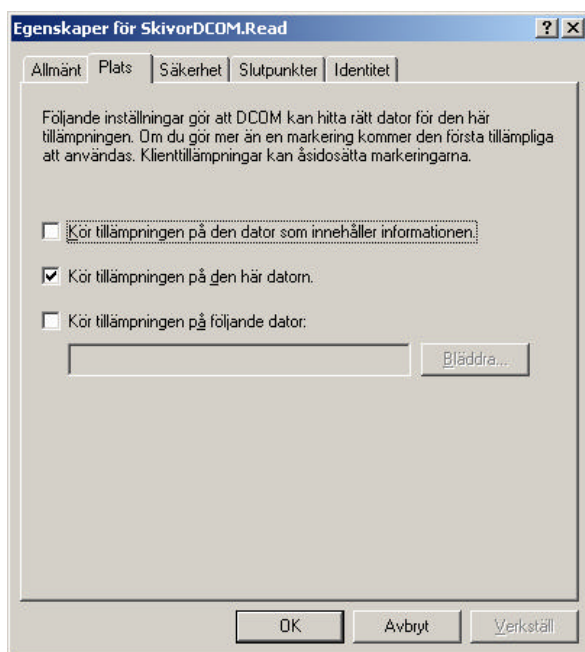
### 6.1.7 Konfigurera server för DCOM i NT4

För att inte göra exemplet mer komplext än det är så kommer vi minska på säkerheten för vår COM-server (SkivorDCOM.exe).

Observera att detta steg ibland skiljer sig en del mellan Windows NT 4 och Windows 2000/XP. Se nästa avsnitt för Windows 2000/XP sättet.

Vi ändra säkerhetsinställningarna för komponenten (på servern) och använder dialogrutan **Egenskaper för Distribuerad COM-konfiguration** för detta.

1. Starta dialogrutan genom att skriva DCOMCNFG.EXE i kommandotolken (eller i Kör-dialogrutan).
2. Kontrollera först att kryssrutan framför **Aktivera delad COM på denna dator** är förbockad på fliken **Standardegenskaper**.
3. Gå sen tillbaka till första fliken (**Program**).
4. Leta upp komponenten (eller COM-servern eftersom komponentens fulla namn används – SkivorDCOM.Read) i listan och markera den. Klicka sen på knappen **Egenskaper**.



5. Kontrollera att det finns en bock framför **Kör programmet på den här datorn** under fliken **Placering**.
6. Välj fliken **Säkerhet**.

Det är nu dags att ändra på start- och åtkomstbehörigheter, d.v.s. vem som får starta (skapa en instans av) komponent och anropa metoder i komponenten. För att minska komplexiteten i exemplet kommer vi tillåta alla (d.v.s. "hela världen") att göra detta. Observera att detta är en säkerhetsrisk!

7. Vi börjar med åtkomsträttigheterna och markera **Använd anpassade åtkomstbehörigheter** under detta alternativ. Klicka sen på knappen **Redigera** till höger för att visa dialogruta med användare med behörigheter. Klicka på knappen

**Lägg till** samt markera **Alla**, klicka på **Lägg till** och sen **OK** för att stänga dialogrutan. Upprepa samma sak för startbehörigheten.

8. Klicka på fliken **Identitet** och markera **Denna användare**. Fyll i användaridentitet (eller klicka på knappen **Bläddra** för att välja användare) och fyll i lösenord för användaren.
9. Klicka på **OK** för att stänga dialogrutan (med egenskaper för komponent) och spara.

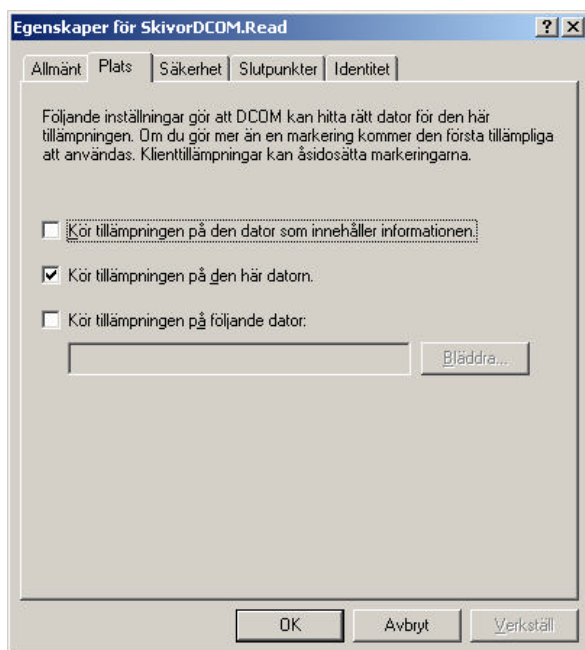
(Det finns en ganska stor chans att man kan få återvända till DCOMCNFG för att släppa lite mer på rättigheter eller liknande om vi inte lyckas med att exekvera klienten.)

### 6.1.8 Konfigurera server för DCOM i 2000/XP

För att inte göra exemplet mer komplext än det är så kommer vi minska på säkerheten för vår COM-server (SkivorDCOM.exe).

I Windows 2000/XP har DCOM blivit en *snap-in* i MMC ("integrerats" med Komponent-tjänster), d.v.s. det är inte en dialogruta som i Windows NT4.

1. Starta Komponenttjänster (t.ex. genom att skriva DCOMCNFG.EXE i kommandotolken/Kör-dialogrutan eller via Kontrollpanelen).
2. Markera grenen **Komponenttjänster** och klicka på ikonen längst till höger i verktygsfältet (Konfigurera Den här datorn). Kontrollera först att kryssrutan framför **Aktivera distribuerad COM på denna dator** är förbockad på fliken **Standardegenskaper**.
3. Expandera grenarna **Komponenttjänster**, **Datorer** och **Den här datorn** samt markera grenen **DCOM-konfiguration**.
4. Leta upp komponenten (eller COM-servern eftersom komponentens fulla namn används – SkivorDCOM.Read) i listan och markera den. Högerklicka sen på komponent och välj **Egenskaper** från menyn som visas för att öppna dialogrutan med egenskaper för komponent.



5. Kontrollera att det finns en bock framför **Kör tillämpningen på den här datorn** under fliken **Plats**.
6. Välj fliken **Säkerhet**.

Det är nu dags att ändra på start- och åtkomstbehörigheter, d.v.s. vem som får starta (skapa en instans av) komponent och anropa metoder i komponenten. För att minska komplexiteten i exemplet kommer vi tillåta alla (d.v.s. "hela världen") att göra detta. Observera att detta är en säkerhetsrisk!

7. Vi börjar med åtkomsträttigheterna och markera **Anpassa** under detta alternativ. Klicka sen på knappen **Redigera** till höger för att visa dialogruta med användare med behörigheter. Klicka på knappen **Lägg till** (för att öppna nästa dialogruta). Fyll i **Alla** i textfältet (för objektnamn) och klicka på **OK**. Kontrollera att Alla har en bock i Tillåt för Åtkomstbehörighet och klicka sen **OK** för att stänga dialogrutan. Upprepa samma sak för startbehörigheten.
8. Klicka på fliken **Identitet** och markera **Den här användaren**. Fyll i användaridentitet (eller klicka på knappen **Bläddra** för att välja användare) och lösenord för användaren som komponenten ska exekvera som.
9. Klicka på **OK** för att spara inställningar och stänga dialogrutan (med egenskaper för komponent).

(Det finns en ganska stor chans att man kan få återvända till DCOMCNFG för att släppa lite mer på rättigheter eller liknande om vi inte lyckas med att exekvera klienten.)

### 6.1.9 Installera på klienten

Vi är nu klara att installera klientdelen av COM-servern (d.v.s. VBR- och TLB- filerna). Kopiera mappen `Distribution` (med filerna `SETUP.EXE`, `SETUP.LST` och `SKIVORDCOM.CAB`) till datorn där klienten ska utvecklas.<sup>14</sup> Kör `SETUP.EXE` för att installera "klientdelen" (komponentens *type library*) av komponenten.

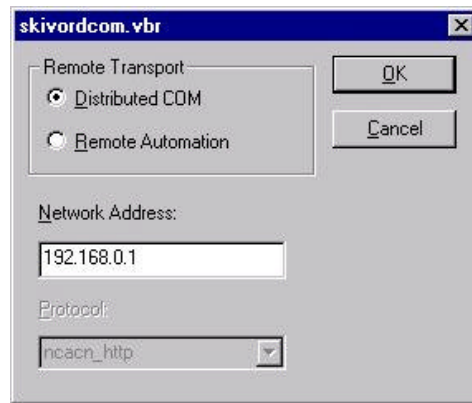
Vi måste sen köra programmet `CLIREG32.EXE` enligt följande

`C:\>CLIREG32 C:\WINNT\SYSTEM32\SKIVORDCOM.VBR`

och fylla i följande information (byt adressen mot den dator som du har installerat komponenten på)

---

<sup>14</sup> Vi utvecklar klientprogrammet på en dator med Visual Basic installerad på för att inte göra denna beskrivning längre än vad den är.



Vi är nu klara att börja utveckla klientprogrammet.

### 6.1.10 Skapa klientprogram

1. Skapa ett nytt projekt som **Standard EXE** och döp projektet till SkivorDCOMKlient.
2. Sätt en referens till **SkivorDCOM** (observera att filen vi sätter referens till är ett *type library*, en TLB-fil).
3. Deklarera en variabel (objSkivor) i början på formuläret för att hålla referensen till komponenten.

```
Public objSkivor As SkivorDCOM.Read
```

4. Dubbelklicka på formulärets bakgrund för att skapa "skalkoden" för metoden Form\_Load() och fyll i koden nedan som saknas för att skapa en instans av komponent.

```
Private Sub Form_Load()  
    Set objSkivor = New SkivorDCOM.Read  
End Sub
```

5. Placera en listruta (List1) och en knapp (btnVisaAlbum) på formuläret.
6. Dubbelklicka på knappen och lägg till följande kod.

```
Private Sub btnVisaAlbum_Click()  
    Dim adoRS As ADODB.Recordset  
  
    'Hämta album  
    Set adoRS = objSkivor.GetAlbum  
  
    'Lägg till artister och dess album till listruta  
    List1.Clear  
    'Rensa listruta  
    While Not adoRS.EOF  
        List1.AddItem adoRS!Artist & " - " & adoRS!Titel  
        adoRS.MoveNext  
    Wend  
  
    Set adoRS = Nothing  
End Sub
```

7. För att testa nästa metod i komponenten behöver vi även en textruta (txtArtist) och en knapp (btnVisaForArtist) till. Koden bakom den knappen blir följande.

```
Private Sub btnVisaForArtist_Click()  
    Dim adoRS As ADODB.Recordset  
  
    If txtArtist.Text <> "" Then  
        Set adoRS = objSkivor.GetAlbumForArtist(txtArtist)  
  
        List1.Clear          'Rensa listruta  
  
        'Om inga poster...  
        If adoRS.BOF And adoRS.EOF Then  
            List1.AddItem "Artist saknas..."  
        Else  
            While Not adoRS.EOF  
                List1.AddItem adoRS!Artist & " - " & adoRS!Titel  
                adoRS.MoveNext  
            Wend  
        End If  
        Set adoRS = Nothing  
    Else  
        MsgBox "Du måste ange artist", vbExclamation  
    End If  
End Sub
```

### 6.1.11 Provkör programmet

Förhoppningsvis så ska allt gå smärtfritt – albumen ska visas i listrutan. Men om vi får ett felmeddelande som säger något i stil med att vi ”refererade till minnesadressen 0x00000000” (eller XP vill skicka fel till Microsoft) så måste vi göra följande p.g.a. en bugg i Visual Basic 6. Redigera filen SKIVORDCOM.VBR och ändra två 0:or till 4:or för registernycklarna ProxyStubClsid och ProxyStubClsid32 (se bild nedan).

```
i} = Read  
i}\ProxyStubClsid = {00020424 0000-0000-C000-000000000046}  
i}\ProxyStubClsid32 = {00020424 0000-0000-C000-000000000046}  
i}\Typelib = {B14050DA-BF14-11D4-8876-00C04F019594}
```

Sen måste vi köra CLIREG32.EXE med VBR-filen som parameter igen för att spika ändringarna i registret.<sup>15</sup>

## 6.2 Visual C++

För att vi ska kunna skapa komponenter i Visual C++ måste vi även skapa DLL-filen för *proxy/stub* utöver själva komponenten. Vi behöver inte, som i VB, skapa någon VBR- eller TLB-fil.

Utvecklingen av komponenten skiljer sig i övrigt inte från utvecklingen av en lokal komponent. Så för att inte göra detta exempel för komplext så gör vi om vår slumpgenerator från tidigare C++-exempel.

