L U K U 5

Sovelluksen toimintojen toteuttaminen

Tässä luvussa

Luvussa 4 opit, kuinka voit Microsoft Visual C++-kehitysympäristön toimintoja hyödyntämällä luoda käyttöliittymän sovellukseesi ilman, että sinun tarvitsee juurikaan kirjoittaa ohjelmakoodia. Yksi visuaalisen ohjelmointiympäristön eduista on se, että voit nopeasti ja helposti toteuttaa suunnittelemasi käyttöliittymän.

Tässä luvussa opit joitain MFC-ohjelmointitekniikoita, joiden avulla voit toteuttaa sovelluksesi toimintoja. Opit, kuinka käytät luvussa 4 tekemiäsi dialogeja ja kuinka hyödynnät Win32-ympäristön monisäikeisyyttä. Opit myös lisää tekniikoista, joiden avulla voit esittää sovelluksen tietoja näytöllä tai tulostimella ja kuinka teet ohjeen MFC-sovellukseen.

Ennen kuin aloitat

Ennen tämän luvun aloittamista sinun tulisi lukea luvut 2-4 ja suorittaa niihin kuuluvat harjoitukset.

Oppitunti 1: Dialogien käyttäminen

Luvun 4 oppitunnilla 2 opit, kuinka teet dialogimalliin pohjautuvia dialogiluokkia, joita voit käyttää dialogien näyttämiseen sovelluksessasi. Tällä oppitunnilla opit, kuinka käytät näitä dialogiluokkia sovelluksesi ohjelmakoodissa. Siirrät tietoja dialogin kontrolleista sovellukseesi, joka muodostaa kontrollista lähtöisin olevan sanoman. Opit myös, kuinka teet dialogimalleista ominaisuusikkunan sivuja.

Tämän oppitunnin jälkeen:

- Tiedät, kuinka DDX ja DDV toiminnot toteutetaan.
- Osaat lisätä ClassWizardilla jäsenmuuttujia dialogin tietoja varten.
- Osaat lisätä ClassWizardia käyttäen funktioita, jotka käsittelevät dialogikontrollilta tulevia sanomia.
- Tiedät, kuinka teet ominaisuusikkunan.

Oppitunnin arvioitu kesto: 60 minuuttia

DDX ja DDV

DDX (Dialog data exchange) on helppo tapa alustaa dialogien kontrolleja ja kerätä tietoja käyttäjältä. DDV (Dialog data validation) tarjoaa helpon keinon käyttäjän dialogiin syöttämien tietojen tarkistamiseen.

DDX ja DDV toteutetaan dialogiluokassa ja niiden koodiarkkitehtuuri on vastaava kuin luvussa 3 esitellyssä sanomienohjausarkkitehtuurissa. Dialogiluokan jäsenmuuttujat vastaavat dialogimallin kontrolleja. Globaaleja MFC-funktioita käytetään tietojen siirtämiseen jäsenmuuttujien välillä ja käyttäjän antamien tietojen tarkistamisessa.

Kun ClassWizard luo dialogiluokan, se lisää luokkaan koodin, jossa toteutetaan perus DDX/DDV-arkkitehtuuri. Voit lisätä dialogiluokkaan kontrolleja vastaavat jäsenmuuttujat ClassWizardia käyttäen ja määritellä samalla yksinkertaisia tarkistusohjeita käyttäjän kontrolleihin kirjoittamien tietojen oikeellisuuden varmistamista varten.

Kun käytät ClassWizardia DDX-jäsenmuuttujien lisäämiseen, ClassWizard tekee puolestasi seuraavat tehtävät:

- Jäsenmuuttujat lisätään luokan määrittelyyn.
- Jäsenmuuttujat alustetaan luokan muodostimessa.
- DDX-funktiot lisätään huolehtimaan tiedonsiirrosta DDX-jäsenmuuttujien ja dialogin kontrollien välillä.

- Jos käytät ClassWizardia jäsenfunktioiden tarkistus kriteerien asettamiseen, lisätään luokkaan DDV-funktiot, joiden avulla käyttäjän syöttämät tiedot tarkistetaan ennen niiden tallettamista DDX-jäsenmuuttujiin.

DDX- ja DDV-funktiot ovat globaaleja MFC-funktioita, jotka siirtävät tietoja dialogin kontrollien ja luokan jäsenmuuttujien välillä (DDX-funktiot) ja tarkistavat syötetyt tiedot (DDV-funktiot). MFC sisältää joukon yleisimpiä kontrollityyppejä vastaavia DDX-funktioita. Yleisimmin käytetyt funktiot ovat **DDX_Text()**, joka siirtää tietoja muokkausruudun ja dialogiluokan CString-tyyppisen jäsenmuuttujan välillä ja **DDX_Check()**, joka välittää valintaruutukontrollin tilan dialogiluokan BOOL-jäsenmuuttujaan ja muuttujasta kontrolliin. MFC sisältää myös joukon DDV-funktioita, joiden avulla voidaan tehdä tarkistuksia numero tai tekstiarvoille, esimerkiksi funktiot **DDV_MaxChars()** ja **DDV_MinMaxInt()**. Saat täydellisen luettelon näistä funktioista siirtymällä Visual C++:n ohjeen **Index-**välilehdelle ja kirjoittamalla **DDX_** (tai **DDV_**) **Keyword**-ruutuun.

Normaalisti määrittelet jäsenmuuttujan (ja näin ollen myös DDX-funktion) jokaista dialogissa olevaa käyttäjältä tietoja vastaanottavaa kontrollia kohden. Tarkistustietoja tarvitaan vain, jos se on tarpeen syötettävien tietojen rajaamista varten.

Seuraavassa harjoituksessa näet, kuinka DDX-jäsenmuuttujia ja tarkistus-kriteereitä asetetaan ClassWizardia käyttäen.

► **DDX-jäsenmuuttujien lisääminen ClassWizardia käyttäen**

1. Palaa MyApp-projektiin.
2. Avaa ClassWizard painamalla CTRL+W. Napauta **Member Variables** -välilehteä.
3. Napauta **Class name** -ruudussa **CConnectDialog**-luokkaa.
4. Napauta **Control IDs** -ruudussa **IDC_USERID**.
5. Napauta **Add Variable**. **Add Member Variable** -dialogi avautuu.
6. Kirjoita **Member variable name** -ruutuun **m_strUserID**.
7. Varmista, että **Category**-ruudussa näkyy **Value** ja **Variable type** -ruudussa näkyy **CString**.
8. Lisää muuttuja napauttamalla **OK**. Muuttujan nimi ja tyyppi näkyvät nyt valittuina **Control IDs** -ruudussa IDC_USERID:n vieressä. Tätä muuttujaa käytetään asetettaessa User ID -muokkausruudussa näkyvää tekstiä ja vastaan otettaessa käyttäjän kontrolliin kirjoittamia arvoja.
9. **Control IDs** -luettelon alapuolella on **Maximum Characters** -muokkausruutu. Luo tarkistusfunktio kirjoittamalla ruutuun **15**, jolloin syötettävä teksti voi olla pituudeltaan korkeintaan 15 merkkiä.

10. Lisää samat vaiheet toistaen **m_strPassword**-niminen CString muuttuja, joka on korkeintaan 15 merkkiä pitkä ja on yhteydessä **IDC_PASSWORD-** kontrolliin.
11. Lisää **int**-muuttuja **m_nAccess**, joka on yhdistetty kontrollitunnisteeseen **IDC_ACCESS**. Huomaa, että kokonaislukuarvoille voidaan määrätä pienin ja suurin arvo, jotka voidaan syöttää. Kirjoita **Minimum value** ruutuun **1** ja **Maximum value** ruutuun **5**.
12. Lisää BOOL-tyyppinen muuttuja **m_bConnect**, joka on yhteydessä kontrollitunnisteeseen **IDC_CHECKCONNECT**. Se asettaa kontrollin valituksi (TRUE) tai valitsemattomaksi (FALSE), ja muuttuu käyttäjän tekemien valintojen mukaan.
13. Sulje ClassWizard napauttamalla **OK**. Voit nyt kääntää ja käynnistää MyApp-sovelluksen.
14. Valitse **Connect**-komento **Data**-valikosta ja kirjoita käyttäjätunnus ja salasana. Kokeile, kuinka vaiheessa 9 asettamasi tarkistusarvo estää antamasta yli 15 merkkiä pitkää käyttäjätunnusta. Toteuttaakseen tämän toiminnon **DDV_MaxChars()**-funktio lähettää alustuksen yhteydessä kontrollille EM_LIMITTEXT-sanoman. Näin toteutettu arvojen tarkistus on välittömämpi ja käyttäjäystävällisempi kuin vaihtoehtoinen menetelmä, jossa arvot tarkistetaan sen jälkeen, kun käyttäjä on napauttanut **OK**:ta (tai **Connect** kuten tässä tapauksessa).
15. Esimerkkinä tyypillisestä tarkistuksesta kirjoita numero **6 Access level** -muokkausruutuun ja napauta **Connect**. Tähän kenttään liitetyn tarkistusfunktion pitäisi näyttää sanomaikkuna, joka varoittaa sinua väärästä arvosta ja palauttaa sinut sen jälkeen kontrolliin, jossa virheellinen arvo on. Todellisessa sovelluksessa tällainen valinta toteutettaisiin yhdistelmäruutuna, jossa on rajotettu määrä valintoja, joista käyttäjä voi valita yhden — ja näin ollen tarkistusfunktio olisi tarpeeton.
16. Sulje dialogi napauttamalla **Cancel**.

ClassWizard-toteutus DDX ja DDV -funktioille

Pääset katsomaan koodia, jonka ClassWizard on lisännyt DDX/DDV arkkitehtuurin toteutusta varten avaamalla ConnectDialog.h-tiedoston ja tutkimalla luokan määrittelyä. Näet, että ClassWizard on lisännyt seuraavat jäsenmuuttujat:

```
//{{AFX_DATA(CConnectDialog)
enum {IDD = IDD_CONNECTDIALOG};
CString   m_strUserID;
CString   m_strPassword;
BOOL      m_bConnect;
int       m_nAccess;
//}}AFX_DATA
```

Kuten kaikki ClassWizardin ylläpitämä koodi nämäkin määrittelyt on suljettu erityiseen //{{AFX_ kommenttilohkoon.

Etsi ConnectDialog.cpp-tiedostosta muodostin, joka sisältää seuraavan ClassWizard-koodin, joka määrittää muuttujien oletusarvot:

```
//{{AFX_DATA_INIT(CConnectDialog)
m_strUserID = _T("");
m_strPassword = _T("");
m_bConnect = FALSE;
m_nAccess = 0;
//}}AFX_DATA_INIT
```

Funktio, joka todella siirtää tietoja dialogin ja sovelluksen välillä on **CWnd::DoDataExchange()**. Tästä funktiosta saadaan aina ylikuormitettu versio, kun ClassWizardilla periytetään luokka **CDialog**-luokasta. ClassWizard päivittää ylikuormitettua **DoDataExchange()**-funktiota lisäämällä DDX ja DDV -funktiot.

CConnectDialog::DoDataExchange()-funktio on ConnectDialog.cpp-tiedostossa ja sen tulisi näyttää tällä hetkellä seuraavalta:

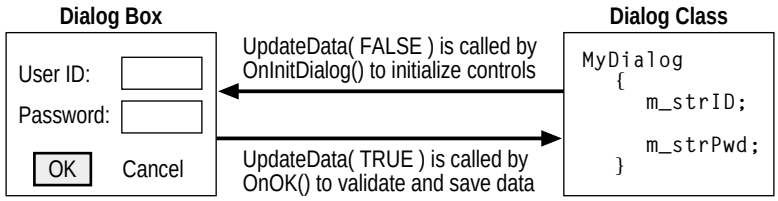
```
void CConnectDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CConnectDialog)
    DDX_Text(pDX, IDC_USERID, m_strUserID);
    DDV_MaxChars(pDX, m_strUserID, 15);
    DDX_Text(pDX, IDC_PASSWORD, m_strPassword);
    DDV_MaxChars(pDX, m_strPassword, 15);
    DDX_Check(pDX, IDC_CHECKCONNECT, m_bConnect);
    DDX_Text(pDX, IDC_ACCESS, m_nAccess);
    DDV_MinMaxInt(pDX, m_nAccess, 1, 5);
    //}}AFX_DATA_MAP
}
```

pDX parametri, joka välitetään DDX ja DDV -funktioille on MFC-luokan **CDataExchange**-objekti, joka edustaa juuri siirrettävänä olevaa dataa. **CDataExchange**-luokassa on jäsenmuuttuja m_bSaveAndValidate, joka kertoo siirron suunnan. Jos m_bSaveAndValidate-arvo on TRUE, tieto siirretään kontrollista ensin tarkistettavaksi ja sitten varastoitavaksi jäsenmuuttujiin. Jos m_bSaveAndValidate-arvo on FALSE, tietoa siirretään ruudulla olevan dialogin kontrolleihin.

DDX-ja DDV-prosessi

CWnd::UpdateData()-funktio kutsuu **DoDataExchange()**-funktiota. **UpdateData()** luo **CDataExchange**-objektin, jonka **DoDataExchange()** saa parametrinä ja välittää edelleen DDX/DDV-funktioille.

UpdateData()-funktio saa yhden BOOL-parametrin, joka ilmoittaa tiedonsiirron suunnan. **CDialog::OnInitDialog()**-funktio, jota kutsutaan dialogia luotaessa, kutsuu **UpdateData()**-funktiota *FALSE*-parametrillä. Näin dialogin kontrollit saadaan alustettua luokan jäsenmuuttujissa olevilla arvoilla. Kun käyttäjä napauttaa dialogin **OK**-painiketta, oletus **CDialog::OnOK()**-käsittelijä kutsuu **UpdateData()**-funktiota *TRUE*-parametrillä, jolloin tiedot tarkistetaan ja arvot talletetaan dialogiluokan kontrolleihin. Prosessi on esitetty kuvassa 5.1.



Kuva 5.1 Dialogin tietojen välitys ja tarkistaminen

Tiedonsiirto ja tarkistus voidaan suorittaa koodista koska vain kutsumalla **CWnd::UpdateData()**-funktiota. Kuvitellaan esimerkiksi, että sinulla on kuvankatseluohjelma, jossa käyttäjä voi muuttaa kuvan kokoa ja värien määrää modaalittoman dialogin kotrolleja käyttämällä. Päivitetyn kuvan tulisi ilmestyä heti käyttäjän muutettua valintoja. Tällaisessa tapauksessa **UpdateData()**-funktiota kutsutaan aina kontrollin arvon muuttuessa, jotta uudet arvot tallentuisivat ja olisivat näin sovelluksen piirtofunktioiden käytettävissä.

► **Alkuarvojen asettaminen dialogin kontrolleihin**

1. Palaa MyApp-projektiin.
2. Avaa **ClassView**-välilehdellä **CMyAppApp**-luokan kuvake.
3. Siirry muokkaamaan funktiota kaksoisnapauttamalla **OnDataConnect()**-kuvaketta.
4. Lisää seuraavat rivit **CConnectDialog**-objektin määrittelyn ja **DoModal()**-funktion kutsun väliin:

```
aCD.m_nAccess = 1;
aCD.m_bConnect = TRUE;
```

Funktion tulisi näyttää nyt kokonaisuudessaan seuraavalta:

```
void CMyAppApp::OnDataConnect()
{
    CConnectDialog aCD;
    aCD.m_nAccess = 1;
    aCD.m_bConnect = TRUE;
    aCD.DoModal();
}
```

5. Käännä ja käynnistä MyApp-sovellus. Valitse **Data**-valikosta komento **Connect**. Varmista, että jäsenmuuttujiin asettamasi arvot näkyvät dialogin kontrolleissa.