### 6.2.1 Skapa komponenten

I detta exempel utgår vi ifrån komponenten från kapitel 4, *KompTest.Slump*, d.v.s. vi kommer använda samma kod. Men vi skapar ett nytt projekt som en EXE-fil med namnet

<sup>15</sup> Enligt Microsoft support så måste man köra CLIREG32.EXE två gånger. Första gången med Remote Automation markerat och andra gången med Distributed COM markerat för att inställningarna ska bita.

KompTestDCOM och en komponent med namnet Slump. Sen lägger vi till metoden Slumpa, implementerar metoden och kompilerar komponenten.

Glöm inte testa komponenten lokalt!

### 6.2.2 Skapa DLL för *proxy/stub*

När vi skapat och testat komponenten lokalt så är det dags att skapa DLL-filen med proxy- och stub-objekten. I Visual C++ så skapas en s.k. *make*-fil med direktiv (kommandon) för hur den ska skapas, d.v.s. vi bara ”kör” *make*-filen för att skapa DLL:en.

*Make*-filen för *proxy/stub*-DLL:en finns i samma mapp som komponentens källkodsfiler. Filen heter samma som projektet med tillägget ”ps” och har filändelsen MK (t.ex. KOMPTESTDCOMPS.MK). För att skapa DLL-filen använder man programmet NMAKE.EXE och skriver följande i kommandotolken.

```
nmake komptestdcomps.mk
```

NMAKE skapar en DLL-fil men samma namn som MK-filen (fast med filändelsen DLL ☺).

### 6.2.3 Registrera DLL för *proxy/stub*

*Proxy/stub*-DLL:en måste registreras både på datorn där komponenten ska exekvera (servern) och klientdatorn där programmet som använder komponenten finns. Detta göra man genom att köra REGSVR32 på bägge datorer enligt följande.

```
regsvr32 komptestdcomps.dll
```

### 6.2.4 Konfigurera komponent med DCOMCNFG.EXE

För att konfigurera åtkomst, säkerhet, m.m. av komponent så använder man programmet DCOMCNFG.

1. Starta DCOMCNFG från kommandotolken (eller Kör... på startmenyn).
2. Kontrollera att kryssrutan **Aktivera delad COM på denna dator** är förbockad på fliken **Standardegenskaper**.
3. Leta upp komponenten (KompTestCpp.Slump) på fliken **Program** och klicka på **Egenskaper**.
4. Klicka på fliken **Säkerhet**. Under starträttigheter – markera alternativet **Använd anpassade...** och klicka på knappen **Redigera**. I nästa dialogruta klickar man på **Lägg till** samt i nästa dialogruta väljer man **Alla** och klickar på **Lägg till**. Stäng de två dialogrutorna genom att klicka på **OK** två gånger. Upprepa samma sak för åtkomsträttigheter.
5. Klicka på fliken **Identitet**. Markera alternativet **Denna användare** och fyll i användaridentitet och lösenord.

### 6.2.5 Skapa klientprogram

När vi registrerat DLL:en för *proxy/stub* på både server och klient kan vi börja utveckla klientprogrammet. Först behöver vi kopiera definitionen av komponenten till samma mapp

som vi utvecklar klientprogrammet i. Definitionen finns i *header*-filen med samma namn som komponentens projekt. Sen skapar vi ett nytt projekt av typ **Win32 Console Application**.

Öppna filen STDAFX.H och lägg till följande rad ovanför de redan inkluderade filerna och stäng sen filen.

```
#define _WIN32_DCOM
```

Lägg till följande rader under de redan inkluderade filerna.

```
#include <atlbase.h>  
#include <iostream.h>  
#include "KompTestDCOM.h"
```

Vi deklarerar några variabler och initierar COM-miljön. Sist deklarerar vi variabler av några av COM:s strukturer som vi använder som parametrar till CoCreateInstanceEx().

API-funktionen CoCreateInstanceEx() används för att skapa objekt på annan dator (*remote server*). Till skillnad från CoCreateInstance() så kan man begära att få gränssnittspekare till fler än ett gränssnitt samtidigt. Som parametrar till CoCreateInstanceEx() så skickar man strukturer som innehåller information om bl.a. vilka gränssnittspekare man vill ha. Genom att använda CoCreateInstanceEx() så kan man även spara tid då man slipper göra flera anrop över nätverket när man hämtar flera gränssnittspekare på en gång.

```
int main(int argc, char* argv){  
    long slumptal = 0;  
    int i = 0;  
    CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);  
    CComPtr<ISlump> pSlump = NULL;  
  
    //Skapa strukturer att bifoga CoCreateInstanceEx nedan  
    COSERVERINFO csi = {0, L"130.243.93.11", NULL, 0}; //Parameter 2 är andra datorn  
    MULTI_QI qi = {&__uuidof(ISlump), NULL, S_OK};  
  
    CoCreateInstanceEx(__uuidof(Slump),  
                       NULL,  
                       CLSCTX_REMOTE_SERVER,  
                       & csi,  
                       1,  
                       &qi);
```

Här näst måste vi "plocka ut" gränssnittspekaren från dessa COM-strukturer och det sker med följande kod.

```
//Hämta "pekaren" till objektet från strukturen qi och  
// koppla pekaren till pekarvariabel med Attach()  
pSlump.Attach(static_cast<ISlump*>(qi.pItf));
```

Sen är det dags att anropa metoden i komponenten – tänk på att vår slumptalsgenerator inte fungerar riktigt bra utan att pausa i en sekund. Och för att göra det lite "roligare" så loopar vi 10 gånger.

```
for(i = 0; i < 10; i++){  
    hr = pSlump->Slumpa(10, &slumptal);  
    cout << "Slumptalet ar: " << slumptal << endl;  
    Sleep(1000);  
}
```

Sist av allt så städar vi upp lite. Det kan uppstå fel om man glömmer att sätta variabeln med gränssnittspekaren till NULL.

```
pSlump = NULL;  
CoUninitialize();  
return 0;  
}
```

Innan vi kör programmet så måste vi lägga till adressen till datorn där komponenten finns i registret.

## 6.2.6 Konfigurerar klient för var komponent finns

Vi behöver köra DCOMCNFG även på klienten för att registrera i registret var komponenten finns (ska skapas och exekvera).

1. Starta DCOMCNFG i kommandotolken.
2. Markera komponenten i listan på fliken **Program** och klicka på **Egenskaper** (denna gång behöver vi inte bry oss om fliken **Standard egenskaper** eftersom komponenten inte ska exekvera på klienten).
3. Klicka på fliken **Placering**, bocka för **Kör programmet på den här datorn** och fyll i adressen till datorn med komponenten i textrutan.
4. Stäng dialogrutorna genom att klicka på **OK** två gånger.

Kör klientprogrammet!

## 6.2.7 Att tänka på om DCOM

När man skapar och använder komponenter på en annan dator så ska man undvika att förstöra objektet i onödan. Generellt sätt så är det mer effektivt att hålla tag i gränssnittspekaren till komponenten (eller snarare dess *proxy*-objekt) än att ständigt behöva upprätta en förbindelse mellan de två datorerna.

Deklarera därför variabler för distribuerade komponenter så globalt som möjligt (men ändå så lokalt som möjligt ☺). Bestäm också när komponenter ska skapas, t.ex. i `Form_Load()` eller första gången de används.

## 7 Microsoft Transaction Server

Microsoft Transaction Server (MTS) är en utökning av COM, d.v.s. ger ytterligare tjänster till komponenter och kompletterar därmed de tjänster som COM ger. Att använda MTS innebär att utvecklare av bl.a. distribuerade applikationer inte behöver skriva kod för den funktionalitet som dessa tjänster ger. Programmerare kan därmed koncentrera sig på att skriva kod för affärslogiken. Utveckling av komponenter som ska exekvera i MTS sker på ett lite annorlunda sätt än för lokala komponenter – vilket kommer behandlas i detta och nästa kapitel.

I COM+ har MTS integrerats med COM (därav COM+) och refereras inte till som en separat programvara längre (även om jag kommer använda termen MTS i denna sammanfattning för att mena både MTS och COM+). Några av de tjänster som MTS och COM+ ger är:

- Transaktionshantering
- Surrogatprocess åt *in-process* komponenter (DLL:er) som exekverar på en server.
- Object Request Broker (ORB)
- Säkerhetssystem

Hur man använder tjänsterna i Visual Basic och Visual C++ beskrivs i detta kapitel och nästa kapitel visar på ett exempel på komponenter som exekverar i MTS.

### 7.1 Transaktioner i MTS

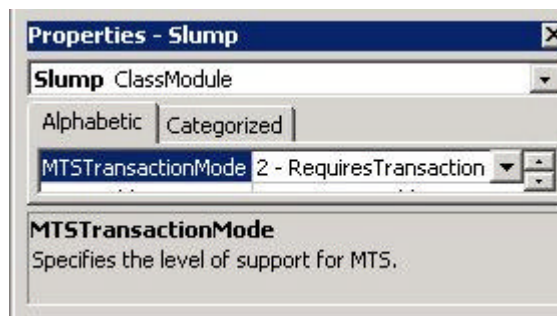
Transaktioner bör (inte bara i MTS) stödja fyra egenskaper (kallade ACID), d.v.s. transaktioner bör vara:

- **Atomära** (*atomicity*) – alla uppdateringar av t.ex. tabeller ska ske eller inga alls.
- **Konsekventa** (*consistency*) – resultatet av två (eller fler) transaktioner ska vara det samma oavsett i vilken ordning transaktionerna avslutas.
- **Isolerade** (*isolation*) – en transaktion ska inte påverka en annan.
- **Beständiga** (*durability*) – förändringar gjorda i transaktioner ska bestå, d.v.s. inte försvinna efter t.ex. ett datorhaveri.

#### 7.1.1 Visual Basic

För att komponenten ska stödja transaktioner (eller inte) i Visual Basic använder man sig av egenskapen `MTSTransactionMode` i egenskapsfönstret för klassen (se bild till höger). Denna egenskap kan ändras när som helst och kan vara en av följande:

- **NoTransactions** – objekten kommer inte stödja transaktioner.
- **RequiresTransaction** – objekten kräver en transaktion. Om anropande klient har skapat en transaktion så kommer objekt av klassen att ingå i klientens transaktion. Skulle klienten inte ingå i en transaktion så kommer en ny transaktion att skapas som objektet kommer ingå i.
- **UsesTransaction** – objekten kan använda en transaktion. Om anropande klient har skapat/ingår i en transaktion så kommer objekt av klassen att ingå i klientens transaktion. Skulle klienten inte ingå i en transaktion så kommer ingen ny transaktion



att skapas – d.v.s. objekten kommer inte ingå i en transaktion. Komponenter som exekverar i MTS och uppdaterar en eller flera tabeller bör använda detta alternativ eller det ovan.

- **RequiresNewTransaction** – objekten kräver en egen transaktion. Även om anropande klient ingår i en transaktion så kommer objektet att skapa en egen transaktion.

Om ett COM-objekt i MTS ska uppdatera en tabell så bör den minst stödja **UsesTransaction** så att den fungerar enligt ACID om den skulle ingå i en transaktion (som t.ex. uppdaterar flera tabeller).

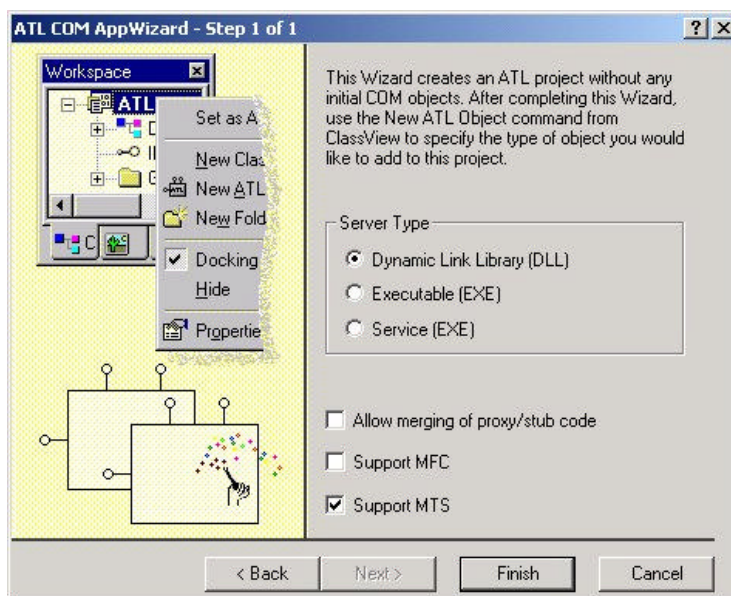
Standardinställningen, **NotAnMTSObject**, innebär att COM-objekten inte kommer att utnyttja MTS, d.v.s. den ska användas som en fristående komponent. Detta ska inte förväxlas med alternativet **NoTransactions** som innebär att komponenten ska exekvera i MTS men utan stöd för transaktioner.

Nu återstår bara en del kod för att ärva och implementera gränssnittet `IObjectContext` samt att använda två (eller tre) metoder i kontextobjektets gränssnitt `IObjectContext` (se följande stycken).

(Skulle ni få felmeddelandet *Run-time error: 430 – No such interface supported* så beror det [antagligen] på att databasen inte stödjer transaktioner. Ni får alltså nöja er med att inte använda transaktioner eller att välja **UsesTransaction**.)

## 7.1.2 Visual C++

I Visual C++ måste man tänka sig för lite när man skapar nya projektet (med **ATL COM AppWizard**) om man vill att komponenterna ska utnyttja MTS tjänster. När COM-klasser skapas (m.h.a. **ATL Object Wizard**) så väljer man om komponenten ska utnyttja transaktioner i MTS genom att bocka för alternativet **Support MTS** (se bild nedan).



Det finns dock inte någon listruta, som i Visual Basic, att välja hur komponenterna ska stödja transaktioner. För detta krävs att man ändra manuellt i IDL-filen för komponenten. Ändringen görs genom att lägga till en av konstanterna `TRANSACTION_NOT_SUPPORTED`,

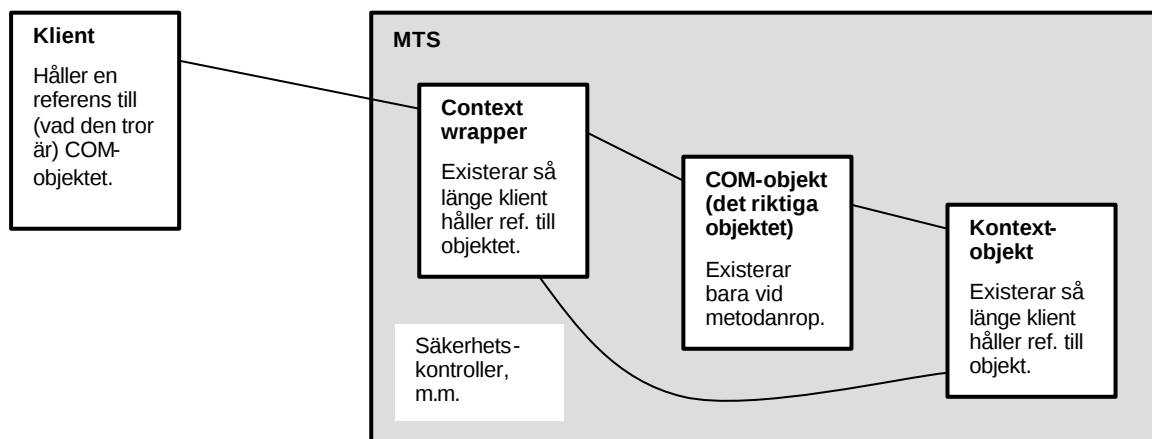
TRANSACTION\_REQUIRED, TRANSACTION\_SUPPORTED eller TRANSACTION\_REQUIRES\_NEW innan deklarationen av COM-klassen i IDL-filen (se exempel nedan).

```
[  
    uuid(CF5D2988-8F19-4F9C-ABD6-D56CC381B291),  
    helpstring("MTSKlass Class"),  
    TRANSACTION_REQUIRED  
]  
coclass MTSKlass  
{  
    [default] interface IMTSKlass;  
};
```

Glöm inte kommatecknet på raden ovanför!

## 7.2 MTS som ORB och surrogatprocess

MTS kan lägga sig i (*intercept*) skapandet av komponenter, d.v.s. ta fullständig kontroll över skapandet, exekverandet och förstörandet av komponenten. När ett COM-objekt skapas i MTS så skapas även två objekt till (som faktiskt är COM-objekt de också): ett **context wrapper** och ett **kontextobjekt** (*context object*). Wrapper-objektet har samma gränssnitt som COM-objektet samt placeras mellan klienten och själva COM-objektet som begärdes, d.v.s. klienten får en gränssnittspekare till wrapper-objektet som i sin tur refererar till COM-objektet. Därmed går alla anrop via wrapper-objektet och MTS kan stödja funktionalitet som **Just-In-Time (JIT) aktivering** och säkerhetskontroller av anrop mot objekt (m.h.a. kontextobjektet). Kontextobjektet innehåller information om objektet så som transaktionsstatus och säkerhetsinställningar.



Figur 1 - När en klient skapar ett COM-objekt i MTS så skapas även ett wrapper-objekt och ett kontextobjekt för varje COM-objekt. Klienten erhåller en gränssnittspekare till, vad den tror är, COM-objektet men som faktiskt är en gränssnittspekare till wrapper-objektet. Först vid ett metodanrop så skapas det "riktiga" COM-objektet och när metoden exekverat fördivgt så förstörs/inaktiveras objektet. Säkerhetskontroller vid metodanrop görs av MTS genom att kontrollera säkerhetsinställningar i kontextobjektet.

JIT-aktivering av komponenter i MTS används för att spara resurser på applikationsserverar. Detta gör MTS genom att inte skapas själva COM-objektet förrän klienten anropar en metod i objektet. När metoden avslutas inaktiveras/förstör MTS objektet för att återta de resurser som objektet använde. Själva wrapper-objektet existerar så länge som klienten har en referens till COM-objektet (d.v.s. wrapper-objektet ☺). Med JIT-aktivering behöver en applikationsserver

som tjänar 5000 klienter kanske endast ha ca. 500-1000 aktiverade objekt och sparar därmed på resurserna (minne, m.m.). Servern blir därmed mer skalbar, d.v.s. kan tjäna fler klienter samtidigt.

I.o.m. Windows 2000 och COM+ så stödjer MTS **poolning av objekt** (*object pooling*).<sup>16</sup> Objektpoolning innebär att objektet inte förstörs direkt utan placeras i en pool där de förvaras tills nästa gång ett anrop till ett objekt av klassen sker. På detta sätt kan man spara tid (och resurser?) för objekt som t.ex. har en krävande initialiseringsprocess (eftersom MTS slipper skapa dessa objekt) om de finns i objektpoolen. De inaktiverade objekten stannar dock inte hur länge som helst i poolen utan förstörs efter en viss tid om de inte blivit använda.

För att JIT-aktivering och objektpoolning ska fungera så måste komponenter vara **tillståndslösa** (*stateless*), d.v.s. komponenterna får inte förutsätta att värden på medlemsvariabler kommer behållas mellan metodanrop till komponenten.<sup>17</sup> "Tillstånden" måste alltså skickas med varje metodanrop eller läsa från t.ex. en databas. JIT-aktivering ställer också krav på komponenterna att det ska tala om för MTS när COM-objekten kan inaktiveras och därmed förstöras eller placeras i objektpoolen (se *Användande av IObjectContext* nedan). Det kan finnas tillfällen när tillståndslösa komponenter inte är ett alternativ, t.ex. då det är för tids- eller resurskrävande att ständigt behöva skapa objekt för varje metodanrop. I sådana fall så är komponenter som behåller sina tillstånd mellan metodanrop den enda lösningen. Denna typ av komponenter bör dock användas med stor försiktighet.

---

### 7.3 Skapa komponenter för exekvering i MTS

MTS ställer en del krav på komponenter som ska exekvera och utnyttja tjänster i MTS. Ett av kraven är att alla komponenter ska vara *in-process* komponenter, d.v.s. DLL:er. COM-klasserna ska också implementera gränssnittet `IObjectControl` så att JIT-aktivering och objektpoolning fungerar. Komponenterna bör även använda metoderna `SetComplete()` och `SetAbort()` i sitt kontextobjekts gränssnitt `IObjectContext` för att meddela MTS att de är klara med exekvering av metoder.

En fördel (?) i MTS är att man kan glömma trådmodeller för MTA (och därmed synkronisering) eftersom komponenter som ska exekvera i MTS bör betraktas som om dom skulle användas av endast en användare. D.v.s. lämplig trådmodell för komponenter i MTS är *Apartment* (även komponenter gjorda i Visual C++).

I COM+ så bör vi dock använda trådmodellen NTA. Ett problem med detta är att denna trådmodell inte fanns när Visual Studio 6 (d.v.s. VB6 och VC++6) skapades. Vi kan alltså inte ange denna trådmodell med hjälp av Visual Studio 6. För övrigt så stödjer inte VB denna trådmodell, men i VC++ kan vi "hacka" i IDL-filen själva och vi kan även hacka i registret.

Ytterligare en skillnad mellan MTS och COM+ är att COM+ använder aktiviteter (*activities*). Aktiviteter används för att hantera synkronisering av komponenter. D.v.s. vi kan använda komponenter för trådmodellen MTA (d.v.s. C++-komponenter) och låta COM+ hantera synkronisering. D.v.s., vi bör utveckla komponenter som ska exekvera i COM+ med trådmodellen MTA.

---

<sup>16</sup> Med den lilla haken att endast komponenter utvecklade i C++.

<sup>17</sup> Detta kan kännas som en motsägelse till objektorienterad programmering där objektens tillstånd är en del av deras identitet.

### 7.3.1 Implementera gränssnittet `IObjectControl`

Med JIT-aktivering så skapas alltså inte COM-objektet förrän vid första anropet av en av objektets metoder och objektet förstörs så fort metoden slutat exekvera. Om objektet även stödjer objektpoolning så kanske det inte behöver skapas utan endast aktiveras (om det finns ett sådant objekt i objektpoolen) när objektet behövs samt inaktiveras och placeras i objektpoolen när det inte behövs längre. För att detta ska fungera behöver vi ett sätt som MTS kan meddela objektet att det aktiveras eller inaktiveras. Detta kan ske genom att komponenten implementerar gränssnittet `IObjectControl`.

Gränssnittet `IObjectControl` innehåller tre metoder, som ingen av dem tar en parameter:

- `Activate()` – anropas då objektet aktiveras, antingen för ett nyskapat objekt eller ett objekt från objektpoolen.
- `CanBePooled()` – anropas när objekt ska förstöras. Svara falskt om objektet inte stödjer objektpoolning, annars sant.
- `Deactivate()` – anropas då objektet inaktiveras eller förstörs.

Dessa tre metoder anropas av MTS, bl.a. då objekt aktiveras vid ett metदानrop och då objektet inaktiveras (inte behövs längre). Metoden `CanBePooled()` returnerar alltid falskt under COM/MTS då objektpoolning inte stöds under COM/MTS. I.o.m. COM+ så kan metoden returnera även sant då COM+ stödjer objektpoolning, dock endast om de skapats i C++.

En av sakerna som man använder metoderna `Activate()` och `Deactivate()` till är för att hämta respektive släppa gränssnittspekare till COM-objektets kontextobjekt (se kodexempel nedan). Man bör eventuellt även placera eventuell initialiseringskod här istället för i klassens konstruktör (*constructor*) och destruktör (*destructor*). Objekt kan nämligen hamna i objektpoolen vilket leder till att dom inte förstörs och därmed utförs inte destruktorn. När objektet sen hämtas från objektpoolen så aktiveras de endast och därmed utförs inte konstruktorn då objektet återaktiveras.

#### 7.3.1.1 Visual Basic

I Visual Basic måste vi först sätta en referens till *type library* för MTS (COM+), vilket görs genom att välja **References...** från Project-menyn och bocka för **Microsoft Transaction Server Type Library (COM+ Services Type Library)**. Därefter skriver vi högst upp i klassen, som implementerar komponenten, `Implements ObjectControl` (OBS! ej `IObjectControl`!). Nu kan vi från listrutan **Object** (till vänster) i kodfönstret välja `ObjectControl` för att sen välja de tre metoderna, i tur och ordning, från listrutan **Procedure** (till höger) för att skapa skalkoden (se bild nedan).



I exemplet nedan visas kod som brukar finnas i metoderna (variabeln med referensen till `ObjectContext` förklaras under *Använda metoder i `ObjectContext`* nedan).

|                          |                                       |
|--------------------------|---------------------------------------|
| Implements ObjectControl | 'Arv från gränssnittet IObjectControl |
|--------------------------|---------------------------------------|

```
Private mobjContext as ObjectContext 'Dekl. Var. för referens t. kontextobjektet

'Implementera metoden Activate() i gränssnittet IObjectControl
Private Sub ObjectControl_Activate()
    Set mobjContext = GetObjectContext 'Hämta en referens till kontextobjektet
End Sub

'Implementera metoden CanBePooled() i gränssnittet IObjectControl
Private Function ObjectControl_CanBePooled() As Boolean
    ObjectControl_CanBePooled = False 'COM/MTS stödjer inte objektpoolning!!
End Sub

'Implementera metoden Deactivate() i gränssnittet IObjectControl
Private Sub ObjectControl_Deactivate()
    Set mobjContext = Nothing 'Släpp referens till kontextobjektet
End Sub
```

### 7.3.1.2 Visual C++

I Visual C++ är det lite lättare att implementera IObjectControl. Då man kör **ATL Object Wizard** för en ny COM-klass så kryssar man i kryssrutan **Support IObjectControl** (se bild till höger) och ATL-guiden kommer att generera metoderna (se kod nedan). Ska komponenten användas i COM+ och kan stödja objektpoolning så kan vi även kryssa i **Can be pooled**.