► **Arvojen asettaminen dialogin kontrolleihin**

1. Palaa **OnDataConnect()** funktioon.
2. Poista seuraava rivi:

```
aCD.DoModal();
```

3. Kirjoita sen paikalle seuraava koodi:

```
if(aCD.DoModal() == IDOK)
{
    CString strMessage;
    strMessage.Format("User %s logged in", aCD.m_strUserID);
    AfxMessageBox(strMessage);
}
```

Paluuarvo IDOK **DoModal()**-funktiolta merkitsee, että käyttäjä on sulkenut dialogin napauttamalla **OK**-painiketta ja että kontrolleihin syötetyt tiedot välitettiin niihin liitetyille tarkistusfunktioille. Käyttäjän syöttämät tiedot on varastoitu dialogiluokan DDX-jäsenmuuttujiin — tässä tapauksessa **User ID**-muokkausruutuun kirjoitettu arvo on nyt talletettuna CConnectDialog:: m_strUserID-muuttujaan.

4. Käännä ja käynnistä MyApp-sovellus ja varmista kokeilemalla **Data**-valikon **Connect**-komentoa, että käyttäjätunnus, jonka syötit, näkyy sanomaruudussa oikein.

Dialogin tietojen siirto ja tarkistaminen

Voit tehdä myös omia DDX- ja DDV-funktioita. DDX-funktio vaatii parametreinä osoittimen **CDataExchange**-objektiin, dialogissa olevan kontrollin tunnisteen ja dialogiluokan jäsenmuuttujan. DDV-funktio tarvitsee myös osoittimen **CDataExchange**-objektiin ja dialogiluokan jäsenmuuttujaan. Ne voivat ottaa

myös lisäparametreja tarkistuksia varten, kuten esimerkiksi **DDV_MinMaxInt()**, joka saa minimi- ja maksimiarvot määrittävät parametrit. DDV-funktio täytyy *aina* sijoittaa suoraan sen DDX-funktion jälkeen, johon se viittaa. Lisätietoja omien DDX/DDV-funktioiden tekemisestä saat tutkimalla Visual C++:n ohjetta Technical Note "TN026: DDX and DDV Functions".

Saatat joutua käyttämään omaa DDV-funktiota, jos haluat asettaa tarkistusehdot käyttämällä muuttujia vakioiden sijasta, tai jos haluat suorittaa ehdollisia tarkistuksia. Ajatellaanpa esimerkiksi seuraavaa koodia. Se suorittaa erilaisen tarkistuksen riippuen IDC_FEMALE-valintaruuden tilasta. Miesten ja naisten maksimi-ikä välitetään muuttujan avulla.

```
//{{AFX_DATA_MAP(CMyClass)
DDX_Check(pDX, IDC_FEMALE, m_bFemale);
DDX_Text(pDX, IDC_EDIT1, m_age);
//}}AFX_DATA_MAP
if (m_bFemale)
   DDV_MinMax(pDX, m_age, 0, m_maxFemaleAge);
else
   DDV_MinMax(pDX, m_age, 0, m_maxMaleAge);
```

Tästä huomataan myös se tärkeä seikka, että omat DDX/DDV-funktiot täytyy sijoittaa ClassWizardin //{{AFX_-kommenttilohkon *ulkopuolelle*.

Seuraavassa harjoituksessa lisätään oma tarkistusfunktio, joka vaatii käyttäjää syöttämään arvon **UserID**-kenttään.

► **Oman tarkistuksen lisääminen MyApp-sovellukseen**

1. Lisää seuraava globaali funktion määrittely ConnectDialog.h-tiedostoon, **CConnectDialog**-luokan määrittelyn *ulkopuolelle*:

   ```
   void PASCAL DDV_Required(CDataExchange * pDX, CString str);
   ```

2. Lisää seuraava **DDV_Required()**-funktion määrittely ConnectDialog.cpp-tiedoston loppuun:

   ```
   void PASCAL DDV_Required(CDataExchange * pDX, CString str)
   {
       if(pDX->m_bSaveAndValidate && str.IsEmpty())
       {
           AfxMessageBox("Please enter the User ID.");
           pDX->Fail();
       }
   }
   ```

   Huomaa, kuinka CDataExchange::m_bSaveAndValidate-jäsenmuuttujaa tutkimalla varmistetaan, että tarkistus on toiminnassa. Huomaa myös, kuinka funktiota **CDataExchange::Fail()** käytetään tarkistusprosessin keskeyt-

tämiseen ja fokuksen palauttamiseen siihen kontrolliin, jonka arvo ei läpäissyt tarkistusta.

3. Etsi **CConnectDialog::DoDataExchange()**-funktion toteutus ConnectDialog.cpp-tiedostosta. Valitse seuraavat rivit:

```
DDX_Text(pDX, IDC_USERID, m_strUserID);
DDV_MaxChars(pDX, m_strUserID, 15);
```

4. Leikkaa nämä kaksi riviä leikepöydälle niiden nykyisestä paikasta //{{AFX_DATA_MAP-kommenttilohkosta painamalla CTRL+X .
5. Liitä rivit //{{AFX_DATA_MAP-lohkon ulkopuolelle **DoDataExchange()**-funktion loppuun painamalla CTRL+V.
6. Lisää seuraavat rivit, joilla kutsutaan uutta **DDV_Required()**-funktiota, välittömästi äskeisten rivien jälkeen:

```
DDV_Required(pDX, m_strUserID);
```

DoDataExchange()-funktion tulisi nyt näyttää seuraavalta:

```
void CConnectDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CConnectDialog)
    DDX_Text(pDX, IDC_PASSWORD, m_strPassword);
    DDV_MaxChars(pDX, m_strPassword, 15);
    DDX_Check(pDX, IDC_CHECKCONNECT, m_bConnect);
    DDX_Text(pDX, IDC_ACCESS, m_nAccess);
    DDV_MinMaxInt(pDX, m_nAccess, 1, 5);
    //}}AFX_DATA_MAP

    DDX_Text(pDX, IDC_USERID, m_strUserID);
    DDV_MaxChars(pDX, m_strUserID, 15);
    DDV_Required(pDX, m_strUserID);
}
```

7. Käännä ja käynnistä MyApp-sovellus ja kokeile **Data**-valikon **Connect**-komentoa. Kokeile, mitä tapahtuu, jos yrität napauttaa **Connect**-painiketta ennen kuin syötät käyttäjätunnuksen.

Kontrollien alustaminen OnInitDialog()-funktion avulla

Kaikkien kontrollien alustaminen ei onnistu MFC:n normaaleilla DDX-funktioilla, eikä ole helposti suoritetavissa myöskään itse tehtyjen DDX-funktioiden avulla. Esimerkiksi **Connect to Data Source** -dialogi

MyApp-sovelluksessa näyttää listan saatavilla olevista tietolähteistä luetteloruudussa. Luetteloruudussa olevat tietolähteet muuttuvat, kun järjestelmään lisätään tai siitä poistetaan tietolähteitä. MFC sisältää funktiot **DDX_LBIndex** ja **DDX_LBString**, joiden avulla voit asettaa vaihtoehdot ja noutaa käyttäjän valinnan, mutta näitä funktioita ei voida käyttää luetteloruudun osien kanssa.

Tavallisin tapa luetteloruudun valintojen asettamiseen on ylikirjoittaa luokassa **CDialog::OnInitDialog()**-virtuaalifunktio. **OnInitDialog()** on funktio, joka kutsuu **CWnd::UpdateData()**-funktiota (ja lopulta DDX/DDV-funktioita), kun dialogiluokka alustetaan. **OnInitDialog()** on sopiva paikka omien alustusten tekemiseen dialogin kontrolleihin, koska se suoritetaan kontrolli-ikkunan luomisen jälkeen, mutta ennen kuin se avataan näytölle.

MFC sisältää kaikkia tavallisia Windows-kontrolleja vastaavat luokat. ClassWizardia käyttämällä voit luoda näitä luokkia vastaavia objekteja dialogiluokkasi jäseniksi. Nämä objektit yhdistetään dialogin kontrolleihin **DDX_Control()**-funktion avulla ja sijoitetaan **DoDataExchange()**-funktioon. Voit käyttää näitä objekteja dialogin kontrollien alustamiseen tai päivittämiseen.

Seuraavassa harjoituksessa lisätään MFC-luokan **CListBox** objekti dialogiluokkaan vastaamaan **Data Source** -luetteloruuta. Objektia käytetään luettelon alusta-miseen ja oletusvalinnan asettamiseen.

► **CListBox-jäsenmuuttujan lisääminen**

1. Avaa ClassWizard painamalla CTRL+W. Napauta **Member Variables** -välilehteä.
2. Valitse **Class name** -ruutuun **CConnectDialog**.
3. Valitse **Control IDs** -ruudusta **IDC_DSNLIST**.
4. Napauta **Add Variable**. **Add Member Variable** -dialogi avautuu.
5. Kirjoita **Member variable name** -ruutuun **m_lbDSN**.
6. Napauta **Category**-ruudussa **Control**.
7. Lisää muuttuja napauttamalla **OK**. Muuttujan nimi ilmestyy valittuna **Control IDs** -ruutuun IDC_DSNLIST jälkeen.
8. Sulje ClassWizard napauttamalla **OK**.

► **OnInitDialog()-funktion ylikirjoittaminen**

1. Avaa ClassWizard painamalla CTRL+W. Napauta **Message Maps** -välilehteä.
2. Valitse **Class name** -ruudussa **CConnectDialog**.
3. **CConnectDialog**-luokan nimen ollessa valittuna **Object IDs** -ruudussa, napauta **WM_INITDIALOG Messages**-ruudussa.
4. Napauta **Add Function** -painiketta.

5. Napauta **Edit Code** -painiketta. Huomaa, että ClassWizard lisää kantaluokan kutsun kohtiin, joissa se on tarpeen. **CDialog::OnInitDialog()**-funktiota kutsutaan niin, että **CWnd::UpdateData()**-funktiota kutsumalla voidaan alustaa kontrollit. Korvaa // TODO -kommentti funktion toteutuksesta seuraavalla koodilla:

```
m_lbDSN.AddString("Accounts");
m_lbDSN.AddString("Admin");
m_lbDSN.AddString("Management");
```

   Huomaa, kuinka lisäämääsi m_lbDSN-muuttujaa käytetään asetettaessa luettelon jäseniä.

Voit myös hakea käyttäjän tekemän valinnan **CListBox**-jäsenobjektin avulla. Voit hakea **CListBox::GetCurSel()**-funktiota käyttämällä nollasta alkavan indeksin, joka ilmaisee käyttäjän valinnan. Valintaa vastaava teksti voidaan sitten noutaa välittämällä indeksi **CListBox::GetText()**-funktiolle. Seuraavassa harjoituksessa haetaan käyttäjän valitseman tietolähteen nimi ja talletetaan se CMyAppApp:: m_strDSN-muuttujaan. Tähän muuttujaan tallennetaan tilarivillä näytettävä tietolähteen nimi. Paras paikka tälle koodille on **OK**-painikkeen käsittelijä, koska käyttäjä napauttaa **OK**-painiketta halutessaan vahvistaa valintansa. Oletuskäsit-telijää **CDialog::OnOK()** täytyy myös muistaa kutsua, koska se kutsuu **CWnd::UpdateData()**-funktiota.

► **Käyttäjän tekemän valinnan selvittäminen**

1. Avaa ClassWizard painamalla CTRL+W. Napauta **Message Maps** -välilehteä.
2. Valitse **Class name** -ruudusta **CConnectDialog**.
3. Valitse **Object IDs** -ruudussa **IDOK**. Napauta **Messages**-ruudussa **BN_CLICKED**.
4. Napauta **Add Function** -painiketta. Hyväksy funktion nimeksi **OnOK**.
5. Napauta **Edit Code** -painiketta. Korvaa funktion toteutuksesta // TODO-kommentti seuraavalla koodilla:

```
CMyAppApp * pApp = dynamic_cast<CMyAppApp *>(AfxGetApp());
ASSERT_VALID(pApp);
int nChoice = m_lbDSN.GetCurSel();


if(nChoice != LB_ERR)
{
    m_lbDSN.GetText(nChoice, pApp->m_strDSN);
    pApp->m_isDatabaseConnected = TRUE;
}
```

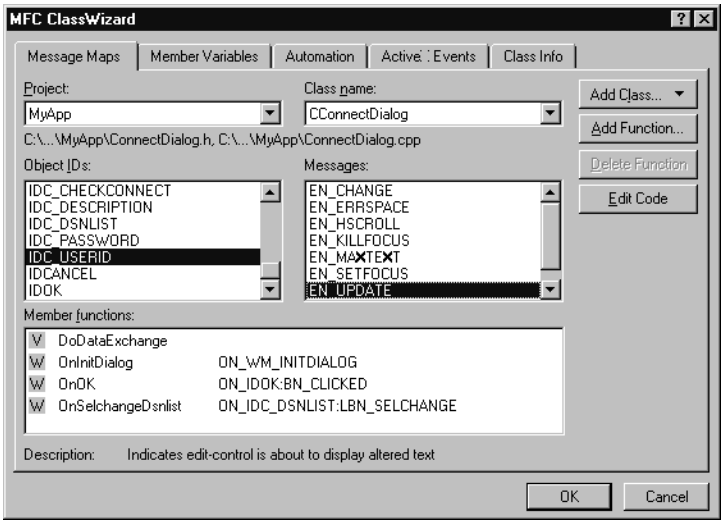
Voit nyt kääntää ja käynnistää MyApp-sovelluksen. Kokeile dialogia nähdäksesi, että luetteloruudun vaihtoehdot näkyvät oikein. Valitse tietolähteen nimi ja napauta **OK**. Valitsemasi tietolähteen nimen pitäisi näkyä tilarivillä.

Kontrollisanomien käsittely

OnOK()-funktio, jonka juuri teit, on esimerkki kontrollisanoman käsittelijästä. Luvussa 3 oppitunnilla 3 kerrottiin, että kontrollit ja muut lapsi-ikkunat lähettävät kontrollisanomia tiedottaakseen kantaikkunalle kontrollin ja käyttäjän välisestä vuorovaikutuksesta. **OnOK()**-funktio käsittelee BN_CLICKED-sanoman, joka lähetetään dialogille (WM_COMMAND-sanoman parametrinä), kun käyttäjä napauttaa **OK**-painiketta kerran.

Erityyppisiin kontrolleihin liittyy erilaisia kontrollisanomia. ClassWizardin **Message Maps** -välilehdeltä näet nopeasti, mitä sanomia dialogissasi oleviin kontrolleihin liittyy. Jos valitset dialogiluokan **Class Name** -ruudusta ja napautat kontrollin tunnistetta **Object IDs** -ruudusta, luettelo kontrolliin liitettävissä olevista sanomista tulee näkyviin **Messages**-ruutuun. Valitsemalla jonkin sanomatunnisteista saat sen kuvauksen näkyviin sivun alareunassa olevaan **Description**-kenttään.

Kuvassa 5.2 näet **MFC ClassWizard** -dialogin **Message Maps** -välilehden ja luettelon sen muokkausruutuun liittämistä sanomista (tässä tapauksessa **IDC_USERID**).