I exemplet nedan heter komponenten MTSKlass vilket ger klassen som implementerar komponenten namnet CMTSKlass.

```
HRESULT CMTSKlass::Activate()
{
    HRESULT hr = GetObjectContext(&m_spObjectContext);
    if (SUCCEEDED(hr))
        return S_OK;
    return hr;
}

BOOL CMTSKlass::CanBePooled()
{
    return FALSE;
}

void CMTSKlass::Deactivate()
{
    m_spObjectContext.Release();
}
```

Pekarevariabeln m\_spObjectContext har deklarerats i klassens *header*-fil enligt följande:

```
CComPtr<IObjContext> m_spObjectContext;
```

## 7.3.2 Använda metoder i IObjContext

När COM-objekt ska skapas eller aktiveras av MTS är självklart – vid metदानrop. Men hur MTS vet när objektet kan inaktiveras eller förstöras är mindre självklart. Det behövs ett sätt

att meddela MTS att objektet har exekverat färdigt sin metod. Gränssnittet `IObjectContext`, som implementeras av kontextobjekt, innehåller två metoder som man kan meddela MTS detta:

- `SetComplete()` – meddelar MTS att metoden exekverat färdigt och att ändringar i eventuella transaktioner kan genomföras (*commit*).
- `SetAbort()` – meddelare MTS att metod exekverat färdigt men att eventuella transaktioner ska avbrytas (*roll back*).

Även om en (tillståndslös) komponent aldrig deltar i en transaktion så bör den anropa antingen `SetComplete()` eller `SetAbort()` för att meddela MTS att komponenten kan inaktiveras eller förstöras. Ytterligare en metod i `IObjectContext` som vi behöver är `CreateInstance()`. Vi använder metoden för att skapa fler objekt som exekverar i MTS från ett objekt som redan exekverar i MTS. Denna metod **måste** användas **istället** för det reserverade ordet `New` eller funktionen `CreateObject()` i komponenter som redan exekverar i MTS! I COM+ så är vi inte längre begränsade till `CreateInstance()`. (Se mer under *Skapa objekt som ska exekvera i MTS.*)

### 7.3.2.1 Visual Basic

Det första vi måste göra är att sätta (eller kontrollera att det finns) en referens till MTS *type library*. Det gör vi genom att välja **References...** från Project-menyn samt leta upp **Microsoft Transaction Server Type Library**<sup>18</sup> och sätta en bock i kryssrutan framför. Sen deklarerar man en variabel av klassen `ObjectContext` för att hålla en gränssnittspekare till objektets kontextobjekt (trots att gränssnittet heter `IObjectContext`!).

Det är nu vi får användning av variabeln `mobjContext` som vi deklarerade i stycket ovan om gränssnittet `IObjectContext`. Variabeln håller en referens (gränssnittspekare) till objektets kontextobjekt som implementerar gränssnittet `IObjectContext` med metoderna `SetComplete()` och `SetAbort()`. Metodanropen placeras sist i alla metoder och alla metoder bör innehålla en "felhanterare" som ger möjlighet till att avbryta en eventuell transaktion om det skulle uppstå ett fel.

Nedan visas ett exempel på en metod (`UppdateraTabell()`) och dess felhanterare.

```
Implements ObjectControl      'Ärv från gränssnittet IObjectControl
Dim mobjContext as ObjectContext 'Dekl. variabel för referens t. kontextobjektet

Sub UppdateraTabell(ByVal Field1 as String, ByVal Field2 As Integer)
    On Error Goto UppdateraTabell_Error 'Om fel gå till felhanterare nedan...
    'Utför en uppdatering av databas

    mobjContext.SetComplete      'Tala om för MTS att det gick bra och att
    Exit Sub                    ' ev. transaktion kan sparas (commit).
                                'Avsluta metod (exekvera ej felhanterare)

UppdateraTabell_Error:        'Felhanterar för metoden
    'Städa upp lite
    mobjContext.SetAbort      'Tala om för MTS att det gick åt skogen
                                ' och att ev. transaktion ska avbrytas
                                ' (roll back).

End Sub
```

<sup>18</sup> Återigen **COM+ Services Type Library** i Windows 2000.

Observera användandet av `ByVal` för parametrar till metod då dessa värden endast skickas till komponent. Om inget returneras via en parameter bör vi alltid använda `ByVal` för att effektivisera (minska) kommunikation mellan klient och komponent.

### 7.3.2.2 Visual C++

Som vi såg i stycket ovan om gränssnittet `IObjectContext`, så genererade guiderna i ATL metoderna för gränssnittet. I metoden `Activate()` så hämtades gränssnittspekare till komponentens kontextobjekt och placerades i variabeln `m_spObjectContext`. Vi får därmed tillgång till metoderna `SetComplete()` och `SetAbort()` som vi kan anropa på samma sätt som i Visual Basic.

```
// Utför någon uppdatering av databas
//Om allt gick OK anropa SetComplete() ...
m_spObjectContext->SetComplete();

//... eller om något gick fel SetAbort()
m_spObjectContext->SetAbort();
```

I C++ används en massa if-satser för att kontrollera om metodanrop gick bra eller inte. D.v.s. anropa av `SetAbort()` bör t.ex. ske i dessa if-satser om ett metodanrop misslyckas och funktion avslutas. Vi kan givetvis även använda ett try-catch-block och anropa `SetAbort()` i catch-block.

## 7.4 Skapa objekt som ska exekvera i MTS

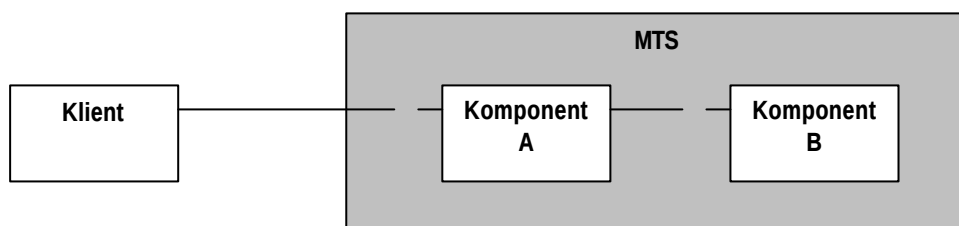
I Visual Basic skapas vanligen objekt med det reserverade ordet `New`. När man från en klientapplikation vill skapa ett objekt som är gjort för att exekvera i MTS bör (måste<sup>19</sup>) man istället använda funktionen `CreateObject()`. Därmed kommer alla instanser av objekt skapas av COM (eller MTS). Metoden har också den fördelen att den kan skapa objekt på en annan dator.

Om man vill skapa ett objekt, inuti ett objekt som exekverar i MTS, så bör (måste) man använda sig av metoden `CreateInstance()` i kontextobjektet. Om man använder sig av `New` eller `CreateObject()` så kommer inte MTS känna till objekten eller exekvera dem i samma transaktion. Detta kan leda till att t.ex. transaktioner inte fungerar riktigt som man tänkt.

Skälet till detta är att MTS är en mjukvara som installeras ovanpå operativsystemet, och därmed även COM (som är en del av operativsystemet). COM är alltså inte medveten om MTS existens utan utan MTS ersätter sökvägen till komponenten med en egen sökväg när en komponent installeras i MTS. Det är på detta sätt som MTS kan "fånga upp" metodanrop till komponenter installerade i MTS. (Vilket inte är någon bra lösning... men som tur är så löses detta i.o.m. Windows 2000/XP. ☺)

I.o.m. COM+ så har MTS integrerats med COM+, som i sin tur har integrerats med operativsystemet. Och i COM+ kan vi numera använda vilket sätt som helst (`New`, `CreateObject()` eller `CreateInstance()`) att skapa objekt, oavsett om dom exekverar i COM+ eller inte.

<sup>19</sup> Det fungerar oftast att använda `New`-operatoren. Men för att objektet ska skapas riktigt så måste man använda `CreateObject()`.



Figur 2 - När klient skapar komponent A används funktionen `CreateObject()`. Men när komponent A skapar komponent B så måste den använda metoden `CreateInstance()` i sitt context-objekt för att komponent B ska exekvera i samma kontext (och därmed samma transaktion).

## 7.5 Paket i MTS/Komponenttjänster

Komponenter som ska exekvera i MTS eller COM+ måste installeras i något som kallas paket (*packages*) respektive tillämpningar<sup>20</sup> vilket görs med det grafiska verktyget **MTS Explorer** respektive **Komponenttjänster**.

MTS Explorer och Komponenttjänster påminner mycket om varandra (då de bägge bygger på MMC), men skiljer sig bl.a. på termer som används. Att lägga till komponenter är dock mycket snarlikt. I nästa avsnitt förklaras hur vi skapar paket i MTS och avsnittet därefter hur vi skapar tillämpningar i COM+. Sist förklaras hur man lägger till komponenter i MTS/COM+.

### 7.5.1 Skapa ett paket i MTS

1. Starta *Transaction Server Explorer* (i *Microsoft Management Console*, MMC) som finns i Startmenyn under **Program->Windows NT 4.0 Option Pack->Microsoft Transaction Server**.
2. Expandera följande grenar i den vänstra delen av MMC: **Microsoft Transaction Server, Datorer, Den här datorn** och **Installerade paket**.
3. Högerklicka på grenen **Installerade paket** och välj **Nytt** och sen **Paket** (det går även att markera **Installerade paket** och sen välja **Nytt** från menyn **Åtgärd** i verktygsfältet).
4. Klicka på **Skapa ett tomt paket** (det andra alternativet används för att installera ett paket som exporterats från MTS Explorer tidigare).
5. Fyll i ett namn för paketet och klicka på **Nästa>**.
6. I dialogrutan **Ange paketidentitet** bör man välja en användare med de rättigheter som krävs för att utföra det applikationen ska göra, t.ex. för att uppdatera en databas krävs rättigheter till databasen. Här kan vi under utveckling välja Administratör som användare genom att markera **Denna användare**: och sen klicka på knappen **Bläddra** och välja användaren Administratör. Fyll i lösenordet för användaren och klicka på **Slutför**.

Under grenen **Installerade paket** skapas ett paket med det namn som valdes ovan. Expanderar vi grenen med paket så finner vi två mappar: Komponenter och Roller. I den första mappen ska vi placera våra komponenter vi vill ska exekvera i MTS. Den andra mappen har med säkerhet att göra. Vanligen anger man säkerhet för en roll i sin applikation och sen kan en administratör, för en applikationsserver (med MTS), skapa rollen och ange

<sup>20</sup> Tillämpningar kallas även applikationer (*applications*).

vilka användare och/eller grupper som ska tillhöra rollen. (Roller och säkerhet behandlas inte i denna sammanfattning.)

### 7.5.2 Skapa en tillämpningar i COM+

1. Starta *Komponenttjänster* (även den i *Microsoft Management Console*, MMC) som finns i Startmenyn under **Inställningar>Kontrollpanelen>Administrationsverktyg**.
2. Expandera följande grenar i den vänstra delen av MMC: **Komponenttjänster, Datorer, Den här datorn** och **COM+-tillämpningar**.
3. Högerklicka på grenen **COM+-tillämpningar**, välj **Nytt** och sen **Tillämpning** (det går även att markera **COM+-tillämpningar** och sen välja **Nytt** från menyn **Åtgärd** i verktygsfältet).
4. Klicka på **Nästa>** för att fortsätta.
5. Klicka på **Skapa ett tomt tillämpningsprogram** (det andra alternativet används för att installera en tillämpning som exporterats från Komponenttjänster tidigare).
6. Fyll i ett namn för paketet, markera om tillämpningen ska vara biblioteks- eller servertillämpning och klicka på **Nästa>**. En bibliotekstillämpning exekverar i samma process som klienten, d.v.s. är lämplig att använda för komponenter som används av många andra tillämpningar. En servertillämpning exekverar i en egen process, vilket är den typ som krävs för distribuerade applikationer (när klient finns på en annan dator). Exempelen i denna sammanfattning kräver en servertillämpning.
7. I dialogrutan **Ange tillämpningsidentitet** bör man välja en användare med de rättigheter som krävs för att utföra det applikationen ska göra, t.ex. för att uppdatera en databas krävs rättigheter till databasen. Här kan vi under utveckling välja **Administratör** som användare genom att markera **Denna användare:** och sen klicka på knappen **Bläddra** för att välja användare. Fyll i lösenordet för användaren och klicka på **Nästa>**.
8. Klicka på **Slutför**.