Kuva 5.2 ClassWizardin käyttäminen kontrollisanomien käsittelyyn

Tarkemman kuvauksen kontrollisanomasta saat hakemalla sanoman tunnisteen avulla Visual C++ -ohjeesta. Kontrollisanomien tunnisteet alkavat kontrollin

nimen lyhenteestä ja *N*_-merkeistä muodostetulla etuliitteellä, kuten taulukosta 5.1 selviää.

Taulukko 5.1 Sanomatunnisteiden etuliitteet

Sanomatunnisteen etuliite	Kontrollityyppi
BN_	Painike (Button)
CBN_	Yhdistelmäruutu (Combo box)
CLBN_	Valintaluettelo (Check list box)
EN_	Muokkausruutu (Edit control)
LBN_	Luetteloruutu (List box)
STN_	Seliteteksti (Static text control)

ClassWizard lisää sanomakarttaan jokaista kontrollityyppiä vastaavat ohjausmak-rot näille sanomille. Näiden makrojen nimet muodostetaan lisäämällä sanomatunnisteen alkuun etuliite *ON*_. Jos esimerkiksi ClassWizardia käyttäen luodaan käsittelijä EN_UPDATE sanomalle, joka tulee IDC_USERID muokkausruudulta (kuten kuvassa 5.2), ClassWizard lisää sanomakarttaan seuraavan merkinnän:

```
ON_EN_UPDATE(IDC_USERID, OnUpdateUserid)
```

Huomio **IDOK** ja **IDCANCEL** -painikkeiden BN_CLICKED-sanomat ohjataan suoraan **CDialog::OnOK()** ja **CDialog::OnCancel()** virtuaalifunktioiden ylikuormitetuille versioille. Sanomakarttaan merkittyjä makroja ei käytetä, jollei sanomien käsittelijäksi merkitä erinimistä funktiota. Yleensä tulisi käyttää **OnOK** ja **OnCancel** -funktioita.

MFC sisältää yleiset **ON_CONTROL** ja **ON_NOTIFY** -makrot, jotka sallivat mukautettujen sanomien käsittelyn. ClassWizard käyttää **ON_NOTIFY**-makroa uudempien Windows NT:n ja Windows 95/98:n mukana käyttöön tulleiden kontrollien yhteydessä.

Seuraavissa harjoituksissa tehdään käsittelijä, joka huolehtii **Data Source** -ruudun valinnan muutoksen seurauksena syntyvän sanoman käsittelystä. Käsittelijää käytetään ohjetekstin näyttämiseen luetteloruuden alla olevassa vain-luku-muokkausruudussa. Ensin luodaan CEdit-muuttuja, joka edustaa koodissa muokkausruutukontrollia.

► **CListBox-jäsenmuuttujan lisääminen**

1. Avaa ClassWizard painamalla CTRL+W. Napauta **Member Variables** -välilehteä.
2. Valitse **Class name** -ruudusta **CConnectDialog**.

3. Valitse **Control IDs** -ruudusta **IDC_DESCRIPTION**. Napauta **Add Variable**.
4. Kirjoita **Member variable name** -ruutuun **m_editDesc**.
5. Valitse **Category**-ruudusta **Control**. Lisää muuttuja napauttamalla **OK**.
6. Sulje ClassWizard napauttamalla **OK**.

► **OnSelChangeDsnlist()-kontrollisanoman käsittelijän lisääminen**

1. Avaa ClassWizard painamalla CTRL+W. Napauta **Message Maps** -välilehteä.
2. Valitse **Object IDs** -ruudusta **IDC_DSNLIST**. Valitse **Messages**-ruudusta **LBN_SELCHANGE**.
3. Napauta **Add Function**. Hyväksy **OnSelchangeDsnlist** funktion nimeksi.
4. Napauta **Edit Code**. Korvaa // TODO-kommentti funktion toteutusosassa seuraavalla koodilla:

```
int nCursel = m_lbDSN.GetCurSel();

switch(nCursel)
{
    case 0 : m_editDesc.SetWindowText("Accounting Data");
        break;

    case 1 : m_editDesc.SetWindowText("Administration Data");
        break;

    case 2 : m_editDesc.SetWindowText("Management Data");
        break;
}
```

   Huomaa, kuinka **CWnd::SetWindowText()**-funktiota käytetään tekstin asettamiseen muokkausruutuun.

5. Käännä ja käynnistä MyApp-sovellus. Varmista **Connect to Data Source** -dialogia kokeilemalla, että kuvausteksti vaihtuu, kun valintaa **Data Source** -ruudussa muutetaan.

Kontrollien käytön salliminen ja estäminen

Luvun 4 oppitunnilla 1 todettiin, että niiden valikko- ja työkalurivikomentojen, jotka eivät voi suorittaa tehtäväänsä, ei pitäisi olla valittavissa. Sama koskee myös dialogin kontrolleja.

Connect to Data Source -dialogi antaa tällä hetkellä käyttäjän yrittää yhdistymistä tietolähteeseen, vaikka yhtään tietolähdettä ei ole valittu ja kirjautumistietoja ei ole syötetty. Tarkistusfunktion käyttämisen sijasta tässä

tapauksessa olisi parempi, jos **Connect**-painiketta ei voi valita ennen kuin kaikki tarvittavat tiedot on syötetty. Seuraavassa harjoituksessa nähdään, kuinka **CWnd::EnableWindow()**-funktiota käyttäen sallitaan tai estetään kontrollin käyttäminen, ja kuinka kont-rollisanomakäsittelijää voidaan käyttää dialogin kontrollien tilan päivittämiseen, kun käyttäjä käsittelee dialogin tietoja.

► **Connect-painikkeen käytön salliminen ja estäminen**

1. Valitse ClassWizardin **Member Variables** -välilehdellä **CConnectDialog**-luokan **IDOK**-kontrollitunniste. Lisää CButton jäsenmuuttuja **m_bnConnect**.
2. Valitse ClassWizardin **Member Variables** -välilehdellä **CConnectDialog**-luokan **IDC_USERID**-kontrollitunniste. Lisää CEdit jäsenmuuttuja **m_editUserID**. Sulje ClassWizard napauttamalla **OK**.
3. Lisää seuraavat **Connect**-painikkeen käytön tarvittavien tietojen syöttämisen jälkeen mahdollistavat koodirivit **CConnectDialog::OnSelchangeDsnlist()**-funktion loppuun, juuri viimeisen aaltosulun eteen. (**CWnd::GetWindowText()** palauttaa kopioitujen merkkien määrän.)

```
char tempbuf[8];
if(m_editUserID.GetWindowText(tempbuf, 7))
    m_bnConnect.EnableWindow(TRUE);
```

4. Valitse ClassWizardin **Message Maps** -välilehdellä **CConnectDialog**-luokan **IDC_USERID**-kontrollitunniste. Valitse **EN_UPDATE**-sanoma ja lisää **OnUpdateUserid()**-kontrollisanomafunktio. Aloita funktion muokkaaminen napauttamalla **Edit Code**.
5. Lisää seuraavat koodirivit **OnUpdateUserid()**-funktioon:

```
char tempbuf[8];
if(m_lbDSN.GetCurSel() != LB_ERR)
{
    if(m_editUserID.GetWindowText(tempbuf, 7))
        m_bnConnect.EnableWindow(TRUE);
    else
        m_bnConnect.EnableWindow(FALSE);
}
```

6. Lisää **CConnectDialog::OnInitDialog()** funktion loppuun ennen return-komentoa seuraava rivi, joka varmistaa, että **Connect**-painike on oikeassa tilassa, kun dialogi avautuu:

```
OnUpdateUserid();
```

7. Koska tämän kentän tarkastusfunktiota ei enää tarvita, voit poistaa **DDV_Required()**-funktion ConnectDialog.cpp-tiedostosta, sekä sen määrittelyn ConnectDialog.h-tiedostosta. Poista **DDV_Required()** kutsu ConnectDialog.cpp-tiedoston **CConnectDialog::DoDataExchange()**-

funktiosta. Palauta seuraavat rivit ClassWizardin hallintaan siirtämällä ne // {{AFX_DATA_MAP-kommenttilohkon sisälle (varmista, että ne pysyvät yhdessä):

```
DDX_Text(pDX, IDC_USERID, m_strUserID);
DDV_MaxChars(pDX, m_strUserID, 15);
```

8. Käännä ja käynnistä MyApp-sovellus. Varmista **Connect to Data Source** -dialogia kokeilemalla, että **Connect**-painike on käytössä vain, jos tietolähde on valittu ja **UserID**-kenttä täytetty.

Dialogin käyttäminen sovelluksen tietojen muokkaamiseen

Tässä käytännön harjoituksessa lisätään muutamia jäsenmuuttujia MyApp-sovelluksen dokumenttiluokkaan. Muista, että MFC-sovelluksessa dokumenttiluokka on oikea säilytyspaikka sovelluksen tiedoille. Harjoituksessa tehdään dialogi, jonka avulla sovelluksen tietomuuttujien arvoja voidaan muuttaa. Sinun tulisi käydä tämä harjoitus läpi ennen kuin jatkat eteenpäin, koska se toimii pohjana seuraavien lukujen esimerkeille ja harjoituksille.

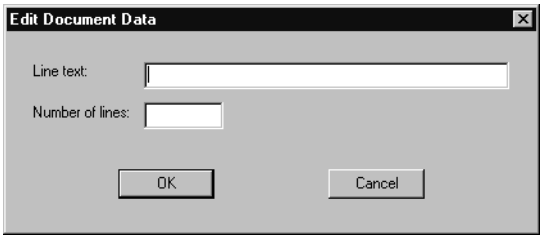
► **Sovelluksen tietomuuttujien lisääminen**

1. Avaa MyApp-projekti. Napauta ClassViewissä hiiren kakkospainikkeella **CMyAppDoc**-luokkaa ja napauta **Add Member Variable**. Lisää julkinen int-muuttuja **m_nLines**.
2. Aloita koodin muokkaaminen kaksoisnapauttamalla **CMyAppDoc**-muodostinta.
3. Aseta m_nLines-muuttujan alkuarvoksi **20**. Liitä 60 merkkiä pitkä merkkijono m_string-muuttujaan niin, että muodostin näyttää suunnilleen tältä:

```
CMyAppDoc::CMyAppDoc()
{
    m_nLines = 20;
    m_string =
        "This is a very long string designed to take up lots of space";
}
```

4. Lisää menueditoria käyttämällä uusi komento **Data**-valikon alkuun. Komennon otsikoksi asetetaan **&Edit** ja kehote merkkijonoksi "Edit document data." Hyväksy tunniste **ID_DATA_EDIT**, joka on luotu oletuksena.
5. Luo dialogieditoria käyttäen uusi dialogimalli tunnisteelle **IDD_EDITDATA**. Dialogin otsikoksi tulee **Edit Document Data** ja se siihen tulee kaksi selitetekstiä otsikoituina **Line text:** ja **Number of lines:**, jotka vastaavat muokkausruutuja IDC_EDIT_LINETEXT ja IDC_EDIT_NUMLINES.

Kontrollit tulisi järjestellä kuten kuvassa 5.3. Valitse **Number**-valintaruutu IDC_EDIT_NUMLINES-kontrollin **Styles**-ominaisuussivulta.



Kuva 5.3 Edit Document Data -dialogi

6. Luo ClassWizardia käyttämällä dialogiluokka **IDD_EDITDATA**-dialogille. Luokan nimeksi tulee **CEditDataDialog**, ja se periytetään luokasta **CDialog**. Lisää kaksi muokkausruutuja vastaavaa jäsenmuuttujaa: CString-muuttuja **m_strLineText** ja UINT-muuttuja **m_nLines**.
7. Lisää seuraava koodirivi MyAppDoc.cpp-tiedoston alkuun muiden #include-lauseiden joukkoon:

```
#include "EditDataDialog.h"
```

8. Lisää ClassWizardia käyttämällä käsittelijä **OnDataEdit()** objektitunnisteelle **ID_DATA_EDIT**. Funktion tulisi käsitellä COMMAND-sanoma ja se lisätään **CMyAppDoc**-luokkaan.
9. Lisää seuraava koodi **OnDataEdit()**-funktion runkoon:

```
CEditDataDialog aDlg;

aDlg.m_nLines = m_nLines;
aDlg.m_strLineText = m_string;

if(aDlg.DoModal())
{
    m_nLines = aDlg.m_nLines;
    m_string = aDlg.m_strLineText;
    UpdateAllViews(NULL);
}
```

10. Käännä ja käynnistä MyApp-sovellus. Valitse **Edit Data**-valikosta ja tarkista, että **Edit Document Data** -dialogi avautuu oikein.

Ominaisuussivujen toteuttaminen

Luvussa 4 opit, että ominaisuussivut ovat dialogeja, joita käytetään erityisesti objektien, kuten sovelluksen tai näkymän nykyisen valinnan ominaisuuksien asettamiseen. Ominaisuussivulla on kolme pääosaa: dialogi, vähintään yksi sivu ja jokaisen sivun yläreunassa oleva välilehti, jota napauttamalla käyttäjä valitsee sivun. Ominaisuussivut ovat käyttökelpoisia tilanteissa, joissa on muutettavana joukko samanlaisia asetuksia tai ominaisuuksia. Ominaisuussivun avulla suuri joukko tietoja voidaan yhdistää helposti käsitettävään muotoon. Visual C++:n **Project Settings** -dialogi on hyvä esimerkki ominaisuussivusta.

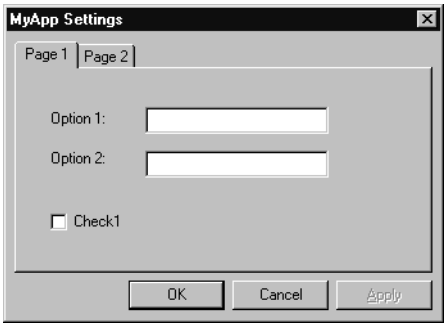
MFC käyttää ominaisuussivujen toteuttamiseen kahta luokkaa: **CPropertySheet**-luokkaa, joka edustaa dialogia ja **CPropertyPage**-luokkaa, joka edustaa ominaisuussivun yksittäisiä sivuja. Ominaisuussivu tehdään näitä luokkia käyttäen seuraavasti:

1. Luodaan dialogimalliresurssi jokaiselle ominaisuussivulle. Niiden kaikkien ei tarvitse olla saman kokoisia, mutta niiden ulkoasu kannattaa pitää mahdollisimman yhtenäisenä, jolloin sivuista tulee intuitiivisesti helppokäyttöisiä. Omi-naisuussivun dialogimalliresurssi tulisi luoda niin, että reunus on tyypiltään **Thin**, sekä **Child** ja **Disabled** ominaisuudet ovat asetettuja. Annettu otsikko ilmestyy sivun välilehteen.
2. Luodaan ClassWizardilla jokaista ominaisuussivun dialogimallia kohden **CPropertyPage**-luokasta periytetty luokka.
3. Luodaan ClassWizardilla jäsenmuuttujat ominaisuussivun arvoja varten ja asetetaan vastaavat DDX ja DDV -funktiot.
4. Luodaan lähdekoodissa **CPropertySheet**-objekti. Yleensä **CPropertySheet**-objekti muodostetaan ominaisuusikkunan avaavan komennon käsittelijässä. Jokaiselle sivulle luodaan oma objekti.
5. Jokaista lisättävää ominaisuussivua kohden kutsutaan funktiota **CPropertySheet::AddPage()** ja annetaan parametriksi kunkin ominaisuussivun osoite.
6. Luodaan modaalinen ominaisuusikkuna funktiolla **CDialog::DoModal()**, tai modaaliton ominaisuussivu funktiolla **CDialog::Create()**.

Sovelluskehys lisää modaaliseen ominaisuusikkunaan oletuksena **OK**, **Cancel** ja **Apply** -painikkeet, ja käsittelee jokaisen ominaisuussivun kontrollien DDX:t ja DDV:t. Modaalinen ominaisuusikkuna kannattaa yleensä tehdä niin, että ensin luodaan pinoon väliaikainen objekti, johon lisätään **CPropertyPage**-objektit, ja kutsutaan sitten funktiota **DoModal()**. Jos haluat lisätä ominaisuusikkunaan muita kontrolleja, luo ClassWizardia käyttäen oma **CPropertySheet**-luokasta periytetty luokka. Modaalittomien ominaisuusikkunoiden kanssa näin täytyy menetellä joka tapauksessa, koska **Create()**-funktio ei luo oletuksena yhtään kontrollia ominaisuusikkunaan.

Lisätietoja tästä aiheesta saat Visual C++:n ohjeesta “Adding Controls to a Property Sheet”. Varmista, että **Search**-välilehden **Search titles only** -valintaruutu on valittuna, kun teet hakua. Näin saat tuloksena vain halutun otsikon.

Seuraavassa harjoituksessa tehdään **MyApp Settings** -ominaisuusikkuna MyApp -sovellukselle. Tällä ominaisuusikkunalla on kaksi välilehteä, kuten näet kuvassa 5.4. Sivuilla olevat kontrollit eivät tee mitään — niiden sisältämiä tietoja ei käsitellä tässä harjoituksessa.



Kuva 5.4 **MyApp Settings** -ominaisuusikkuna

► **Page 1 -dialogimallin luominen**

1. Valitse MyApp-projektissa **ResourceView** ja avaa MyApp-resurssikansio.
2. Napauta hiiren kakkospainikkeella Dialog-kansiota. Valitse **Insert Dialog**.
3. Uusi tyhjä dialogi avautuu. Poista **OK** ja **Cancel** -painikkeet valitsemalla ne ja painamalla DELETE.
4. Aloita dialogin ominaisuuksien muokkaaminen painamalla ENTER. Kirjoita **ID**-ruutuun **IDD_PPAGE1**. Kirjoita **Caption**-muokkausruutuun **Page 1**.
5. Valitse **Styles**-välilehti. Valitse **Style**-ruudusta **Child**.
6. Valitse **Border**-ruudusta **Thin**.
7. Varmista, että **Title bar** -valintaruutu ei ole valittuna.
8. Valitse **More Styles** -välilehti. Valitse **Disabled** -valintaruutu.
9. Lisää sivulle pari kontrollia — voit käyttää kuvaa 5.4 mallina. Lisää vähintään yksi muokkausruutu ja anna sille tunnisteeksi **IDC_EDIT_PP1**.

► **CPage1-luokan luominen**

1. **IDD_PPAGE1**-dialogin ollessa avoinna dialogieditorissa, avaa ClassWizard painamalla CTRL+W.
2. Avaa **New Class** -dialogi napauttamalla **OK**.
3. Kirjoita **Name**-muokkausruutuun **CPage1**.
4. Valitse **Base Class** -ruudusta **CPropertyPage**.

5. Luo **CPage1**-dialogi napauttamalla **OK**.
6. Sulje ClassWizard napauttamalla **OK**. Sulje dialogieditori.

Luo toinen ominaisuussivumalli. Erota se ulkoisesti ensimmäisestä muuttamalla kontrollien kokoa ja lisäämällä kontrolleja. Anna mallille tunniste **IDD_PPAGE2** ja otsikko **Page 2**. Varmista, että sillä on samat tyylimäärittelyt kuin ensimmäisellä-kin sivulla. Tee tästä sivusta **CPropertyPage**-luokasta periytetty luokka **CPage2**.

Seuraavaksi tehdään ominaisuusikkuna, joka sisältää edellä tehdyt kaksi ominaisuussivua. Ominaisuusikkuna avataan **View**-valikkoon lisättävällä **Settings** -komennolla.

► **Ominaisuusikkunan avaavan Settings-komennon lisääminen**

1. Lisää menueditoria käyttämällä uusi komento **View**-valikkoon. Kirjoita otsikoksi **Setti&ngs** ja kehotteeksi **Application settings**. Anna editorin luoda tunniste **ID_VIEW_SETTINGS**. Sulje menueditori.
2. Avaa ClassWizard. Lisää **CMainFrame**-luokkaan komennon **ID_VIEW_ SETTINGS** käsittelijä **OnViewSettings()**.
3. Lisää seuraava koodi **OnViewSettings()**-funktion runkoon:

```
CPropertySheet PropSheet;
CPage1  pp1;
CPage2  pp2;

PropSheet.AddPage(&pp1);
PropSheet.AddPage(&pp2);

PropSheet.SetTitle("MyApp Settings");
PropSheet.DoModal();
```

4. Lisää seuraavat rivit CMainFrame.cpp-tiedoston alkuun:

```
#include "Page1.h"
#include "Page2.h"
```

5. Käännä ja käynnistä MyApp-sovellus. Avaa ominaisuusikkuna kokeilemista varten napauttamalla **View**-valikon **Settings**-komentoa. Huomaa, kuinka ominaisuussivut on sovitettu ominaisuusikkunaan ja kuinka otsikoita käytetään välilehtien tunnisteina. Huomaa myös, kuinka sovelluskehys automaattisesti lisää **Apply**-painikkeen.

Ominaisuusikkunan Apply-painikkeen käsitteleminen

OK-painike sulkee ominaisuusikkunan, sekä tallentaa ja tarkastaa kaikkien ominaisuussivujen tiedot. **Apply**-painike puolestaan antaa käyttäjälle

mahdollisuuden tallentaa ja tarkistaa tietoihin tehdyt muutokset ilman, että ominaisuusikkuna sulkeutuu. Tämä on käytännöllinen tapa, jos käyttäjä haluaa saattaa yhden ominaisuussivun asetukset voimaan ennen kuin jatkaa asetusten tekemistä toisella ominaisuussivulla.

Apply-painikkeen käyttö on oletuksena estetty. **Apply**-painike saadaan käyttöön käyttäjän syötettyä tietoja tai muutettua jonkin kontrollin tilaa kutsumalla funktio-ta **CPropertyPage::SetModified(TRUE)**. **Apply**-painikkeen napauttaminen ai-heuttaa **CPropertyPage::OnApply()**-käsittelijän kutsumisen, jolloin tiedot talletetaan ja tarkistetaan sulkematta ominaisuusikkunaa.

Jos päätät olla käyttämättä **Apply**-painiketta ominaisuusikkunassasi, sinun ei silti tarvitse poistaa sitä. Microsoftin suunnitteluohjeiden mukaan se voidaan jättää paikoilleen.

Seuraavassa harjoituksessa nähdään, kuinka **Apply**-painike saatetaan käyttöön käyttäjän syötettyä tietoja muokkausruutuun.

► **Apply-painikkeen käyttöön saattaminen**

1. Avaa ClassWizard ja valitse **Message Maps** -välilehti. Napauta **CPage1**.
2. Valitse **Object IDs** -ruudusta **IDC_EDIT_PP1**. Valitse **Messages**-ruudusta **EN_UPDATE**.
3. Napauta **Add Function**. Hyväksy funktion nimeksi **OnUpdateEditPp1**.
4. Napauta **Edit Code**. Korvaa funktiosta // TODO-kommentti seuraavalla koodirivillä:

```
SetModified(TRUE);
```

5. Käännä ja käynnistä MyApp-sovellus. Kirjoita muutamia merkkejä muokkausruutuun, joka vastaa IDC_EDIT_PPI-resurssia. Varmista, että **Apply**-painike tulee kirjoituksen aikan käytettäväksi. Napauta **Apply**-painiketta ja huomaa, kuinka se poistetaan automaattisesti käytöstä.

Oppitunnin yhteenveto

DDX:ää voidaan käyttää sovelluksen dialogeissa näkyvien kontrollien alkuarvojen asettamiseen ja käyttäjän syöttämien tietojen noutamiseen sovelluksen käyttöön. DDV:n avulla voidaan käyttäjän syöttämät tiedot tarkistaa ennen niiden välittämistä sovellukselle. DDX:n ja DDV:n toteutus on suurelta osin automatisoitu ClassWizardin avulla.

DDX:n perustoiminnot lisätään luomalla ClassWizardilla dialogin kontrolleihin kir-joitettuja tietoja vastaavat jäsenmuuttujat. Yksinkertainen DDV-toiminto voidaan lisätä määrittelemällä ClassWizardilla yksinkertaiset kelpoisuussäännöt muuttujia luotaessa. Samalla kun määritellään muuttujat ja kepoisuussäännöt, ClassWizard lisää ennalta määriteltyihin MFC-funktioihin kutsut

ylikuormitettuun **DoDataExchange()**-funktioon, joka suorittaa tietojen tarkistamisen ja siirron.

DoDataExchange()-funktiota kutsuu **CWnd::UpdateData()**-funktio. **CDialog ::OnInitDialog()** alustaa kontrollit kutsumalla **UpdateData()**-funktiota. **CDialog::OnOK()** puolestaan kutsuu **UpdateData()**-funktiota saadakseen kontrollien sisältämät arvot ja suorittaakseen tarkistukset.

DDX/DDV-toimintoja voi lisätä kirjoittamalla omia mukautettuja funktioita. Mu-kautetut DDX ja DDV -funktiot tulisi sijoittaa ClassWizardin ylläpitämän koodilohkon ulkopuolelle. Kaikki DDX ja DDV -funktiot vaativat parametrinä tämänhetkisen tiedon välityksen sisältöä edustavan **CDataExchange**-objektin. Luokka sisältää jäsenmuuttujan m_bSaveAndValidate, joka osoittaa tiedonsiirron suunnan ja **Fail()**-funktion, joka keskeyttää tiedonsiirtoprosessin ja palauttaa fokuksen siihen kontrolliin, jossa tarkistuksessa hylätty tieto on.

Kontrollit, joiden alustaminen DDX-funktion avulla on vaikeaa, voidaan alustaa ylikuormittamalla dialogiluokan **CDialog::OnInitDialog()**-funktio. Dialogin tietoja voidaan käsitellä myös ylikuormitetun **CDialog::OnOK()**-funktion kautta.

MFC sisältää jokaista Windowsin peruskontrollia vastaavan luokan. Käyttämällä ClassWizardia voit tehdä näiden kontrolliluokkien objekteista dialogiluokan jäsen-muuttujia. Nämä objektit yhdistetään dialogin kontrolleihin **DoDataExchange()**-funktion **DDX_Control()**-funktion avulla. Näitä kontrolliobjekteja käyttämällä voidaan dialogin kontrollit alustaa ja päivittää.

ClassWizardilla voidaan tehdä käsittelijät kontrollien kantaikkunalleen (dialogille) lähiettämien sanomien käsittelemistä varten. Nämä käsittelijät toteutetaan yleensä osana dialogiluokkaan. Käsittelijöiden tehtävänä on usein huolehtia kontrollien käytön sallimisesta ja estämisestä käyttäjän käyttäessä dialogia.

Ominaisuusikkunat ovat monisivuisia dialogeja, joiden avulla tietoja voidaan ryh-mitellä helposti käsiteltävään muotoon. MFC:ssä ominaisuusikkunat toteutetaan kahden luokan avulla. **CPropertySheet**-luokka edustaa dialogia ja **CPropertyPage**-luokka ominaisuusikkunan yksittäistä sivua. Ominaisuusikkunaa tehtäessä täytyy ensin tehdä dialogimalli jokaiselle sivulle tai välilehdelle, ja sen jälkeen luoda **CPropertyPage**-luokasta periytetty luokka. Lähdekoodissa luodaan **CPropertySheet**-objekti ja lisätään ominaisuussivut kutsumalla **CPropertySheet ::AddPage()**-funktiota niin monesti kuin on tarpeen. Yleensä **CPropertySheet**-objekti luodaan pinoon ominaisuusikkunan avaavan komennon seurauksena.

Sovelluskehys luo ominaisuusikkunaan oletuksena **Apply**-painikkeen, jonka käyt-tö on estetty. **Apply**-painikkeen avulla käyttäjä voi toimittaa ja tarkistaa tekemän-sä muutokset sulkematta ominaisuussivua. Jos sovelluksessa käytetään **Apply** -painiketta, sen käyttäminen tulee sallia heti, kun käyttäjä muuttaa

dialogin kont-rolleissa olevia tietoja. **Apply**-painikkeen käyttö sallitaan kutsumalla **CPropertyPage::SetModified(TRUE)**-funktiota.

Oppitunti 2: Sovelluksen tietojen näyttäminen ja tulostaminen

Luvun 3 oppitunnilla 4 näit, kuinka MFC:n **CView**-luokasta periytettyä luokkaa käytettiin muodostettaessa kuvaa sovelluksen tiedoista. Opit, että MFC sisältää **CDC**-luokan, joka kapseloi piirtopinnan ja joukon piirtopinnalla toimivia funktioita, joilla voidaan muodostaa kuva tulostuslaitteelle. Opit myös kuinka sovelluskehys välittää **CDC**:stä johdetun nykyistä tulostuslaitetta vastaavan piirtopintaobjektin **CView::OnDraw()**-funktiolle. Näin **OnDraw()**-funktio toimii sovelluksen piirtokoodin keskityspaikkana. Samaa koodia voidaan käyttää riippumatta siitä, onko kohde sovelluksen ikkuna, esikatseluikkuna vai tulostuslaite.

Tällä oppitunnilla opit lisää **CView** ja **CDC** -luokista periytettyjen luokkien käytöstä. Opit myös, kuinka MFC:n piirtotyökaluluokkia käytetään sovelluksen tietojen lähettämiseen näytölle tai tulostimelle.

Tämän oppitunnin jälkeen:

- Tiedät, kuinka loogista koordinaatistoa käyttämällä saadaan sovelluksen tiedot näkymään oikein kaikilla piirtopinnoilla, jotka tukevat Windows Graphical Device Interface (GDI) rajapintaa.
- Tiedät, Windowsin piirtotilat ja kuinka niitä käytetään.
- Tiedät, kuinka MFC-sovelluksessa toteutetaan vieritys.
- Tiedät, kuinka MFC:n piirtotyökaluluokkia käytetään piirtopinnalle piirrettäessä.
- Tiedät, kuinka tulostus ja tulostuksen esikatseluprosessit toimivat ja kuinka mukautettuja tulostustoimintoja tehdään.

Oppitunnin arvioitu kesto: 40 minuuttia

Ennen tämän oppitunnin aloittamista täytyy sovelluksen piirtofunktiota muuttaa niin, että se näyttää “Hello World!”-tekstin sijasta sovelluksen tietoja.