Under grenen **COM+-tillämpningar** skapas en tillämpning med det namn som valdes ovan. Expanderar vi grenen med tillämpning så finner vi två<sup>21</sup> mappar: Komponenter och Roller. I den första mappen ska vi placera våra komponenter vi vill ska exekvera i COM+. Den andra mappen har med säkerhet att göra. Vanligen anger man säkerhet för en roll i sin applikation och sen kan en administratör, för en applikationsserver (med COM+), skapa rollen och ange vilka användare och/eller grupper som ska tillhöra rollen. (Roller och säkerhet behandlas inte i denna sammanfattning.)

### 7.5.3 Lägga till komponenter i ett paket

Vi kan antingen installera komponenterna i paketet/tillämpningen genom att använda MTS Explorer/Komponenttjänster för att välja redan registrerade komponenter eller DLL-filer med icke registrerade komponenter. Men lättast är att dra och släppa DLL-filerna från t.ex. Utforskaren på/i mappen **Komponenter** i MTS Explorer/Komponenttjänster.

---

<sup>21</sup> Det finns faktiskt tre i XP. Utöver de två ovan finns en mapp Äldre komponenter.

## **7.6 Tips när man skapar komponenter för MTS/COM+**

Utveckla komponenten utanför MTS/COM+ först och testa så att komponenten fungerar felfritt. Anpassa sen komponenten för MTS/COM+ genom att implementera `IObjectContext` samt lägg till anrop av `SetComplete()` och `SetAbort()`. Glöm inte att ändra från `New-`operatorn till `CreateObject()` eller `CreateInstance()` i Visual Basic.

(Denna sida har avsiktligt lämnats blank.)

## 8 Exempel: En n-lager applikation

I detta exempel kommer vi titta på hur man implementerar en n-lager applikation. Vi kommer använda två komponenter för affärslogiklagret och två för datalagret. För presentationslagret kommer vi använda ett grafiskt gränssnitt i Visual Basic. I nästa kapitel kommer vi visa hur vi, med smärre förändringar, kan använda IIS som del i presentationslagret. En applikation i denna storlek är kanske inte fullt realistiskt att implementera med fyra komponenter – två hade antagligen räckt (en för att läsa och en för att uppdatera tabeller). Men som vi även kommer se i nästa kapitel så kommer vi att skapa två komponenter till som fungerar tillsammans med IIS för att skapa presentationslagret. De komponenter vi utvecklar i detta kapitel kommer att förbli opåverkade av detta.

Enligt Kirtland[98] ska man börja med att designa tabellerna för att sen fortsätta med data-, affärslogik- resp. presentationslagren i den ordningen. Och även om detta exempel börjar i den änden så är klasserna viktigast i den objektorienterade designen! D.v.s. det kan kanske vara bättre att börja med att modellera systemet på en högre (mer abstrakt) nivå och utgå ifrån det för att ta reda på vad som behöver sparas i tabeller. Lhotka[98] visar i sin bok hur man kan börja med komponenterna i affärslogiklagret för att sen övergå till att skapa ett gränssnitt i presentationslagret för att testa komponenterna. Sist av allt skapar han tabellerna och komponenterna i datalagret.

Exemplet bygger på ett system för bokning av datorer i datorsalar. Ett sådant system skulle vid EKI på Mälardalens högskola ha en potential att serva ca 3 000 studenter. Sannolikheten att flera student skulle boka samtidigt är relativt stor.

---

### 8.1 Beskrivning av bokningssystemet

Studenten ska, via ett grafiskt gränssnitt, kunna boka en dator genom att fylla i datum, datorsal, dator i sal och sitt namn. Och eftersom det lättaste sättet att undvika fel i applikationer är att endast ge användaren möjlighet att välja från existerande alternativ så kommer vi att använda oss av kombinationsrutor för datum, datorsalar och datorer.

För att göra det enkel för oss att kontrollera att bokningarna sparats i databasen så kommer vi även att göra ett formulär där vi kan ange datorsal och visa bokningarna för datorsalen.

---

### 8.2 Databasen

I detta exempel kommer vi att använda en Access-databas (DatorBokning.mdb) med tre tabeller – Salar, Datorer respektive Bokningar. Tabellernas design är enligt nedanstående tabeller – primärnyckeln visas med (PK) i beskrivningen och kan bestå av fler än ett fält.

Valet av Access som databas har gjorts för att när man använder ADO så kommer koden vara (i stort sett) lika oavsett vilken databas man använder. Det som skiljer är Connection-objektets ConnectString och att vissa databaser kanske inte stödjer någon funktion i ADO (t.ex. `CursorLocation = adUseClient`).

#### 8.2.1 Tabellernas utformning

Som tabellen Bokningar visar så har bokningarna begränsats till en person per dator och dag. Detta har gjorts för att datamodellen inte ska bli för komplex. Det finns även en del fält i tabellerna som inte kommer användas i exemplet (HusId, CPU, MAC och IP). Dessa finns med för att göra databasmodellen mer "realistisk" och ge möjlighet till fortsatt egen utveckling av applikationens komponenter.

### 8.2.1.1 Salar (tabellnamn tblSalar)

| Fältnamn | Datotyp | Fältstorlek  | Beskrivning                     |
|----------|---------|--------------|---------------------------------|
| SId      | Tal     | Långt heltal | Id för datorsal (PK)            |
| HusId    | Tal     | Byte         | Id för hus som datorsal finns i |

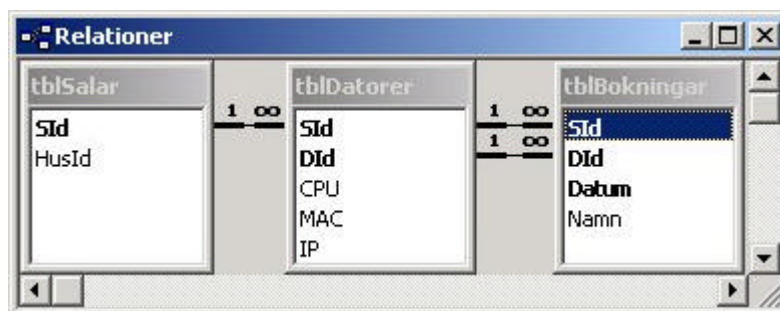
### 8.2.1.2 Datorer (tabellnamn tblDatorer)

| Fältnamn | Datotyp | Fältstorlek  | Beskrivning                                             |
|----------|---------|--------------|---------------------------------------------------------|
| SId      | Tal     | Långt heltal | Id för datorsal som dator finns i (PK)                  |
| DId      | Tal     | Långt heltal | Id för dator i datorsal SId (PK)                        |
| CPU      | Text    | 3            | Processor i dator (t.ex. 486, P1 och P3)                |
| MAC      | Text    | 12           | MAC-adress på nätverkskort i dator (t.ex. 00a020F32434) |
| IP       | Text    | 15           | IP-adress för dator (t.ex. 130.243.90.72)               |

### 8.2.1.3 Bokningar (tabellnamn tblBokningar)

| Fältnamn | Datotyp   | Fältstorlek  | Beskrivning                            |
|----------|-----------|--------------|----------------------------------------|
| SId      | Tal       | Långt heltal | Id för datorsal som dator finns i (PK) |
| DId      | Tal       | Långt heltal | Id för dator som bokning avser (PK)    |
| Datum    | Datum/tid | Kort datum   | Datum som bokning avser (PK)           |
| Namn     | Text      | 50           | Namn på person som bokar               |

Relationerna mellan tabellerna visas i figuren nedan.



## 8.3 Komponenter

Under första fasen av utvecklingen kommer vi att placera alla komponenter och det grafiska gränssnittet på samma dator. (När vi övergår till att anpassa komponenterna för MTS krävs att du har MTS installerat<sup>22</sup> på en dator med Windows NT 4 eller använder Windows 2000!) Vi kommer även att börja med att testa applikationen steg för steg innan vi anpassar komponenterna för MTS.

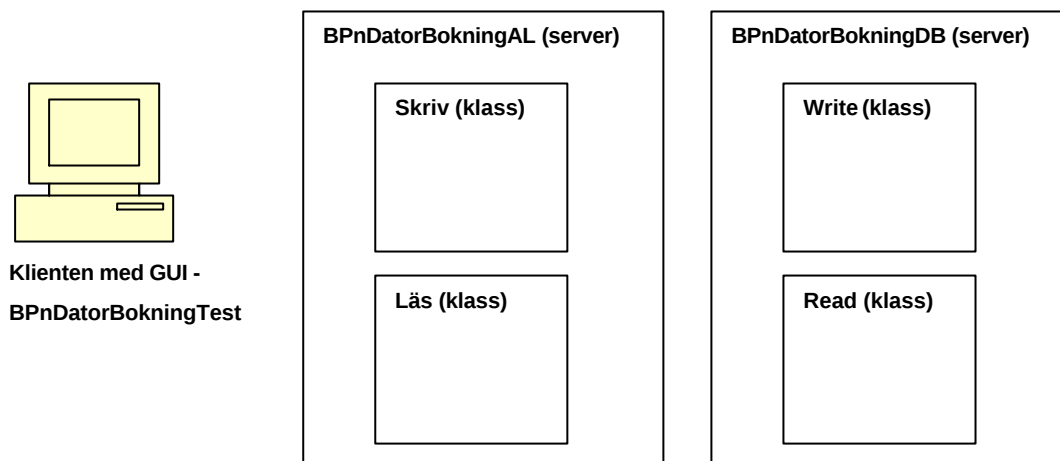
Exemplet kommer innehålla fyra komponenter där vi ger datakomponenterna engelska namn och prefix för att skilja dem från affärskomponenterna som vi ger svenska namn och prefix.

- två datakomponenter – Read och Write

<sup>22</sup> MTS följer, liksom IIS, med NT 4 Option Pack.

- två affärskomponenter – Läs och Skriv

Uppdelningen av komponenterna på detta sätt möjliggör för komponenterna som uppdaterar tabeller, Skriv och write, att stödja transaktioner. Datakomponenterna placerar vi i en COM-server (projekt/DLL) och affärskomponenterna i en annan COM-server (se bild nedan).



Det är dags att skapa komponenterna. Vi börjar med datakomponenterna och testar dem med enkla VB-gränssnitt. När dessa fungerar felfritt (? ☺) så skapar vi affärskomponenterna som använder datakomponenterna.

### 8.3.1 Datakomponenten Read

Skapa ett nytt Visual Basic-projekt och välj **ActiveX DLL** som typ samt ändra projektets namn till DatorBokningDB. (Eftersom flera kan tänkas skapa dessa och andra komponenter i datorsalarna kan det vara bra idé om ni använder er användaridentitet som prefix i COM-servrarnas namn. D.v.s. projektet bör heta något i stil med bpn97099DatorBokningDB. I detta exempel använder jag prefixet "BPn".) Suffixet DB anger att det är en datakomponent som COM-servern innehåller – för affärskomponenterna använder jag suffixet AL (affärslogik). Glöm inte att bocka för kryssrutan **Unattended Execution!**

Komponenter som läser från databaser är oftast de lättaste att utveckla (eftersom man inte skriver till databasen och behöver bry sig om låsningar i databasen). De kan dock bli ganska omfattande om man vill presentera data på många olika sätt, d.v.s. man får utveckla en metod för varje sätt. Men genom att börja med denna typ av komponent kan man kontrollera resultatet av uppdateringar som görs mot databasen. Därför börjar vi med denna datakomponent som endast läser från databasen.

#### 8.3.1.1 Deklarationer

Först och främst vill vi använda direktivet `Option Explicit` – det bör vi göra i alla Visual Basic-moduler för att slippa behöva felsöka felstavade variabelnamn. Vi deklarerar även en konstant (`cstrCONN`) som innehåller Connection-objektets `ConnectionString` (anpassa sökvägen till er databas) så att vi slipper ändra på flera ställen om vi ändrar databas senare.

```
Option Explicit
Private Const cstrConn = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=C:\Student\BPnDatorBokning\DatorBokning.mdb;" _
```

```
& "Persist Security Info=False"
```

En `ConnectionString` deklareras med fördel som en globalvariabel för projektet. På så sätt så blir den tillgänglig i hela projektet och man behöver endast ändra på ett ställe om man byter datakälla.

### 8.3.1.2 Metoden `GetSalar()`

Eftersom vi ska använda kombinationsrutor för att välja från så behöver vi metoder för att hämta värdena vi ska använda i kombinationsrutorna. Vi börjar på 1-sidan i databasmodellen och gör en metod för att hämta datersalarna – metoden heter `GetSalar()`. För att tala om att vi hämtar något så ger vi metoden prefixet "Get". Eftersom detta är datakomponent så kommer vi att returnerar datersalarna som poster i ett `Recordset`-objekt.