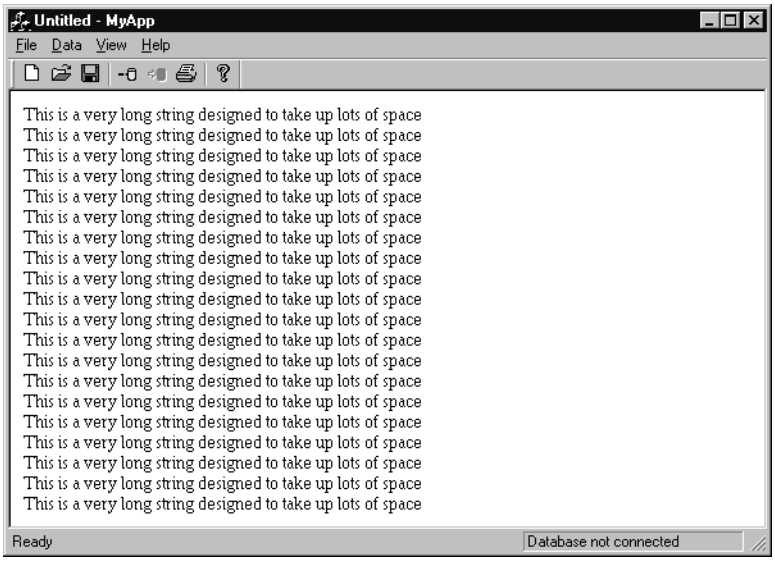
► OnDraw()-funktion muokkaaminen

1. Etsi **CMyAppView::OnDraw()**-funktion toteutus. Korvaa nykyinen versio seuraavalla koodilla:

```
void CMyAppView::OnDraw(CDC* pDC)
{
```

```
    CMyAppDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CFont aFont;
    aFont.CreateFont(16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        FF_ROMAN, 0);

    CFont * pOldFont = pDC->SelectObject(&aFont);

    CSize TextSize = pDC->GetTextExtent(pDoc->m_string);
    int nLinePos = 10;

    for(int i = 0; i < pDoc->m_nLines; i++)
    {
        pDC->TextOut(10, nLinePos, pDoc->m_string);
        nLinePos += TextSize.cy;
    }

    pDC->SelectObject(pOldFont);
}
```

2. Käännä ja käynnistä MyApp-sovellus.
3. Valitse **Edit**-valikosta **Data**.
4. Kirjoita **Edit Document Data** -dialogin **Line text** -muokkausruutuun 60 merkkiä pitkä merkkijono ja numero **20** muokkausruutuun **Number of lines**.
5. Sulje dialogi napauttamalla **OK**. Kirjoittamasi tekstin tulisi ilmestyä näyttöön 20 kertaa kuten kuvassa 5.5.



Kuva 5.5 MyApp-sovelluksen testituloste

Koordinaatistojärjestelmä

Graafiset laitteet muodostavat tulosteensa kaksiulotteiseen koordinaatistoon. Näytölle tulostettaessa käytetään mitoituksessa pikseleitä ja tulostimissa väripisteitä. GDI-piirtofunktioita käytetään Windows-sovelluksissa tulostuksen määrittelyyn samalla tavoin laitteesta riipumatta. GDI:ssä piirrokset mitoitetaan ja skaalataan abstrakteilla yksiköillä, joita kutsutaan *loogisiksi yksiköiksi* (logical units).

Esimerkiksi seuraava **OnDraw()**-funktiosta oleva koodi alustaa fonttiobjektin näytölle tulostamista varten. Fontin korkeudeksi tulee 16 loogista yksikköä.

```
aFont.CreateFont(16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, FF_ROMAN, 0));
```

Seuraava koodi siirtyy ensin sovelluksen koordinaatiston pisteeseen (100,200), ja piirtää sitten 200 loogista yksikköä pitkän pystyviivan pisteeseen (300,200).

```
pDC->MoveTo(100, 200);
pDC->LineTo(300, 200);
```

Loogisella yksiköllä ei ole mitään määrättyä mittaa. Loogisen yksikön mittojen määrittäminen käyttävän laitteen mittojen mukaan on sinun tehtäväsi. GDI toteuttaa yhtenäisen koordinaatistojärjestelmän, tarjoten näin standardin liittymän moniin erityyppisiin näyttöihin, tulostimille ja muille sovelluksen mahdollisesti käyttämille tulostuslaitteille. Voit määritellä *kohdistustavan* (mapping mode), joka määrittää, kuinka sovelluksen piirtoalueen loogiset yksiköt siirretään laitteiston piirtoalueelle (laitteiston yksikköinä). GDI takaa, että siirto tuottaa yhtenäisen tulosteen kaikille liitetyille kohdelaitteille.

GDI sisältää kahdeksan piirtotilan määrittelyä, jotka määrittelevät loogisen koordinaatiston ja laitteiston koordinaatiston suhteen. Oletuskohdistustapa on MM_TEXT. Se siirtää loogisen yksikön yhdeksi pikseliksi riippumatta laitteen resoluutiosta. Tästä voi seurata ongelmia tulostettaessa, kuten seuraava harjoitus havainnollistaa.

► **MM_TEXT-kohdistustavan vaikutusten tutkiminen**

1. Käynnistä MyApp-sovellus.
2. Valitse **Data**-valikosta **Edit**, ja kirjoita noin 60 merkkiä pitkä merkkijono näytettäväksi 20 kertaa.
3. Jos tulostin on asennettuna, tulosta sovelluksen tiedot valitsemalla **Print**-komento **File**-valikosta. Jos tulostinta ei ole asennettu, valitse **Print Preview** -komento **File**-valikosta, jolloin näet tiedot siinä muodossa kuin ne tulostuisivat näytölle.

Huomaat, että tulostimen tulostus on erittäin pientä. Tämä johtuu näytön ja tulostimen erilaisesta resoluutiosta. Piirtofunktiot skaalaavat tulosteen kohdelaitteen resoluution mukaan. Näin ollen kirjain, joka on korkeudeltaan 16 pikseliä, on 800x600 näytöllä noin 0.21 tuumaa korkea. 600 x 600 dpi (pistettä tuumalle) tulostimella korkeus on vain noin 0.026 tuumaa.

MM_TEXT on yksi kuudesta kiinteästä (fixed) kohdistustavasta, jotka ovat ennalta määrättyjä suhteita loogisten fyysisten yksiköiden välillä. Muut kiinteät kohdistustavat määrittävät loogisten yksiköiden mitat laitteen mittoina. MM_LOENGLISH määrittää esimerkiksi, että looginen yksikkö vastaa 0.01 tuumaa kohdelaitteella. Kiinteät kohdistustavat on lueteltu taulukossa 5.2.

Taulukko 5.2 Kiinteät kohdistustavat

Piirtotila	Yksi yksikkö vastaa
MM_TEXT	1 laitteen pikseli
MM_LOENGLISH	0.01 tuumaa
MM_HIENGLISH	0.001 tuumaa
MM_LOMETRIC	0.1 millimetriä
MM_HIMETRIC	0.01 millimetriä
MM_TWIPS	1/1440 tuumaa

Huomio Windowsissa ei pystytä tosiasiassa antamaan näytölle fyysisiä mittoja, koska näytön ajuri ei tunne monitorin todellisia mittasuhteita. Näin ollen edellisessä taulukossa ilmoitetut tuumat ja millimetrit ovat ideaalisen monitorin mittoihin perustuvia loogisia arvoja. Toisaalta tulostimissa mitat ovat fyysisesti tarkkoja.

Kiinteiden kohdistustapojen lisäksi voidaan käyttää myös itse määriteltäviä kohdistustapoja MM_ISOTROPIC ja MM_ANISOTROPIC. Näiden kohdistustapojen suhteita ei ole ennalta määritelty, joten suhde loogisten ja fyysisten mittojen välillä on määriteltävä itse. Kun valitaan MM_ISOTROPIC tapa, GDI suorittaa kohdistuksen niin, että loogiset yksiköt kohdistuvat samalla tavoin sekä *x*- että *y*-suunnassa. MM_ANISOTROPIC-tapa kohdistaa *x*- ja *y*-koordinaatit toisistaan riippumatta.

Loogisten ja fyysisten yksiköiden välinen suhde määritellään kahden suorakaiteen avulla, joista toinen määrittelee loogisen tilan mittasuhteet (*ikkuna*, window) ja toinen laitteen tilan (*katseluikkuna*, viewport). Ikkunan mittasuhteet määritellään loogisina yksikköinä käyttämällä **CDC::SetWindowExt()**-funktiota. Katseluikku-nan mitat asetetaan käyttäen funktiota **CDC::SetViewportExt()** ja ne määritel-lään laitteen yksikköinä. Tiedot nykyisen laitteen mitoista saadaan käyttämällä **CDC::GetDeviceCaps()**-funktiota.

Seuraava koodi asettaa MM_LOENGLISH-tapaa vastaavan kohdistustavan:

```
pDC->SetMapMode(MM_ANISOTROPIC);
pDC->SetViewPortExt(pDC->GetDeviceCaps(LOGPIXELSX),
    pDC->GetDeviceCaps(LOGPIXELSY);
pDC->SetWindowExt(100, -100);
```

GetDeviceCaps()-funktion *LOGPIXELSX* ja *LOGPIXELSXY* parametrit palauttavat kuvapisteiden määrän loogista tuumaa kohden näytön korkeus- ja leveyssuunnassa. Koodissa määritellään, että 1 x 1 -tuuman neliö laitteella vastaa sovel-luksen piirtoalueella 100 x 100 loogista yksikköä. Tai toisin sanoen yksi looginen yksikkö vastaa 0.01 tuumaa tulostuslaitteella.

Huomaa, että **SetWindowExt()**-funktion *y*-parametrilla on negatiivinen arvo. Ole-tuksena laitteen koordinaatisto noudattaa Windowsin käytäntöä, että origo on ikkunan vasemmassa yläkulmassa ja *y*:n arvot kasvavat alaspäin mentäessä. Jos ikkunan ja katseluikkunan *y*-arvojen välinen suhde määritellään käänteiseksi, piir-tofunktiot voivat käyttää perinteistä matemaattista koordinaatistoa, jossa y:n arvo kasvaa ylöspäin mentäessä. Muista, että näytöllä oleva alkupiste (sijainti *(0, 0)*) sijoitetaan oletuksena ikkunan vasempaan yläkulmaan, joten jos käytät ylöspäin kasvavia *y*-koordinaatteja, piirtojälki ei tule suoraan näkyviin. Piirtäminen pitää tehdä näkyvän ikkunan yläpuolelle. Voit joko käyttää käänteisiä *y* koordinaatteja tai voit siirtää origon loogisen ikkunan vasempaan alakulmaan käyttämällä **CDC::SetViewportOrg()**-funktiota.

Kiinteät kohdistustavat käyttävät matemaattista käytäntöä, jonka mukaan *x*-koordinaatit kasvavat oikealle ja *y*-koordianaatit kasvavat ylöspäin mentäessä. MM_TEXT-kohdistustapa noudattaa Windowsin käytäntöä, jossa *y* koordinaatin arvo kasvaa alaspäin mentäessä.

Kun näkymään on valittu sopiva kohdistustapa, voidaan GDI-piirtofunktioilla muodostaa tuloste käyttäen loogisia yksiköitä. Muunnos loogisten ja fyysisten (laite) koordinaatistojen välillä hoidetaan automaattisesti. Joissain tilanteissa muunnos täytyy tehdä itse käyttämällä **CDC**-metodeja **LPtoDP()** ja **DPtoLP()**, koska tietynlaiset tiedot saadaan vain laitteiston koordinaatteina. Esimerkiksi hiiren sijainti hetkellä, jolloin sen painiketta napautetaan, saadaan **CView:: OnLButtonDown()**-käsittelijän parametreiksi kahtena fyysisenä koordinaattina. Jos halutaan selvittää, onko hiirtä napautettu loogisen piirroksen tietyllä alueella, täytyy koordinaatit muuntaa loogisiksi.

Vieritysnäkymät

Kun piirretään loogiseen tilaan, rajoituksena on vain koordinaatiston lukualue. Koordinaattien täytyy Windows 95:ssä ja 98:ssa olla välillä –32,768:sta 32,767:än. Käytettäesä MM_LOENGLISH kohdistustapaa, tämä alue vastaa

lähes 280 neliömetriä fyysistä piirtoaluetta — mikä on huomattavasti enemmän kuin yhdessäkään käytettävässä tulostuslaitteessa.

Ne ikkunat, jotka eivät pysty näyttämään sovelluksen tietoja yhdessä ruudussa, tulisi varustaa vierityspalkeilla.Vierityspalkit tulisi toteuttaa aina, kun käyttäjällä on mahdollisuus ikkunan koon muuttamiseen ja ne pitää tuoda näyttöön heti, kun ikkunan kehys alkaa peittää työalueella olevia tietoja.

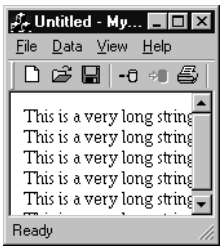
Onneksi MFC:n avulla vierityksen toteuttaminen dokumentti/näkymä-sovellukseen on helppoa. Muistanet luvun 2 oppitunnilta 1, että MyApp-sovelluksen näkymäluokka periytettiin **CScrollView**-luokasta. Voit lisätä vierityksen mihin tahansa näkymäluokkaan käyttämällä **CWnd**-vieritysfunktioita ja käsittelemällä WM_VSCROLL ja WM_HSCROLL -sanomat, mutta **CScrollView**-luokka helpottaa vierityksen toteuttamista:

- Huolehtimalla ikkunan ja katseluikkunan kohdistustavoista.
- Vierittämällä näkymää automaattisesti vierityspalkkien sanomien mukaan.

Seuraavassa harjoituksessa tutustutaan **CScrollView**-funktion vieritysominaisuuksiin.

► **MyApp-sovelluksen vieritystoimintoihin tutustuminen**

1. Käynnistä MyApp-sovellus.
2. Valitse **Data**-valikosta **Edit,** ja kirjoita noin 60 merkkiä pitkä rivi toistettavaksi 20 kertaa.
3. Muuta ikkunan kokoa niin, että se on kuvan 5.6 mukainen.



Kuva 5.6 MyApp-sovelluksen vieritystoimintojen kokeileminen

Huomaa, että vaikka vieritys käynnistyi automaattisesti, se ei toimi aivan niin kuin pitäisi. Kuvasta 5.6 näet, että tekstirivien loppu on näkymän ulkopuolella ja vaakavierityspalkki ei ole käytössä. Vierityspalkin oletetaan tulevan näyttöön heti ikkunan reunan peitettyä ikkunassa olevia tietoja. Huomaat myös, että kavennettuasi ikkunaa edelleen vaakavierityspalkki ilmestyy näkyviin, mutta vieritys ei edelleenkään toimi.

Tällainen toiminta johtuu siitä, että vieritysnäkymän loogisen koon määrittäminen ja kohdistustavan valinta ovat ohjelmoijan vastuulla. Kun AppWizard-sovellus luodaan periyttämällä näkymä luokasta **CScrollView**, AppWizard lisää seuraavan koodin ylikuormitettuun **CView::OnInitialUpdate()**-metodiin:

```
CSize sizeTotal;
// TODO: calculate the total size of this view
sizeTotal.cx = sizeTotal.cy = 100;
SetScrollSizes(MM_TEXT, sizeTotal);
```

Vieritettävän näkymän looginen koko ja kohdistustapa valitaan **CScrollView::SetScrollSizes()**-funktiota käyttämällä. Oletus metodi **OnInitialUpdate()** asettaa vieritettävän näkymän mitoiksi 100 x 100 loogista yksikköä. MM_TEXT-kohdistustavalla tämä vastaa näytöllä 100 x 100 kuvapistettä — joka on noin yksi neliötuuma 800 x 600 monitorilla. **SetScrollSizes()**-funktiolla on lisäprametrejä, joiden avulla asetetaan *juovan koko* (line size) ja *sivun koko* (page size). Juovan koko määrittää matkan, jonka vieritys etenee vaaka- tai pystysuunnassa käyttäjän napautettua vieritysnuolta. Sivun koko on vieritettävä matka, kun käyttäjä napauttaa vierityspalkkia, muttei vieritysnuolta. Näille parametreille lasketaan oletusarvot suhteessa ikkunan kokonaismittoihin.