Vi behöver alltså en funktion för att kunna returnera posterna. I funktionen kommer vi använda oss av ett `Connection`-objekt för att öppna en förbindelse till databasen och sen en `SQL`-fråga för att hämta posterna. Frågan exekverar vi genom att använda `Connection`-objektets funktion `Execute()` med `SQL`-frågan som parameter. Resultatet från frågan skickar vi direkt som resultat för vår funktion. Och för att komma ihåg att vi ska uppdatera komponenten när vi ska anpassa den för MTS så skriver vi en "todo-kommentar" sist i metoden. På så sätt kan vi söka efter strängen "TODO:" för att hitta var vi ska lägga till `SetComplete()` eller `SetAbort()`. Observera att det fungerar att använda `New`-operatorn för att skapa `ADO`-objekten – de använder sig inte av MTS.

```
Public Function GetSalar() As ADODB.Recordset
    Dim adoConn As ADODB.Connection
    Dim adoRS As ADODB.Recordset
    Dim strSQL As String

    strSQL = "SELECT Sid FROM tblSalar;"
    Set adoConn = New ADODB.Connection
    adoConn.CursorLocation = adUseClient
    adoConn.Open cstrConn
    Set adoRS = adoConn.Execute(strSQL)
    Set adoRS.ActiveConnection = Nothing
    Set GetSalar = adoRS
    Set adoConn = Nothing
    'TODO: Tala om för MTS att vi är klara
End Function
```

'Deklarera variabler  
'Skapa fråga  
'Skapa Connection-objektet  
'Ange att klient hanterar postpekare  
'Öppna förbindelsen  
'Utför fråga  
'Koppla loss Recordset  
'Returnera  
'Stäng förbindelse

### 8.3.1.3 Metoden `GetDatorerForSal()`

Vi kan nu välja bland datersalarna och behöver en metod för att kunna hämta datorerna i datersalen till nästa kombinationsruta. För detta skapar vi en funktion där vi som parameter skickar datersalen som vi vill ha datorerna i. Denna metod skiljer sig inte så mycket från föregående, utom att `SQL`-frågan får en `WHERE`-sats.

```
Public Function GetDatorerForSal(ByVal Sal As Integer) As ADODB.Recordset
    Dim adoConn As ADODB.Connection
    Dim adoRS As ADODB.Recordset
    Dim strSQL As String

    strSQL = "SELECT Did FROM tblDatorer WHERE Sid = " & Sal & ";"
    Set adoConn = New ADODB.Connection
    adoConn.CursorLocation = adUseClient
    adoConn.Open cstrConn
    Set adoRS = adoConn.Execute(strSQL)
    Set adoRS.ActiveConnection = Nothing
```

'Ange att klient hanterar postpekare  
'Koppla loss Recordset

```
Set GetDatorerForSal = adoRS  
Set adoConn = Nothing  
'TODO: Tala om för MTS att vi är klara  
End Function
```

#### 8.3.1.4 Metoden GetBokningarForSal()

Sist av allt skapar vi en funktion för att hämta bokningarna för en datorsal som vi kan använda för vårt formulär där vi kontrollerar att bokningarna sparats i databasen. Inte heller denna funktion skiljer sig så mycket från de två tidigare funktionerna.

```
Public Function GetBokningarForSal(ByVal Sal As Integer) As ADODB.Recordset  
    Dim adoConn As ADODB.Connection  
    Dim adoRS As ADODB.Recordset  
    Dim strSQL As String  
  
    strSQL = "SELECT DID FROM tblBokningar WHERE SID = " & Sal & ";"  
    Set adoConn = New ADODB.Connection  
    adoConn.CursorLocation = adUseClient 'Ange att klient hanterar postpekare  
    adoConn.Open cstrConn  
    Set adoRS = adoConn.Execute(strSQL)  
    Set adoRS.ActiveConnection = Nothing 'Koppla loss Recordset  
    Set GetBokningarForSal = adoRS  
    Set adoConn = Nothing  
'TODO: Tala om för MTS att vi är klara  
End Function
```

#### 8.3.1.5 Testa metoderna

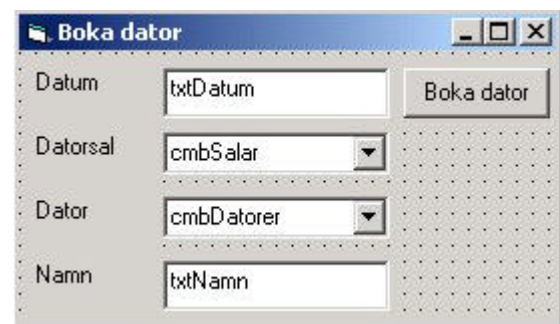
Innan vi anpassar komponenten för MTS bör vi som sagt testa funktionaliteten hos komponenten. För detta skapar vi lämpligen ett grafiskt gränssnitt i Visual Basic som jobbar direkt mot datakomponenten. Det grafiska gränssnittet kan vi sen anpassa (med smärre justeringar) så att det jobbar mot affärskomponenten istället för datakomponenten.

För att testa skapar vi ett nytt Visual Basic-projekt av typen Standard EXE. På formuläret (som vi ger namnet frmBoka) placerar vi två textrutor (txtDatum och txtNamn), två kombinationsrutor (cmbSalar och cmbDatorer), en knapp (btnBoka) och fyra etiketter (alltid bra att vi vid ett senare tillfälle förstår vad vi ska fylla i textrutorna ☺) enligt bilden till höger.

Ändra egenskapen **Style** för

kombinationsrutorna till 2 - Dropdown List

(så att användaren inte kan skriva egna alternativ utan endast välja från listan).



#### Deklarera variabler

Eftersom chansen är stor att komponenterna kommer exekvera på en annan dator så deklarerar vi komponenternas variabler som globala. På så sätt så kommer klienten hålla kvar referenserna till komponenterna under hela programmets exekvering. Detta leder till effektivare exekvering då det inte behöver etableras kontakt med de distribuerade komponenterna i varje metoanrop.

```
'Deklarera variabel av komponentens klass  
Public objReadDB As BPnDatorBokningDB.Read
```

Det introducerar dock en intressant fråga: När ska komponenter skapas? Det finns bl.a. två svar på frågan:

- När program startar (i t.ex. Form\_Load() för ett programs menyformulär).
- När komponent behövs första gången.

Enklast lösningen ligger i första svaret. Det innebär dock att vi kanske skapar instanser av komponenter som vi aldrig använder eller använder först när vi avslutar program. D.v.s. det kan var mindre effektivt.

Andra svaret innebär dock att vi varje gång måste testa om komponent finns innan vi använder komponenten. Om komponenten inte finns skapas den, annars används den redan existerande komponenten. (Detta gör å andra sidan kanske inte så mycket om det är ett VB-gränssnitt där vi inte använder datorns processor till max. ☺)

I detta exempel har jag valt att skapa komponenterna när första formuläret laddas. Det andra formuläret (se längre ner) kommer alltså använda komponenten som skapats i det första formuläret. Därför har variabeln med komponenten deklarerats som publik.

### Fylla cmbSalar med värden

Det första vi måste göra (i kod) är att fylla kombinationsrutan för datorsalar med värden (kombinationsrutan för datorer fyller vi på när studenten valt en datorsal). Därför placerar vi följande kod i metoden Form\_Load() (dubbelt klicka på formulärets bakgrund för att skapa skalkod för metoden).

```
Private Sub Form_Load()  
    'Vi behöver ett Recordset-objekt för att hantera resultatet  
    Dim adoRS As ADODB.Recordset  
    'Skapa objekt av komponent. Vi använder CreateObject för att  
    'slippa behöva ändra koden när vi anpassar för MTS.  
    Set objReadDB = CreateObject("BPNdatorBokningDB.Read")  
    'Hämta salarna från databasen (som Recordset-objekt)  
    Set adoRS = objReadDB.GetSalar  
    'Loopa över posterna och lägg till i kombinationsrutan  
    While Not adoRS.EOF  
        cmbSalar.AddItem adoRS.Fields("SID")  
        adoRS.MoveNext 'Glöm _inte_ att flytta till nästa post!  
    Wend  
  
    Set adoRS = Nothing 'Rensa upp genom att förstöra objekten  
End Sub
```

### Fyll cmbDatorer med värden

Nästa steg är att fylla på kombinationsrutan för datorer genom att använda händelsemetoden Click() för cmbSalar. (Händelsen Change fungerar tyvärr inte även om man kan tycka att det vore det logiska alternativet.) Koden ser nästan likadan ut som förra metoden (vill ni så kan ni kopiera koden från ovanstående metod och ändra det som är gråmarkerat)

```
Private Sub cmbSalar_Click()  
    'Vi behöver ett Recordset-objekt för att hantera resultatet  
    Dim adoRS As ADODB.Recordset  
    'Hämta datorer för salen från databasen  
    Set adoRS = objReadDB.GetDatorerForSal(CInt(cmbSalar.Text))  
    'Loopa över posterna och lägg till i kombinationsrutan  
  
    While Not adoRS.EOF  
        cmbDatorer.AddItem adoRS.Fields("DId")  
        adoRS.MoveNext 'Glöm _inte_ att flytta till nästa post!
```

```
Wend

Set adoRS = Nothing      'Rensa upp genom att förstöra objekten
End Sub
```

I koden ovan använder vi funktionen CInt() för att konvertera texten i kombinationsrutan till ett tal.

### Koden bakom knappen

Innan vi kan skriva koden bakom knappen så måste vi skapa datakomponenten write (så att vi även kan skriva till databasen). Men metदानropet för att skriva en bokning till databasen bör kanske ha ett utseende enligt följande:

```
objWriteDB.Book CInt(cmbSalar.Text), CInt(cmbDatorer.Text), txtDatum, txtNamn
```

### Testa sista metoden

Den sista metoden i komponenten, GetBokningarForSal(), testar vi lämpligast genom att i samma projekt skapa ett nytt formulär (frmVisaBokningar) med en listruta (lstBokningar) på. För att visa bokningar för en sal så väljer användaren först en datorsal och klickar sen på en knapp för att visa bokningar för den salen (se nedan för koden bakom knappen).



Vi placerar koden för att fylla listrutan med bokningar i Form\_Load(). Observera att vi använder variabeln med komponenten från det första formuläret (frmBoka) för att inte skapa två instanser av samma komponent.

```
Private Sub Form_Load()
    Dim adoRS As ADODB.Recordset
    'Hämta bokningar av datorer från databasen
    Set adoRS = _
        frmBoka.objReadDB.GetBokningarForSal(CInt(frmBokaDator.cmbSalar.Text))

    'Loopa över posterna och lägg till i listrutan
    While Not adoRS.EOF
        lstBokningar.AddItem "Dator: " & adoRS.Fields("DId") & " Datum: " & _
            adoRS.Fields("Datum") & " Student: " & adoRS.Fields("Namn")
        adoRS.MoveNext 'Glöm inte att flytta till nästa post!
    Wend

    Set adoRS = Nothing      'Rensa upp genom att förstöra objekten
End Sub
```

Nu återstår bara att ge användaren möjlighet att visa formuläret. Lägg till en knapp (btnVisaBokningar) på det första formuläret (frmBoka) och skriv följande kod i dess Click-metod.

```
Private Sub btnVisaBokningar_Click()
    If cmbSalar.Text = "" Then 'Om ingen sal valts i cmbSalar...
        MsgBox "Du måste välja en datorsal först!", vbInformation
    Else
        '... annars visa formulär
    End If
End Sub
```

```
frmVisaBokningar.Show  
End If  
End Sub
```

### 8.3.1.6 Anpassa komponenten för MTS/COM+

Om komponenten fungerade i testerna så kan vi anpassa komponenten för MTS (COM+). Vi börjar med att lägga till en referens till MTS (COM+) *type library*. Markera komponentens klass och välj **References...** från Project-menyn samt leta upp och bocka för **Microsoft Transaction Server Type Library (COM+ Services Type Library)**.<sup>23</sup> Sen ändrar vi klassens egenskap **MTSTransactionMode** till 1 - NoTransactions. Nu kan vi lägga till koden som behövs för att implementera gränssnitt **IObjectContext** och för att hämta en referens till objektets kontextobjekt. Under raden **Option Explicit** lägger vi till nedanstående kod.

```
Option Explicit  
Implements ObjectControl          'Gränssnitt som objekt i MTS måste implementera  
Private mobjContext As ObjectContext 'Var. för objektets kontextobjekt
```

Sen väljer vi **ObjectControl** från listrutan **Object** i kodfönstret samt de tre metoderna i listrutan **Procedure** i tur och ordning för att skapa skalkoden för metoderna. Implementationen av metoderna visa nedan.

```
Private Sub ObjectControl_Activate()  
    Set mobjContext = GetObjectContext  
End Sub  
  
Private Function ObjectControl_CanBePooled() As Boolean  
    ObjectControl_CanBePooled = False  
End Function  
  
Private Sub ObjectControl_Deactivate()  
    Set mobjContext = Nothing  
End Sub
```

Och sist av allt så letar vi upp alla ”todo-kommentarerna” och ersätter dem med `mobjContext.SetComplete`.