Kun MyApp-sovellus on riittävän suuri näet tekstin kokonaisuudessaan. Jos kokoa muutetaan niin, että näkymästä tulee **SetScrollSizes()**-funktion avulla määriteltyä kokoa pienempi, vierityspalkit tulevat näkyviin. Vierityspalkin skaalaus ja sijainti lasketaan myös vieritettävän näkymän koon perusteella. Tämä oli syynä siihen, että kokeiltaessa MyApp-sovelluksen vieritystoimintoja, loogisen 100 x 100 ikkunan ulkopuolella olevien tietojen vierittäminen näyttöön oli mahdotonta.

Tehtäessä vieritysnäkymää dokumentti/näkymä-sovellukseen, täytyy ensin päättää, kuinka suuren loogisen piirtoalueen sovelluksen tietojen esittäminen vaatii. Tämän jälkeen määritellään sopiva kohdistustapa piirroksen skaalaamiseksi tulostuslaitteelle, ja asetetaan arvot kutsumalla **SetScrollSizes()**-funktiota luokan **OnInitialUpdate()**-funktiossa. Seuraava koodin pätkä havainnollistaa, kuinka asetetaan 8.5 x 11 tuuman kokoinen näkymä:

```
sizeTotal.cx = 850;
sizeTotal.cy = 1100;
SetScrollSizes(MM_LOENGLISH, sizeTotal);
```

Tämä menetelmä on sopiva, jos dokumentin tietojen koko voidaan määritellä kiinteästi. On kuitenkin sovelluksia, kuten tekstinkäsittelyohjelmat, joissa dokumentin koolle ei aseteta rajoituksia. Kun käyttäjä lisää uuden tekstirivin täytyy loogisen ikkunan kokoa muuttaa ja laskea vierityspalkkien skaalaus uudelleen. Tällaisessa tapauksessa täytyy **SetScrollSizes()**-funktiota kutsua säännöllisesti aina dokumentin tietojen muuttuessa.

Seuraavassa harjoituksessa käytetään **SetScrollSizes()**-funktiota CMyAppView-näkymän muokkaamiseen niin, että vierityspalkit ilmestyvät näkyviin samalla, kun näytettäviä tietoja jää ikkunan ulkopuolelle näkymättömiin. **SetScrollSizes()**-funktiota kutsutaan piirtofunktiosta niin, että vieritysala sovitetaan näytettävän merkkijonon mukaan. Funktio asettaa kohdistustavaksi MM_LOENGLISH, joten tulostus näkyy yhtenäisesti sekä näytöllä että tulostimella.

► **Vierityksen koon asettaminen MyApp-sovelluksessa**

1. Avaa ClassViewissä **CMyAppView**-luokan kuvake.
2. Aloita **OnDraw()**-metodin muokkaaminen kaksoisnapauttamalla sen kuvaketta.
3. Lisää seuraavan rivin alle

```
CSize TextSize = pDC->GetTextExtent(pDoc->m_string);
```

nämä rivit:

```
CSize scrollArea =
    CSize(TextSize.cx, TextSize.cy * pDoc->m_nLines);

// Allow a margin
scrollArea += CSize(20, 20);

SetScrollSizes(MM_LOENGLISH, scrollArea);
```

Vierityksen mittasuhteet ovat suhteessa näytettävän tiedon kokoon ja ne määritellään merkkijonon pituuden ja näytettävien rivien määrän perusteella.

Kun käytetään MM_TEXT-kohdistustapaa, *y*-koordinaatin arvo kasvaa alaspäin mentäessä ja muut kiinteät kohdistustavat seuraavat matemaattista käytäntöä, jonka mukaan *y*:n arvo kasvaa ylöspäin mentäessä. Kun kohdistustapa vaihdetaan oletuksena olevasta MM_TEXT-tavasta MM_LOENGLISH-tapaan, täytyy *y*- koordinaattien arvot kääntää näyttökoodissa niin, että niiden arvot kasvavat edettäessä sivulla alaspäin.

► ***y*-koorinaattien kääntäminen näyttökoodissa**

1. Etsi seuraavat rivit **OnDraw()**-funktiosta:

```
int nPos = 10;
```

```
for(int i = 0; i < pDoc->m_nLines; i++)
{
    pDC->TextOut(10, nPos, pDoc->m_string);
    nPos += TextSize.cy;
}
```

2. Muuta **nPos** arvoksi **–10**.
3. Muuta **– =** operaattoriksi, joka kasvattaa **nPos**-arvoa silmukassa. Funktion tulisi näyttää nyt kokonaisuudessaan seuraavalta:

```
void CMyAppView::OnDraw(CDC* pDC)
{
    CMyAppDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CFont aFont;
    aFont.CreateFont(16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        FF_ROMAN, 0);

    CFont * pOldFont = pDC->SelectObject(&aFont);

    CSize TextSize = pDC->GetTextExtent(pDoc->m_string);

    CSize scrollArea =
        CSize(TextSize.cx, TextSize.cy * pDoc->m_nLines);

    // Allow a margin
    scrollArea += CSize(20, 20);

    SetScrollSizes(MM_LOENGLISH, scrollArea);

    int nPos = -10;

    for(int i = 0; i < pDoc->m_nLines; i++)
    {
        pDC->TextOut(10, nPos, pDoc->m_string);
        nPos -= TextSize.cy;
    }

    pDC->SelectObject(pOldFont);
}
```

4. Käännä ja käynnistä MyApp-sovellus.
5. Valitse **Data**-valikosta **Edit** ja lisää näyttöön 20 pitkää tekstiriviä.
6. Pienennä sovelluksen ikkunaa ja varmista, että vierityspalkit ilmestyvät näkyviin, kun ikkunasta tulee pienenmpi kuin näytettävät tiedot vaativat.

Muuta kokeeksi rivien pituutta ja määrää, jolloin vierityspalkit skaalataan automaattisesti vastaamaan näytettäviä tietoja.

7. Tulosta dokumentti tai käytä tulostuksen esikatselua nähdäksesi, että tuloste on kooltaan järkevä.

Piirtopinnalle piirtäminen

Sovelluskehys välittää **OnDraw()**-funktiolle piirtopinnan osoittimen. Piirtopinta edustaa sovellusikkunan työaluetta. **CDC**-luokka, joka on kaikkien MFC:n piirtopintaobjektien kantaluokka, sisältää joukon funktioita, joiden avulla voidaan piirtää viivoja, tulostaa tekstiä ja käsitellä bittikartan kuvioita. Olet jo tutustunut muutamiin näistä funktioista kuten **LineTo()**, **Rectangle()** ja **TextOut()** tämän luvun aiemmissa esimerkeissä.

Piirtofunktiot toimivat yhteistyössä MFC:n piirtotyökaluluokkien kanssa. Näihin luokkiin kuuluvat muiden muassa **CPen** (viivojen piirtäminen), **CBrush** (alueiden täyttäminen), **CFont** ja **Cbitmap**. Tietyn tyyppinen piirtotyökalu valitaan piirtopinnalle käytettäväksi piirtofunktioiden kanssa. Näin ollen **CDC::Rectangle()**-funktio piirtää piirtopinnalle suorakaiteen käyttämällä nykyistä **CPen** määritystä ja täyttää sen käyttämällä nykyistä **CBrush**-työkalua.

Piirtotyökaluobjektien suositeltava käyttötapa on seuraava:

1. Luo grafiikkaobjektit kahdessa vaiheessa. Esittele objekti ensin ja luo se sitten tyyppikohtaisen funktion kuten **CPen::CreatePen()** avulla.
2. Valitse objekti nykyisellä piirtopinnalla funktiota **CDC::SelectObject()** käyttäen. **SelectObject()**-funktio on useilla tavoilla ylikuormitettu vastaamaan valittavissa olevia erityyppisiä grafiikkaobjekteja.
3. **SelectObject()**-funktio palauttaa alunperin valitun grafiikkaobjektin osoittimen. Luo tämän osoittimen arvon tallettamiseen soveltuva osoitin.
4. Kun nykyistä grafiikkaobjektia ei enää tarvita, palauta vanha grafiikkaobjekti piirtopinnalle käyttämällä tallennettua osoitinta. Piirtopinta tulisi jättää siihen tilaan, jossa se oli aloitettaessa.

MyApp-sovellukseen lisätty **OnDraw()**-funktio on hyvä esimerkki tästä menettelytavasta. Seuraavat rivit esittelevät ja luovat nykyiselle piirtopinnalle uuden fontin:

```
CFont aFont;
aFont.CreateFont(16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, FF_ROMAN, 0);
CFont * pOldFont = pDC->SelectObject(&aFont);
```

aFont-fonttia käytetään seuraavissa **CDC::TextOut()**-funktion kutsuissa tekstin esittämiseen piirtopinnalla. Kun funktio ei enää tarvitse fonttia, se hävittää

aFont-objektin ja käyttää seuraavaa koodiriviä palauttaessaan objektin, jonka osoitin on varastoitu **pOldFont**-osoittimeen:

```
pDC->SelectObject(pOldFont);
```

Vihje Erillisten piirtopinnan objektien tallentamisen ja palauttamisen sijaan voit käyttää **CDC**:n metodeja **SaveDC()** ja **RestoreDC()** koko piirtopinnan tallentamiseen ja palauttamiseen.

Tässä kurssimateriaalissa kerromme vain, kuinka sovelluksen tietojen passiivinen esittäminen toteutetaan. Jos halutaan luoda näkymä, joka tukee objektien valintaa, leikkaamista, kopiointia, liittämistä, näytöllä olevien objektien muokkaamisen sallimista, hiirellä piirtämistä ja niin edelleen, täytyy tehdä melkoinen määrä lisätyötä. Visual C++:n mukan toimitettava DRAWCLI-esimerkkisovellus on hyvä esimerkki monien tällaisten toimintojen toteutuksesta.

Tulostusprosessi

Luvun 3 oppitunnilla 4 opittiin, että kuvan muodostaminen sovelluksen tiedoista tulostettavalle sivulle on yksinkertaista, koska sama **OnDraw()**-funktio soveltuu sekä näytölle että paperille tulostamiseen. Funktiot **CView::OnPrint()** ja **CWnd::OnPaint()** kutsuvat molemmat **OnDraw()**-funktiota muodostaessaan tulostetta.

Tulostettaessa sovelluskehys toteuttaa seuraavat tehtävät:

- Näyttää **Print**-dialogin.
- Luo piirtopintaobjektin tulostimelle.
- Ilmoittaa tulostimelle **CDC::StartDoc()**-funktiota kutsumalla, että kaikki seuraavat sivut ovat samaa työtä, kunnes **CDC::EndDoc()**-funktiota kutsutaan. Tällä varmistetaan, että yhtä sivua pitempien tulostusten väliin ei tule muita tulostustöitä.
- Ilmoittaa toistuvasti tulostimelle kunkin sivun alkamisesta ja loppumisesta kutsumalla **CDC::StartPage()** ja **CDC::EndPage()** funktioita.
- Kutsuu näkymän ylikuormitettavia funktioita niin monta kertaa kuin on tarpeen.

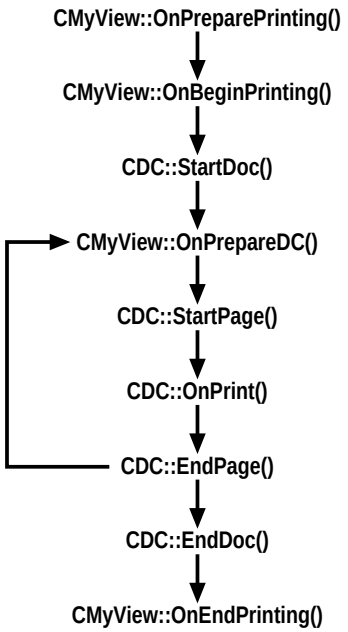
Useimmat funktioista voidaan ylikuormittaa. Ylikuormitettavia funktioita ovat esimerkiksi tulostettavan dokumentin sivuttamisesta huolehtiva funktio, GDI-

resurssit tulostukselle kohdentava funktio ja funktio, joka lähettää tulostimelle koodin vaihtomerkejä ennen sivun tulostamista. Näiden funktioiden avulla voidaan tulostusprosessia mukauttaa. Taulukossa 5.3 on lueteltu funktiot, jotka voidaan ylikuormittaa (kaikki ovat **CView**-luokan jäseniä).

Taulukko 5.3 Ylikuormitettavissa olevat tulostusfunktiot

Nimi	Toiminta
OnPreparePrinting()	Antaa mahdollisuuden sijoittaa tietoja **Print**-dialogiin muokkaamalla **CPRINTINFO**-struktuuria — käytetään yleensä dokumentin pituuden määrittämiseen. Välittää **CPRINTINFO**-rakenteen **CView::DoPreparePrinting()**-metodille, joka avaa dialogin ja luo tulostimen piirtopinnan.
OnBeginPrinting()	Mahdollistaa fonttien tai muiden tulostustyössä käytettävien GDI-resurssien allokoinnin.
OnPrepareDC()	Mahdollistaa tietyn sivun piirtopinnan ominaisuuksien muuttamisen. Jos dokumentin pituutta ei ole määritelty, suoritetaan dokumentin lopun tarkistus.
OnPrint()	Mahdollistaa tietyn sivun tulostamisen. Tavallisesti tämä funktio vain muodostaa tulosteen kutsumalla **OnDraw()**-funktiota. Ylikuormittamalla tämä funktio saadaan tuottamaan näytöllä olevasta tulosteesta poikkeava paperituloste.
OnEndPrinting()	Mahdollistaa GDI-resurssien pois kytkemisen.

Kuvassa 5.7 on esitetty koko tulostusprosessi ja järjestys, jossa näitä funktioita kutsutaan.



Kuva 5.7 MFC-tulostusprosessi

Print-dialogi antaa nykyisellään mahdollisuuden valita tulostettavata sivut. Tämä toiminto pitää poistaa käytöstä sovelluksissa, joissa tuloste on vain yhden sivun pituinen. Seuraavassa harjoituksessa lisätään funktioon **CMyAppView:: OnPreparePrinting()** -koodi, joka asettaa tulostettavien sivujen maksimimääräksi yhden sivun. Tämä aiheuttaa sen, että sivualueen valinta **Print**-dialogissa poistuu käytöstä.

► **Tulostettavien sivujen maksimimäärän asettaminen**

1. Käynnistä MyApp-sovellus. Varmista, että sivumäärän valintatoiminto on käytössä valitsemalla **File**-valikosta **Print**.
2. Avaa ClassViewissä **CMyAppView**-luokan jäsenet näkyviin.
3. Aloita metodin muokkaaminen kaksoisnapauttamalla **OnPreparePrinting()**-kuvaketta.
4. Kirjoita seuraava rivi funktion runkoon ennen return-komentoa:

```
pInfo->SetMaxPage(1);
```

5. Käännä ja käynnistä MyApp-sovellus. Valitse **Print**-komento **File-valikosta** ja varmista, että sivumäärän valinta -toiminto ei ole enää käytettävissä.

Tulostuksen esikatselu

Kun käyttäjä valitsee **Print Preview** -komennon **File**-valikosta, sovelluskehys luo **CPreviewDC**-objektin. Aina kun sovellus suorittaa tulostimen piirtopintaan vaikuttavan toimenpiteen, sovelluskehys suorittaa vastaavan toimenpiteen esikatselun piirtopinnalle. Jos sovelluksessa esimerkiksi valitaan tulostuksessa käytettävä kirjasinlaji, sovelluskehys valitsee esikatseluun tulostimen kirjasintyyliä muistuttavan kirjasinlajin. Sovelluksen lähettäessä tulosteita kirjoittimelle, sovelluskehys lähettää tulosteen näytölle.

Esikatselu eroaa tulostamisesta tavassa, jolla dokumentin sivut piirretään. Tulostettaessa sovelluskehys suorittaa silmukkaa, kunnes tietty määrä sivuja on tulostettu. Esikatselussa näytetään yksi tai kaksi sivua, jonka jälkeen sovellus jää odottamaan. Seuraavia sivuja ei näytetä ennen kuin käyttäjä napauttaa joko **Next Page** tai **Previous Page**. Esikatselun aikana sovelluksen täytyy myös vastata WM_PAINT-sanomiin, aivan samoin kuin tavanomaisessa näyttötilassakin.

OnPreparePrinting()-funktiota kutsutaan esikatseluun siirryttäessä, aivan samoin kuin tulostustyötä aloitettaessakin. **CPRINTINFO**-struktuuri, joka funktiolle välitetään, sisältää useita arvoja, joita asettamalla voidaan esikatselutoiminnon ominaisuuksia määritellä. Esimerkiksi asettamalla **m_nNumPreviewPages**-jäsenen arvo voidaan määritellä, näyttääkö esikatselu yhden vai kaksi sivua kerrallaan.

Esikatselun mukauttaminen

Voit suhteellisen helposti mukauttaa esikatselun toimintaa ja ulkoasua monin tavoin. Mukautettavia ominaisuuksia ovat esimerkiksi seuraavalla sivulla luetellut.

- Määrätään esikatselu näyttämään vierityspalkit, jotta siirtyminen mille tahansa dokumentin sivulle on helppoa.
- Määrätään esikatselu huomioimaan kohta, jossa käyttäjä on dokumentissa ollut niin, että esikatselu alkaa nykyiseltä sivulta.
- Määritetään erilaiset alustukset esikatselulle ja tulostukselle.

Oppitunnin yhteenveto

Windows GDI on sovellusten ja monien erilaisten tulostuslaitteiden välinen abstraktiotaso. Sovelluksen graafinen tulostus mitoitetaan ja skaalataan käyttäen loogisia yksiköitä. GDI sisältää koordinaatistojen kohdistusjärjestelmän, joka määrittää, kuinka sovelluksen tulosteen loogiset yksiköt suhteutetaan laitteiston piirtotilaan. Näin varmistetaan tulostuksen yhdenmukaisuus käytettävästä laitteistosta riippumatta.

GDI määrittelee kahdeksan kohdistustapaa, joiden avulla määritellään loogisten ja laitteiston kooridinaattien suhde. Voit valita jonkin kuudesta kiinteästä kohdistustavasta, jotka käyttävät ennalta kiinteästi määrättyä loogisten ja laitteiston mittayksiköiden välistä suhdetta, tai kahdesta vapaasti määriteltävästä kohdistustavasta, jotka antavat sinun määritellä loogisten ja laitteistoyksiköiden välisen suhteen.

Ikkunaan tulisi lisätä vierityspalkit aina, kun ikkunassa ei pystytä kerralla näyttämään sovelluksen tulostetta kokonaan. MFC:n avulla vierityksen lisääminen dokumentti/näkymä-sovellukseen on helppoa, kun sovelluksen näkymäluokka periytetään MFC:n **CScrollView**-luokasta. Ohjelmoijan vastuulla on vieritettävän näkymän koon ja kohdistustavan valinta niin, että sovelluskehys tietää, milloin ja miten vieritys tulee toteuttaa.

MFC sisältää joukon piirtotyökaluluokkia, joita käytetään yhteistyössä GDI piirtofunktioiden kanssa muodostettaessa tulostusta piirtopinnalle. Yleensä piirtotyökaluobjektit luodaan ja mukautetaan sovelluksen tarpeiden mukaan. Näitä objekteja valitaan piirtopinnalle käyttämällä **CDC::SelectObject()**-funktiota, joka palauttaa osoittimet aiemmin valittuna olleeseen objektiin. Piirtopinta tulee palauttaa tätä osoitinta käyttäen alkuperäiseen tilaansa piirtofunktion lopussa.

Tulostettaessa sovelluskehys kutsuu funktioita tietyssä järjestyksessä. Kaikki funktiot ovat **CView**-luokan virtuaalifunktioita. Voit mukauttaa tulostusprosessia ylikuormittamalla näitä funktioita. Yleisin tehtävä on tulostettavan dokumentin sivuttaminen. Muita tehtäviä ovat esimerkiksi tulostuksessa tarvittavien GDI-resurssien varaaminen ja tulostustilan vaihtaminen ohjauskoodien avulla ennen

sivun tulostamista. Voit myös käyttää osaa näistä funktioista esikatseluikkunalle ominaisten asetusten tekemiseen.

Oppitunti 3: Monisäikeisyyden hyödyntäminen

Luvun 4 oppitunnilla 2 opit, että *prosessi* on suoritettavan ohjelman ilmentymä. *Säie* on prosessin sisällä oleva suorituspolku, joka on pienin koodin osa, jonka suorittamien prosessorissa voidaan aikatauluttaa. Säie toimii prosessin osoite-alueella ja käyttää prosessille osoitettuja resursseja. Kaikilla prosesseilla on vähin-tään yksi suoritettava säie, jota kutsutaan *pääsäikeeksi* (primary thread). Voit luoda sen lisäksi *toissijaisia säikeitä* (secondary threads), joiden avulla voit hyö-dyntää 32-bittisten Windows käyttöjärjetelmien moniajoa.

Tällä oppitunnilla opit, kuinka voit tehostaa sovelluksen toimintoja ja toimintaa luomalla toissijaisia säikeitä MFC:n luokkia ja globaaleja funktioita käyttäen.

Tämän oppitunnin jälkeen:

- Tiedät, milloin käytetään monisäikeisiä ohelmointitekniikoita.
- Tunnet kaksi säietyyppiä, joita voidaan luoda MFC:n avulla.
- Tunnet MFC-luokan **CWinThread**-roolin toissijaisten säikeiden luomisessa.
- Tiedät, kuinka luodaan ja tuhotaan toissijaisia säikeitä MFC-sovelluksessa.
- Tiedät, kuinka MFC:n synkronointiluokkien avulla kontrolloidaan säikeiden pääsyä jaettuihin tietoihin ja resursseihin.

Oppitunnin arvioitu kesto: 60 minuuttia

Monisäikeiset sovellukset

Sovelluksessa voidaan käyttää useita säikeitä aina, kun useiden erillisten tehtävien samanaikaisella suorittamisella voidaan parantaa sovelluksen suorituskykyä. Aja-tellaan esimerkiksi tekstinkäsittelyohjelmaa, joka varmistaa avoimen dokumentin viiden minuutin välein. Käyttäjän dokumenttiin syöttämien tietojen käsittelemises-tä vastaa pääsäie. Sovellus voi luoda erillisen toissijaisen säikeen, joka huolehtii automaattisen varmistuksen aikatauluttamisesta ja suorittamisesta. Toissijaisen säikeen luominen estää pitkien dokumenttien varmistusprosessia häiritsemästä sovelluksen käyttöliittymän toimintaa.

Tilanteista, joissa useiden säikeiden käyttäminen voi parantaa sovelluksen suorituskykyä ovat esimerkiksi:

- **Aikataulutetut (aikaohjatut) toiminnot** Tekstinkäsittelyohjelma-esimerkissä automaattisen varmistuksen suorittava säie suoritetaan viiden

minuutin välein. Säikeiden aikataulutus voidaan asettaa Win32-sovelluksissa millisekunnin tarkkuudella.

- **Tapahtumaohjatut toiminnot** Säikeen voi käynnistää toiselta säikeeltä tuleva signaali. Näin toimitaan esimerkiksi valvontajärjestelmässä, jossa virheenkirjaussäie aktivoituu vasta, kun jokin toisista säikeistä ilmoittaa virhetilasta.
- **Hajautetut toiminnot** Kun tietoja täytyy kerätä (tai jakaa) useille tietokoneille, on järkevää luoda jokaiselle pyynnölle oma säie, jolloin nämä tehtävät voidaan suorittaa rinnakkaisesti omissa kehyksissään.
- **Priorisoidut tehtävät** Win32-säikeille voidaan asettaa *prioriteetti*, joka mää-rittelee suhteellisen suoritusajan, jonka säikeiden aikataulutus kyseiselle säi-keelle varaa. Joskus sovelluksen toimivuutta voidaan lisätä jakamalla tehtäviä toimintoja niin, että käyttöliittymää hoidetaan korkealle priorisoidussa säi-keessä ja taustatyöt matalalle priorisoidussa säikeessä.

Monisäikeisyys MFC:ssä: CWinThread-luokka

Kaikkia säikeitä MFC-sovelluksissa edustavat **CWinThread**-objektit. Näin myös sovelluksen pääsäiettä, joka on toteutettu **CWinApp**-luokasta periytetyssä luo-kassa. **CWinApp** periytyy suoraan **CWinThread**-luokasta.

Vaikka Win32 API sisältää **_beginthreadex**-funktion, jonka avulla säikeitä voi-daan käynnistää, tulisi MFC:n toimintoja hyödyntävät säikeet luoda aina **CWinThread**-luokkaa käyttäen. Tämä johtuu siitä, että **CWinThread**-luokka käyttää säiekohtaista tietovarastoa MFC-ympäristöön liittyvien tietojen tallet-tamiseen. **CWinThread**-objektit voidaan esitellä suoraan, mutta monissa tapauk-sissa globaalin MFC-funktion **AfxBeginThread()** annetaan huolehtia **CWinThread**-objektin luomisesta.

Funktiota **CWinThread::CreateThread()** käytetään uuden säikeen käynnis-tämiseen. **CWinThread**-luokka sisältää myös funktiot **SuspendThread()** ja **ResumeThread()**, joiden avulla säikeen suorittaminen voidaan väliaikaisesti keskeyttää ja jatkaa suorittamista.

Työsäikeet ja käyttöliittymäsäikeet

MFC jakaa säikeet kahteen tyyppiin: *työsäikeisiin* (worker thread) ja *käyttöliit-tymäsäikeisiin* (user interface). Tämän jaon tekee vain MFC; Win32 API ei erottele säikeitä.

Työsäikeitä käytetään yleensä sellaisten taustatoimintojen suorittamiseen, jotka eivät vaadi käyttäjää syöttämään tietoja. Tällaisia toimintoja ovat esimerkiksi tietokannan varmistaminen ja verkkoyhteyden tilan seuraaminen.

Käyttöliittymäsäikeet voivat käsitellä käyttäjän syöttämiä tietoja ja ne toteuttavat sanomasilmukan, joka vastaa käyttäjän sovellusta käyttäessään aiheuttamiin sa-

nomiin. Paras esimerkki käyttöliittymäsäikeestä on sovelluksen pääsäie, jota edustaa sovelluksen **CWinApp**-luokasta periytetty luokka. Toissijaisia käyttöliittymäsäikeitä voidaan käyttää, kun halutaan antaa mahdollisuus sovelluksen toiminnan käyttämiseen ilma, että muiden ominaisuuksien toimintakyky laskee. Ajatellaan esimerkiksi sovellusta, jonka avulla nukutuslääkäri seuraa potilaan tilaa leikkauksen aikana. Käyttöliittymäsäikeen avulla nukutuslääkäri voi syöttää määrättyjen lääkkeiden annostusta koskevia yksityiskohtia ilman, että häiritsee säikeitä, jotka vastaavat potilaan elintoimintojen seuraamisesta.

Toissijainen säie luodaan MFC-sovelluksessa globaalia **AfxBeginThread()**-funktiota kutsumalla. **AfxBeginThread()**-funktiosta on olemassa kaksi ylikuormitettua versiota, toinen työsäikeiden luomiseen ja toinen käyttöliittymäsäikeiden luomista varten. Seuraavassa osassa kerrotaan, kuinka näitä versioita käytetään.

Työsäikeen luominen

Työsäie luodaan yksinkertaisesti tekemällä kontrolloiva funktio, joka huolehtii tehtävistä, joita säikeen tulee tehdä ja välittämällä kontrolloivan funktion osoitin **AfxBeginThread()**-funktion sopivalle versiolle.

Kontrolloivan funktion syntaksin tulee olla seuraava:

```
UINT MyControllingFunction(LPVOID pParam);
```

Parametri on 32-bittinen arvo. Parametria voidaan käyttää useilla eri tavoilla tai se voidaan jättää huomiotta. Sen avulla funktiolle voidaan välittää yksi arvo tai osoi-tin rakenteeseen, joka sisältää useita parametrejä. Jos parametri sisältää osoitti-men struktuuriin, sitä voidaan käyttää arvojen välittämiseen sekä kutsujalta säi- keelle että myös säikeeltä kutsujalle.

AfxBeginThread()-funktion työsäieversio on esitelty seuraavasti:

```
CWinThread* AfxBeginThread(AFX_THREADPROC pfnThreadProc,
    LPVOID pParam,
    int nPriority = THREAD_PRIORITY_NORMAL,
    UINT nStackSize = 0,
    DWORD dwCreateFlags = 0,
    LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL);
```

Kaksi ensimmäistä parametria ovat osoittimia, joista ensimmäinen sisältää kontrolloivan funktion osoitteen ja sille välitettävän parametrin osoitteen. Loput parametrit (joilla kaikilla on oletusarvot) antavat mahdollisuuden määrittää säikeen prioriteetin, pinon koon ja sen, luodaanko säie odotustilaan vai käynnistetäänkö se heti. Viimeisen parametrin avulla voidaan määritellä säikeen suojausattribuutit — oletusarvo NULL tarkoittaa, että säie perii kutsuvan säikeen attribuutit.

AfxBeginThread() luo uuden **CWinThread**-objektin, käynnistää sen suorituksen kutsumalla säikeen **CreateThread()**-funktiota, ja palauttaa säikeen osoittimen. Koko prosessin ajan suoritetaan tarkistuksia, joiden avulla huolehditaan siitä, että mikään prosessin vaihe ei epäonnistu. Säie lopetetaan kutsumalla säikeessä globaalia funktiota **AfxEndThread()** tai yksinkertaisesti palaamalla työsäikeen kontrolloivasta funktiosta. Kontrolloivan funktion paluuarvoa käytetään usein ilmaisemaan säikeen lopettamisen syytä. Perinteisesti paluuarvo on 0, jos funktion tehtävät saatiin suoritettua onnistuneesti. Nollasta poikkeavilla arvoilla ilmaistaan tietyntyyppisiä virheitä.