### 8.3.1.7 Klassen i sin helhet

För att visa på helheten så följer här den fulla koden för klassen **Read**.

```
*****  
*** BESKRIVNING av komponenten BPnDatorBokningarDB.Write  
*** Komponenten exekverar i MTS och använder transaktioner.  
***** BPn - EkI, Mdh 2000 *****  
Option Explicit  
Implements ObjectControl          'Gränssnitt som objekt i MTS måste implementera  
Private mobjContext As ObjectContext 'Var. för objektets kontextobjekt  
Private Const cstrConn As String = "... " 'ConnectionString för Connection-objektet  
  
*****  
*** Funktionen returnera alla bokningar för salen Sal
```

<sup>23</sup> **Observera** att dessa *type libraries* endast är tillgängliga på Windows NT 4 med MTS (NT4 Option Pack) installerat respektive Windows 2000/XP!

```
*****
Public Function GetBokningarForSal(ByVal Sal As Integer) As ADODB.Recordset
    Dim adoConn As ADODB.Connection
    Dim adoRS As ADODB.Recordset
    Dim strSQL As String

    strSQL = "SELECT * FROM tblBokningar WHERE SId = " & Sal & ";"
    Set adoConn = New ADODB.Connection
    adoConn.CursorLocation = adUseClient      'Ange att klient hanterar postpekare
    adoConn.Open cstrConn
    Set adoRS = adoConn.Execute(strSQL)
    Set GetBokningarForSal = adoRS
    Set adoRS.ActiveConnection = Nothing      'Koppla loss Recordset
    Set adoConn = Nothing
    mobjContext.SetComplete
End Function

*****
'*** Funktionen returnera alla datorer i salen Sal
'*** (Denna metod skulle kunna kompletteras med en metod som
'*** GetObokadeDatorerISal() som returnerar alla obokade datorer i salen.)
'***
Public Function GetDatorerForSal(ByVal Sal As Integer) As ADODB.Recordset
    Dim adoConn As ADODB.Connection
    Dim adoRS As ADODB.Recordset
    Dim strSQL As String

    strSQL = "SELECT DID FROM tblDatorer WHERE SId = " & Sal & ";"
    Set adoConn = New ADODB.Connection
    adoConn.CursorLocation = adUseClient      'Ange att klient hanterar postpekare
    adoConn.Open cstrConn
    Set adoRS = adoConn.Execute(strSQL)
    Set adoRS.ActiveConnection = Nothing      'Koppla loss Recordset
    Set GetDatorerForSal = adoRS
    Set adoConn = Nothing
    mobjContext.SetComplete
End Function

*****
'*** Funktionen returnera alla salar i tabellen tblSalar
'***
Public Function GetSalar() As ADODB.Recordset
    Dim adoConn As ADODB.Connection
    Dim adoRS As ADODB.Recordset
    Dim strSQL As String

    strSQL = "SELECT SId FROM tblSalar;"
    Set adoConn = New ADODB.Connection
    adoConn.CursorLocation = adUseClient      'Ange att klient hanterar postpekare
    adoConn.Open cstrConn
    Set adoRS = adoConn.Execute(strSQL)
    Set adoRS.ActiveConnection = Nothing      'Koppla loss Recordset
    Set GetSalar = adoRS
    Set adoConn = Nothing
    mobjContext.SetComplete
End Function

*****
'*** START Implementation av IOBJECTCONTROL
'***
'Metoden anropas då objekt aktiveras.
'I denna komponents skapas även ConnectString för Connection-objektet
Private Sub ObjectControl_Activate()
    Set mobjContext = GetObjectContext
    cstrConn = "Provider=Microsoft.Jet.OLEDB.4.0;" _
        & "Data Source=C:\Student\BPN\DatorBokning\DatorBokning.mdb;" _
        & "Persist Security Info=False"
End Sub

'Metoden anropas då objektet inaktiveras. Under MTS svarar alltid metoden
' falskt då MTS inte stödjer objektpoolning. Så även under COM+ då VB inte
' stödjer poolning.
Private Function ObjectControl_CanBePooled() As Boolean
    ObjectControl_CanBePooled = False
End Function
```

```
'Metoden anropas då objektet inaktiveras.  
Private Sub ObjectControl_Deactivate()  
    Set mobjContext = Nothing  
End Sub  
'*****  
'*** SLUT Implementation av IObjectControl  
'*****
```

### 8.3.2 Datakomponenten Write

För att lägga till klassen Write till projektet BPnDatorBokningDB så högerklickar vi på projektet i **Project Explorer** (längst upp till höger) samt väljer **Add...** från menyn som visas och sen **Class Module**. Ändra namnet på klassen till Write.

Komponenten Write innehåller bara en metod, Book(), som skriver en post till tabellen Bokningar. Eftersom uppdateringar av databaser ofta kan leda till fel så ska vi göra en felhanterare för metoden.

Metoden implementerar vi som en funktion som returnerar en sträng som visar om bokningen gick bra eller om vi försöker boka en dator som redan är bokad (d.v.s. posten finns redan i tabellen).

```
'*****  
'*** BESKRIVNING av komponenten BPnDatorBokningarDB.Write  
'*** Komponenten exekverar i MTS och använder transaktioner.  
'*** Se komponenten BPnDatorBokningarDB.Read för förklaringar av metoder  
'*** som inte förklarats här.  
'*** Komponenter skulle kunna kompletteras med metoder för att rensa bort  
'*** t.ex. bokningar som inte behövs och för underhåll av övriga tabeller.  
'***** BPn - EkI, Mdh 2000 *****  
Option Explicit  
Implements ObjectControl  
  
Private mobjContext As ObjectControl  
Private Const cstrConn As String = "..."  
  
'*****  
'*** Funktionen lägger till en post i tabellen tblBokningar.  
'*** Det enda felet som funktionen hanterar är om posten redan finns i  
'*** tabellen. Övriga fel skickas vidare till anropande metod.  
'*** Funktionen bör returnera ett tal istället för en sträng så att  
'*** felhantering blir lättare att implementera (inte att förstå!).  
'*****  
Public Function Book(ByVal Sal As Long, ByVal Dator As Long, _  
    ByVal Dag As Date, ByVal Namn As String) As String  
    On Error GoTo Book_Error  
    Dim adoConn As ADODB.Connection  
    Dim strSQL As String  
    'Skapa SQL-fråga för att lägga till en post  
    strSQL = "INSERT INTO tblBokningar VALUES (" & Sal & "," & Dator & _  
        & "," & Dag & "," & Namn & "');"   
  
    Set adoConn = New ADODB.Connection  
    adoConn.Open cstrConn  
    adoConn.Execute strSQL  
    adoConn.Close  
    Set adoConn = Nothing  
  
    Book = "Datorn bokades."  
    mobjContext.SetComplete  
    Exit Function  
  
Book_Error:  
    If Err.Number = -2147467259 Then  
        Book = "FEL: Du försöker boka en dator som redan är bokad!"  
    Else  
        Err.Raise Err.Number, Err.Source, Err.Description  
    End If
```

```
If adoConn.State = adStateOpen Then
    adoConn.Close
End If
Set adoConn = Nothing
mobjContext.SetAbort
End Function

'*****
'*** START Implementation av IObjectControl
'*****

Private Sub ObjectControl_Activate()
    Set mobjContext = GetObjectContext
    cstrConn = "Provider=Microsoft.Jet.OLEDB.4.0;" _
        & "Data Source=C:\Student\BPnDatorBokning\DatorBokning.mdb;" _
        & "Persist Security Info=False"
End Sub

Private Function ObjectControl_CanBePooled() As Boolean
    ObjectControl_CanBePooled = False
End Function

Private Sub ObjectControl_Deactivate()
    Set mobjContext = Nothing
End Sub

'*****
'*** SLUT Implementation av IObjectControl
'*****
```

### 8.3.2.1 Testa komponenten

För att testa komponenten så skriver vi kod bakom knappen (btnBoka) på bokningsformuläret (frmBokaDator). Innan vi gör det så måste vi deklarera variabeln med komponent längst upp i formulär.

```
Public objWrite As BPnDatorBokningDB.Write
```

Vi lägger även till skapandet av objektet i formulärets metod Form\_Load().

```
'Måste använda funktionen CreateObject istället för New-operatorn
' för att använda COM (och MTS) att skapa objektet och då objektet
' som skapas kan skapas på annan dator.
Set objWrite = CreateObject("BPnDatorBokningDB.Write")
```

Och sist lägger vi till koden "bakom" knappen.

```
Private Sub btnBoka_Click()
    Dim strResultat As String
    strResultat = objWrite.Book(cmbSalar.Text, cmbDatorer.Text, _
        txtDatum.Text, txtNamn.Text)
    MsgBox strResultat 'Visa om bokning gick bra eller om det blev fel
End Sub
```

### 8.3.3 Affärskomponenten Las

För affärskomponenten Las (och Skriv) skapar vi ett nytt projekt.<sup>24</sup> Ändra projektets namn till BPnDatorBokningAL och bocka för **Unattended Execution**. Ändra klassens namn till Las i egenskapsfönstret.

Komponenten har samma antal metoder som datakomponenten Read och metoderna har liknande namn, fast med prefixet "Hamta" istället för "Get". Metodernas uppgift är att skapa en instans av datakomponenten och anropa metod för att hämta posterna från databasen, för att sen returnera posterna till klienten.

När vi utvecklar komponenter som kommer att skapa instanser av andra komponenter i MTS så måste vi använda `CreateInstance()`. Men innan vi anpassat komponenterna för MTS så har vi inte tillgång till något kontextobjekt utan måste använda `CreateObject()`.

**Glöm inte** att testa komponenten **innan** den installeras i MTS! I koden nedan kommenterar vi då bort `mobjContext.SetComplete()` samt ersätter anrop av `mobjContext.CreateInstance()` med `CreateObject()`. När vi sen ska installera komponenten i MTS så tar vi bort kommentarstecknen och återställer anropen till `CreateInstance()`.

```
*****
'*** BESKRIVNING av komponenten BPnDatorBokningarAL.Las
'*** Komponenten exekverar i MTS men använder inte transaktioner eftersom
'*** den bara läser från databasen.
'*** Se komponenten BPnDatorBokningarDB.Read för förklaringar av metoder
'*** som inte förklarats här.
***** BPn - EkI, Mdh 2000 *****
Option Explicit
Implements ObjectControl

Private mobjContext As ObjectContext

*****
'*** Funktionen returnerar bokningar för salen Sal
*****
Public Function HamtaBokningarForSal(ByVal Sal As Long) As ADODB.Recordset
    Dim objDBR As BPnDatorBokningDB.Read
    Set objDBR = mobjContext.CreateInstance("BPnDatorBokningDB.Read")
    Set HamtaBokningarForSal = objDBR.GetBokningarForsal(Sal)
    Set objDBR = Nothing
    mobjContext.SetComplete
End Function

*****
'*** Funktionen returnerar datorer i salen Sal
*****
Public Function HamtaDatorerForSal(ByVal Sal As Long) As ADODB.Recordset
    Dim objDBR As BPnDatorBokningDB.Read
    Set objDBR = mobjContext.CreateInstance("BPnDatorBokningDB.Read")
    Set HamtaDatorerForSal = objDBR.GetDatorerForsal(Sal)
    Set objDBR = Nothing
    mobjContext.SetComplete
End Function

*****
'*** Funktionen returnerar alla salar
*****
Public Function HamtaSalar() As ADODB.Recordset
    Dim objDBR As BPnDatorBokningDB.Read
    Set objDBR = mobjContext.CreateInstance("BPnDatorBokningDB.Read")
    Set HamtaSalar = objDBR.GetSalar
    Set objDBR = Nothing
```

<sup>24</sup> Glöm inte att du kan lägga till ett nytt projekt (för att skapa en projektgrupp) och därmed lättare felsöka under utveckling!

```
        mobjContext.SetComplete
    End Function

    '*****
    '*** START Implementation av IObjectControl
    '*****
    Private Sub ObjectControl_Activate()
        Set mobjContext = GetObjectContext
    End Sub

    Private Function ObjectControl_CanBePooled() As Boolean
        ObjectControl_CanBePooled = False
    End Function

    Private Sub ObjectControl_Deactivate()
        Set mobjContext = Nothing
    End Sub
    '*****
    '*** SLUT Implementation av IObjectControl
    '*****
```

### 8.3.3.1 Testa komponenten

För att testa komponenten Las behöver vi ändra en del mindre saker i våra formulär. Bl.a. måste vi ändra klassen för variablerna från BPnDatorBokningDB.Read till BPnDatorBokningAL.Las. Förändringar i Form\_Load() har gråmarkerats i koden nedan och liknande förändringar behöver göras i övriga metoder. (Vill ni kan ni även ändra namnet på variabeln objReadDB till objLasAL för att bättre avspegla klassen för objektet i variabeln.)

```
Private Sub Form_Load()
    Dim objReadDB As BPnDatorBokningAL.Las
    Dim adoRS As ADODB.Recordset
    Set objReadDB = CreateObject("BPnDatorBokningAL.Las")
    Set adoRS = objReadDB.HamtaSalar
    While Not adoRS.EOF
        cmbSalar.AddItem adoRS.Fields("SID")
        adoRS.MoveNext 'Glöm _inte_ att flytta till nästa post!
    Wend

    Set adoRS = Nothing 'Rensa upp genom att förstöra objekten
    Set objReadDB = Nothing
End Sub
```

### 8.3.4 Affärskomponenten Skriv

Vi lägger till en ny klass i projektet BPnDatorBokningAL med namnet Skriv för att implementera komponenten.

Precis som datakomponenten write så innehåller affärskomponenten Skriv endast en metod – Boka() – som motsvaras av Book(). Och precis som för affärskomponenten Las så har metoden till uppgift att skapa en instans av datakomponenten för att sen anropa metoden. Metoden Boka() implementeras som en funktion för att kunna returnera resultatet från metoden Book().

Även här får vi inte glömma att testa komponenten **innan** den installeras i MTS. Återigen kommenterar vi bort rader med mobjContext.SetComplete() och ersätter anrop av mobjContext.CreateInstance() med CreateObject() (för att återställa när vi installerar i MTS).

```
'*****
```

```
''' BESKRIVNING av komponenten BPnDatorBokningarAL.Skriv
''' Komponenten exekverar i MTS och använder transaktioner.
''' Se komponenten BPnDatorBokningarDB.Read för förklaringar av metoder
''' som inte förklarats här.
''' Komponenter skulle kunna kompletteras med metoder för att rensa bort
''' bokningar som inte behövs och underhåll av övriga tabeller.
'''***** BPn - EkI, Mdh 2000 *****
Option Explicit
Implements ObjectControl

Private mobjContext As ObjectContext

'''*****
''' Funktionen anropar metoden Boka i komponenten BPnDatorBokningDB för
''' att lägga till en bokning. Funktionen returnerar en sträng som
''' talar om i fall bokningen gick bra eller inte.
'''*****
Public Function Boka(ByVal Sal As Long, ByVal Dator As Long, _
    ByVal Datum As Date, ByVal Namn As String) As String
    Dim objDBW As BPnDatorBokningDB.Write
    Dim strResultat As String

    Set objDBW = mobjContext.CreateInstance("BPnDatorBokningDB.Write")
    strResultat = objDBW.Book(Sal, Dator, Datum, Namn)
    Boka = strResultat
    mobjContext.SetComplete
End Function

'''*****
''' START Implementation av IObjectControl
'''*****
Private Sub ObjectControl_Activate()
    Set mobjContext = GetObjectContext
End Sub

Private Function ObjectControl_CanBePooled() As Boolean
    ObjectControl_CanBePooled = False
End Function

Private Sub ObjectControl_Deactivate()
    Set mobjContext = Nothing
End Sub
'''*****
''' SLUT Implementation av IObjectControl
'''*****
```

### 8.3.4.1 Testa komponenten

För att testa komponenten Skriv behöver vi även här ändra en del mindre saker i våra formulär. Bl.a. måste vi ändra klassen för variablerna från BPnDatorBokningDB.Write till BPnDatorBokningAL.Skriv. Ändringar i metoden btnBoka\_Click() visas nedan gråmarkerade. (Även här kan ni ändra variabelnamnet objWriteDB till objSkrivAL om ni vill.)

```
Private Sub btnBoka_Click()
    Dim objWrite As BPnDatorBokningAL.Skriv
    Dim strResultat As String
    Set objWrite = CreateObject("BPnDatorBokningAL.Skriv")
    strResultat = objWrite.Boka(cmbSalar.Text, cmbDatorer.Text, _
        txtDatum.Text, txtNamn.Text)
    MsgBox strResultat 'Visa om bokning gick bra eller om det blev fel
End Sub
```

(För att kunna motivera existensen av affärskomponenterna Las och Skriv så bör man i metoderna utföra någon form av kontroll att organisationens affärsregler efterföljs. T.ex. skulle man kunna kontrollera att studenter endast bokar en dator per dag. Hade det varit ett

säljsystem vi utvecklade i detta exempel så skulle t.ex. kontrollen att man utfört kreditupplysning på kunden då den handlar över 1000 kronor ske här.)

## 8.4 Formulär

När vi gör den slutgiltiga versionen av klientgränssnittet så måste (bör) vi deklarerar alla variabler som refererar till en komponent i MTS/COM+ som globala.<sup>25</sup> Detta för att undvika att vi måste etablera kontakt med server varje gång vi vill använda komponenterna. Vi kan t.ex. skapa instanser (objekt) av komponenterna i Form\_Load() så att de alltid finns tillgängliga.

I formulären BokaDator och VisaBokningar har kombinationsrutornas egenskap **Style** satts till 2 - Dropdown List så att användaren endast kan välja från listan.

### 8.4.1 BokaDator

Detta formulär har ändrat lite sen vi testade datakomponenterna. Bl.a. så har vi bytt ut textrutan txtDatum mot en kombinationsruta (cmbDatum) och fyller den med 14 datum med start på dagens datum (se Form\_Load() i koden nedan). På så sätt har vi eliminerat risken att användaren fyller i ett ogiltigt datum. Det görs också en kontroll att både datum och datorsal har angivits innan dator väljs. (Ett steg mot att kunna göra en metod i komponenterna som bara visar datorer som inte bokats istället för att som nu visa alla datorer i datorsalen.)

Kombinationsrutan för datorer (cmbDatorer) har inaktiverats vid start (med egenskapen **Enabled**) och aktiveras först när både datum och datorsal har valts i kombinationsrutorna Datum och Salar (cmbSalar\_Click()).

```
*****
*** Formulär som låter användaren boka en dator. Alla fält utom namnet på
*** den som bokar har placerats i komborutor för att göra det enkelt att
*** boka.
*****
Option Explicit

Private objDBR As BPnDatorBokningAL.Las
Private objWrite As BPnDatorBokningAL.Skriv
```

<sup>25</sup> I "vanliga fall" så ska variabler deklarerarar så lokalt som möjligt, d.v.s. globala variabler ska undvikas i största mån. Om ni t.ex. har ett Recordset-objekt (som används för resultat från metoder i komponenter) så bör variabeln deklarerarar i varje metod (och inte en för varje formulär/modul) även om variabel används i flera metoder. Jag vet att en del tycker att det är "jobbigt" att deklarerarar variabeln i varje metod, men det är god sed och minskar risken för fel (buggar).

```
Private Sub btnStang_Click()
    Unload Me
End Sub

Private Sub btnBoka_Click()
    Dim strResultat As String
    strResultat = objWrite.Boka(cmbSalar.Text, cmbDatorer.Text, _
                                cmbDatum.Text, txtNamn.Text)
    MsgBox strResultat 'Visa om bokning gick bra eller om det blev fel
End Sub

Private Sub cmbSalar_Click()
    Dim adoRS As ADODB.Recordset
    If cmbSalar.Text = "" Or cmbDatum = "" Then 'Om varken datum eller sal valts
        MsgBox "Du måste välja datum _och_ datorsal innan du kan väljar dator!", _
            vbInformation
    Else
        cmbDatorer.Clear 'Töm listrutan så att endast aktuell sals datorer visas
        'Skapa COM-objektet och hämta datorsalarna (se btnBoka_Click())
        Set objDBR = CreateObject("BPnDatorBokningAL.Las")
        Set adoRS = objDBR.HamtaDatorerForSal(cmbSalar.Text)
        'Fyll på komborutan med datorsalarna
        While Not adoRS.EOF
            cmbDatorer.AddItem adoRS.Fields(0)
            adoRS.MoveNext
        Wend
        Set objDBR = Nothing
        cmbDatorer.Enabled = True 'Aktivera möjligheten att välja en dator
    End If
End Sub

Private Sub Form_Load()
    Dim adoRS As ADODB.Recordset
    Dim Idag As Date
    Dim i As Integer

    'Skapa COM-objektet
    'Måste använda funktionen CreateObject istället för New-operatörn
    ' för att använda COM (och MTS) att skapa objektet och då objektet
    ' som skapas kan skapas på annan dator.
    Set objWrite = CreateObject("BPnDatorBokningAL.Skriv")
    Set objDBR = CreateObject("BPnDatorBokningAL.Las")
    '... och hämta datorsalarna
    Set adoRS = objDBR.HamtaSalar
    'Fyll på komborutan med datorsalarna
    While Not adoRS.EOF
        cmbSalar.AddItem adoRS.Fields(0)
        adoRS.MoveNext
    Wend
    Set objDBR = Nothing
    'Skapa datum att placera i komborutan cmbDatum
    Idag = Date
    For i = 0 To 13
        cmbDatum.AddItem DateAdd("d", i, Idag)
    Next i
End Sub
```

## 8.4.2 VisaBokningar

Detta formulär har gjorts om lite i jämförelse med det formulär vi skapade när vi skulle testa datakomponenten Read. Bl.a. har beroendet av det andra formuläret tagits bort genom att placera en kombinationsruta direkt på formuläret som man kan välja vilken sal man vill se bokningar för.



```
*****
'*** Formulär som låter användaren välja en datorsal som användaren vill
'*** se bokningar för. När användaren väljer datorsal i kombinationsrutan
'*** cmbSalar visas bokningarna i listrutan lstBokningar
*****
Option Explicit      'Be VB kontrollera att alla variabler är deklarerade

Private objDBR As BPnDatorBokningAL.Las

Private Sub btnStang_Click()
    Unload Me      'Ladda ur formuläret => formuläret visas inte
End Sub

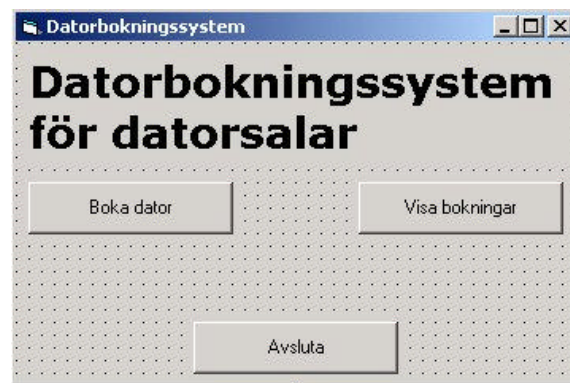
Private Sub cmbSalar_Click()
    Dim adoRS As ADODB.Recordset
    'Sök bara bokningar om en sal valts i cmbSalar...
    If cmbSalar.Text = "" Then
        MsgBox "Du måste välja en datorsal att visa bokningar för", _
            vbInformation
    Else
        Set objDBR = CreateObject("BPnDatorBokningAL.Las")
        Set adoRS = objDBR.HamtaBokningarForSal(cmbSalar.Text)

        lstBokningar.Clear 'Töm listrutan innan nya bokningar visas
        While Not adoRS.EOF 'Så länge det finns poster kvar fyll listruta
            lstBokningar.AddItem "Dator: " & adoRS.Fields(1) & " Datum: " _
                & adoRS.Fields(2) & " Student: " & adoRS.Fields(3) _
            adoRS.MoveNext
        Wend
    End If
End Sub

Private Sub Form_Load()
    Dim objDBR As BPnDatorBokningAL.Las
    Dim adoRS As ADODB.Recordset
    'Skapa COM-objektet och hämta datorsalarna
    Set objDBR = CreateObject("BPnDatorBokningAL.Las")
    Set adoRS = objDBR.HamtaSalar
    'Fyll på komborutan med datorsalarna
    While Not adoRS.EOF
        cmbSalar.AddItem adoRS.Fields(0)
        adoRS.MoveNext
    Wend
    Set objDBR = Nothing
End Sub
```

### 8.4.3 Main

För att visa en meny vid start av applikation kan man skapa följande formulär (frmMain). Formuläret har ingen annan funktion än att visa tre knappar som öppnar bokningsformuläret, öppnar visningsformuläret respektive stänger applikationen.



```
*****
'*** Detta formulär är ett menyformulär med knappar som laddar andra
'*** formulär. När detta formulär laddas ur så stängs applikationen.
'***
Option Explicit

Private Sub btnAvsluta_Click()
    'Prova gärna att ta bort kommentar på första raden och för
    '   programsatsen i Form_Unload()
    'End      'Avslutar programmet på ett mindre snyggt sätt
    Unload Me 'Avslutar programmet på ett snyggt sätt genom att ladda ur
              ' formuläret. Om formuläret är det enda öppna så kommer
              ' applikationen att stängas.
End Sub

Private Sub btnBokaDator_Click()
    frmBokaDator.Show
End Sub

Private Sub btnVisaBokningar_Click()
    frmVisaBokningar.Show
End Sub

Private Sub Form_Unload(Cancel As Integer)
    'Denna procedur körs _bara_ om formuläret laddas ur med Unload Me
    'MsgBox "Nu avslutas jag..:"
End Sub
```