Työsäikeen luominen

Seuraavassa harjoituksessa nähdään, kuinka MyApp-sovellukseen tehdään työsäie. Harjoituksessa luodaan yksinkertainen ajastinfunktio, joka avaa viestiruudun järjestelmän kellon saavuttaessa ajastimeen asetetun ajan. Käyttäjä asettaa ajastimen käytämällä dialogia. Kun ajastin on asetettu, työsäie seuraa järjestelmän kelloa sekunneittain. Kun ajastimessa asetettu aika saavutetaan, säie avaa sanomaruudun ja päättää suorituksensa.

Ajastin kapseloidaan luotavaan **CTimer**-luokkaan. Tämä luokka sisältää ajastimen arvon tallentamiseen tarvittavan suojatun MFC:n **CTime** muuttujan. Se sisältää myös julkisen **CWinThread**-osoittimen, jonka avulla voidaan määrittää, viittaako aktiivinen säie **CTimer**-objektiin.

► **CTimer-luokan luominen**

1. Avaa CMyApp-projekti. Kaksoisnapauta sitten FileView:issä **MainFrm.cpp**-tiedoston kuvaketta.
2. Lisää seuraava koodi tiedoston alkuun #include-lauseiden jälkeen:

```
class Ctimer
{
protected:
    CTime m_time;

public:
    CWinThread * m_thread;

    CTimer() {Reset();}

    CTime GetTime() {return m_time;}
    void  SetTime(CTime time) {m_time = time;}

    void  Reset() {
        m_time = CTime::GetCurrentTime();
        m_thread = NULL;
```

```
    }
};
```

GetCurrentTime()-funktio on **CTime**-luokan staattinen jäsen, joka palauttaa järjestelmän ajan **CTime**-muodossa.

3. Määrittele globaali **CTimer**-objekti lisäämällä seuraava rivi suoraan luokan määrittelyn alapuolelle:

```
CTimer g_timer;
```

Seuraavaksi työsäikeelle tehdään kontrolloiva funktio. Tämä funktio vertaa järjestelmän aikaa ajastimeen asetettuun aikaan kerran sekunnissa. Kun ajastimessa määrätty aika saavutetaan, funktio avaa sanomaruudun ja resetoi ajastimen.

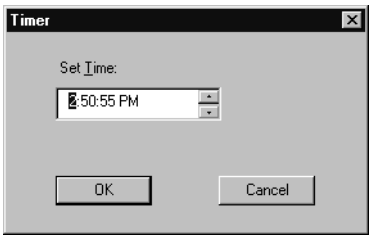
► **DoTimer()-funktion lisääminen**

1. Lisää seuraava funktion määrittely MainFrm.cpp-tiedostoon suoraan luokan määrittelyn alle:

```
UINT DoTimer(LPVOID pparam)
{
    CTime currenttime = CTime::GetCurrentTime();

    while(currenttime < g_timer.GetTime())
    {
        Sleep(1000);
        currenttime = CTime::GetCurrentTime();
    }

    AfxMessageBox("Time's up!");

    g_timer.Reset();

    return 0;
}
```

Sleep()-funktio pysäyttää säikeen, jossa funktiota kutsutaan millisekunteina määritellyksi ajaksi.

2. Seuraavaksi määritellään dialogi, jonka avulla asetetaan ajastimen aika. Tee dialogieditorilla kuvan 5.8 mukainen dialogi.



Kuva 5.8 **Timer**-dialogi

Koko dialogin tunnisteen tulisi olla **IDD_TIMER**. Yhdistelmäruudun näköinen kontrolli on **Date Time Picker** -kontrolli, joka on valittavissa kontrollityö-kaluriviltä. Tälle kontrollille tulisi antaa tunniste **IDC_DTPSETTIME** ja se tulisi asettaa näyttämään vain aikaa.

► **Asetetaan IDC_DTPSETTIME-kontrolli näyttämään vain aikaa**

1. Valitse dialogieditorissa **IDC_DTPSETTIME**-kontrolli. Aloita kontrollin ominaisuuksien editoiminen painamalla ENTER.
2. Napauta **Styles**-välilehteä. Valitse **Format**-ruudusta **Time**.
3. Sulje ominaisuusikkuna napauttamalla sen ulkopuolelle.

Seuraavaksi luodaan **Timer**-dialogin dialogiluokka.

► **CTimerDialog-luokan luominen**

1. Avaa ClassWizard painamalla CTRL+W dialogieditorin ollessa auki. Luo pyydettäessä **CTimerDialog**-luokka.
2. Valitse **Member Variables** -välilehti. Luo Value-jäsenmuuttuja tunnisteelle **IDC_DTPSETTIME**. Sen tulisi olla tyypiltään CTime ja sen nimen pitäisi olla **m_settime**.
3. Sulje ClassWizard painamalla **OK**.

Seuraavaksi täytyy lisätä komento ja käsittelijäfunktio, jotka huolehtivat ajastimen ajan asettamisesta ja ajastinsäikeen käynnistämisestä.

► **Timer-toiminnon lisääminen View-valikkoon ja OnViewTimer()-käsittelijän tekeminen**

1. Luo menueditoria käyttämällä **Timer**-komento **View**-valikkoon. Hyväksy oletuksena annettu tunniste **ID_VIEW_TIMER**.
2. Lisää ClassWizardin **Message Maps** -välilehteä käyttämällä käsittelijä **ID_VIEW_TIMER**-tunnisteelle **CMainFrame**-luokkaan. Anna funktiolle nimeksi **OnViewTimer()**.

3. Kun käsittelijä on lisätty, etsi **CMyAppApp::OnViewTimer()**-funktion toteutus napauttamalla **Edit Code** -painiketta. Lisää seuraava koodi funktion runkoon:

```
CTimerDialog aTDlg;

aTDlg.m_time = g_timer.GetTime();

if(aTDlg.DoModal() == IDOK)
{
    g_timer.SetTime(aTDlg.m_time);

    // Only one timer running per instance
    if(!g_timer.m_thread)
        g_timer.m_thread = AfxBeginThread(DoTimer, 0);
}
```

4. Lisää seuraava rivi MainFrm.cpp-tiedoston alkuun muiden #include-lauseiden joukkoon:

```
#include "TimerDialog.h"
```

5. Käännä ja käynnistä nyt MyApp-sovellus. Kokeile ajastimen toimivuutta valit-semalla **Timer**-komento **View**-valikosta. **Timer**-dialogi avautuu ja näyttää ny-kyisen kellonajan Date Time Picker -kontrollissa. Aseta kontrollia käyttämällä ajastimen aika minuutin tai kahden päähän. Sulje **Timer**-dialogi ja käynnistä ajastin napauttamalla **Start**.

Samalla kun odotat ajastimen toimintaa, kokeile muita MyApp-sovelluksen käyttö-liittymän toimintoja. Huomaat, että itsenäisesti toimiva ajastinsäie ei vaikuta mui-hin toimintoihin, vaikka tarkistaakin järjestelmän kellon kerran sekunnissa.

Kun järjestelmän kello saavuttaa **Timer**-dialogissa asetetun ajan, sanomaruutu avautuu ja työsäikeen suoritus päättyy.

Käyttöliittymäsäikeiden luominen

Kuten aiemmin mainittiin, kaikkia MFC-sovelluksen säikeitä edustavat **CWinThread**-luokan objektit, jotka on luonut **AfxBeginThread()**-funktio. Työ-säiettä luotaessa **AfxBeginThread()** luo tavallisen **CWinThread**-objektin, ja si-joittaa sen kontrollifunktion osoitteen CWinThread::m_pfnThreadProc-jäsen-muuttujaan. Kun luodaan käyttöliittymäsäie, täytyy **CWinThread**-luokasta pe-riyttää oma versio ja välittää luokan suorituksenaikaiset tiedot käyttöliittymän **AfxBeginThread()**-funktiolle.

Seuraava MFC:n lähdekoodista otettu katkelma näyttää, kuinka sovelluskehys erottelee työsäikeet ja käyttöliittymäsäikeet toisistaan. Koodi on otettu funktiosta **_AfxThreadEntry()**, joka on kaikkien MFC-säikeiden aloituspiste.

```
// First — check for simple worker thread
DWORD nResult = 0;
if (pThread->m_pfnThreadProc != NULL)
{
    nResult = (*pThread->m_pfnThreadProc)(pThread->m_pThreadParams);
    ASSERT_VALID(pThread);
}
// Else — check for thread with message loop
else if (!pThread->InitInstance())
{
    ASSERT_VALID(pThread);
    nResult = pThread->ExitInstance();
}
else
{
    // Will stop after PostQuitMessage called
    ASSERT_VALID(pThread);
    nResult = pThread->Run();
}
// Clean up and shut down the thread
threadWnd.Detach();
AfxEndThread(nResult);
```

Jos m_pfnThreadProc osoittaa kontrollifunktioon, koodi tietää, että se käsittelee työsäiettä. Kontrolloivaa funktiota kutsutaan ja säie lopetetaan. Jos m_pfnThreadProc on NULL, funktio olettaa käsittelevänsä käyttöliittymäsäiettä. Säieobjektin **InitInstance()**-funktiota kutsumalla suoritetaan säikeen alustus — luodaan esimerkiksi pääikkuna ja muita käyttöliittymäobjekteja. Jos **InitInstance()**:n funktion suoritus onnistuu (eli se palauttaa arvon TRUE), kutsutaan **Run()**-funktiota. **CWinThread::Run()** toteuttaa säikeen pääikkunalle osoitettujen sanomien prosessoinnissa tarvittavan sanomasilmukan.

InitInstance() ja **Run()** -funktioiden pitäisi olla sinulle jo valmiiksi tuttuja. Ne ovat kaksi tärkeintä **CWinApp**-virtuaalifunktiota, jotka kuvattiin luvun 3 oppitunnilla 3, Win32-sovellusarkkitehtuurin MFC-toteutuksen yhteydessä.

Ohjelmoijan kuuluu tehdä säieluokkaan ylikuormitettu versio **InitInstance()**-funktiosta. **CWinThread::ExitInstance()**-funktion ylikuormitettuun versioon sijoitetaan usein luokan omia siivoustoimintoja. **Run()**-funktiosta käytetään yleensä kantaluokan versiota.

Seuraavassa harjoituksessa nähdään yksinkertaisin tapa, jolla **CWinThread**-luokasta voidaan luoda periytetty luokka.

► **CMyUIThread-luokan luominen**

1. Avaa ClassWizard CMyApp-projektissa painamalla CTRL+W.
2. Napauta **Add Class** -painiketta ja valitse alasvetovalikosta **New**. **New Class** -dialogi avautuu.

3. Kirjoita **Name**-ruutuun **CMyUIThread**. Valitse **CWinThread**-vaihtoehto **Base class** -valikosta.
4. Luo luokka napauttamalla **OK**, ja sulje ClassWizard napauttamalla uudelleen **OK**.

Aloita säieluokan koodiin tutustuminen avaamalla MyUIThread.h ja MyUIThread.cpp tiedostot. Huomaa, että ClassWizard on luonut tynkäfunktiot **InitInstance()** ja **ExitInstance()**, joihin sinun tulee sijoittaa luokkakohtaiset tehtävät hoitava koodi. Huomaa myös, että säieluokkaan on tehty sanomakartta. **DECLARE_DYNCREATE** ja **IMPLEMENT_DYNCREATE** -makrot ovat tässä tärkeässä osassa, koska ne toteuttavat käyttöliittymäsäikeen **AfxBeginThread()**-funktion vaatimat suorituksenaikaiset tiedot **CObject**-luokasta periytetyistä luokista.

Kun säieluokan toteutus on valmis, kutsutaan käyttöliittymäsäikeitä varten tehtyä versiota **AfxBeginThread()**-funktiosta ja ja välitetään sille säieluokan suorituk-senaikaiset tiedot osoittimen avulla:

```
AfxBeginThread(RUNTIME_CLASS(CMyUIThread));
```

RUNTIME_CLASS makro palauttaa osoittiminen **CRuntimeClass**-struktuuriin, jossa säilytetään kaikkien **CObject**-luokasta periytettyjen **DECLARE_ DYNAMIC, DECLARE_DYNCREATE**, tai **DECLARE_SERIAL** makrojen avulla esiteltyjen luokkien suorituksenaikaisia tietoja. Näin **AfxBeginThread()** osaa luoda oikean tyyppisen säieobjektin.

Käyttöliittymäsäikeiden **AfxBeginThread()**-versiolla on samat oletusparametrit kuin työsäieversiollakin. Se palauttaa luomansa objektin osoitteen.

Säikeiden synkronointi

Toissijaiset säikeet suorittavat usein *asynkronisia* operaatioita. Asynkroninen ope-raatio tarkoittaa toimintoa, joka suoritetaan itsenäisesti muista operaatioista ja tapahtumista riippumatta. Jos ajatellaan ajastinsäiettä niin huomataan, että se toimii itsenäisesti omana suorituspolkunaan tarkastaessaan järjestelmän kellonaikaa. Sen ei tarvitse odottaa sovelluksen päsäikeen tapahtumia ja sovellus voi edetä suorituksessaan tarvitsematta odottaa, että säie saa työnsä valmiiksi.

On monia tilanteita, joissa näitä asynkronisia toimintoja täytyy *synkronoida* tai koordinoida niiden toimintoja. Ajatellaan esimerkiksi tulostuksen aikataulutusta hoitavaa säiettä, joka asettaa eri sovellusten tulostustöitä tulostusjonoon. Tulos-tussäikeiden täytyy ilmoittaa aikataulusäikeelle, että ne haluavat liittyä jonoon ja aikataulusäikeen täytyy ilmoittaa tulostussäikeelle, kun se voi aloittaa tulostamisen.

Toinen tapaus, jossa tyypillisesti tarvitaan synkronointia, on sovelluksen globaalien tietojen muuttaminen. Ajatellaanpa valvontasovellusta, joka kirjoittaa sensorilta saamansa mittaustulokset tiettyyn tietorakenteeseen. Toinen säie lukee tiedot tästä rakenteesta ja esittää ne näytöllä. Kuvitellaanpa tilanne, jossa tietojen näyttä-misestä huolehtiva säie lukee tietoja juuri samalla hetkellä kuin sensoria lukeva säie on kirjoittamassa uusia tietoja. Tilanteen seurauksena tiedot todennäköisesti sekoittuvat ja seuraukset saattavat olla vakavia, jos tarkkailtavana on esimerkiksi ydinvoimala. Säikeet, jotka pääsevät käsiksi globaaleihin tietoihin, täytyy synkronoida niin, että vain yksi säie kerallaan lukee tai kirjoittaa tietoja.

Yksi tapa synkronoinnin toteuttamiseen on globaalin objektin käyttäminen säikei-den välisenä välittäjänä. MFC sisältää useita synkronointiluokkia. Ne on lueteltu taulukossa 5.4. Näitä synkronointiluokkia, jotka on periytetty kantaluokasta **CSyncObject**, voidaan käyttää kaikenlaisten asynkronisten tapahtumien koordi-nointiin.

Taulukko 5.4 MFC:n synkronointiluokat

Nimi	Kuvaus
CCriticalSection	Synkronointiluokka, joka sallii vain yhden nykyisen prosessin säikeistä käsitellä objektia.
CMutex	Synkronointiluokka, joka sallii vain yhden minkä tahansa prosessin säikeistä käsitellä objektia.
CSemaphore	Synkronointiluokka, joka sallii yhden tai enintään erikseen määritellyn maksimimäärän säikeitä käsitellä objektia samanaikaisesti.
CEvent	Synkronointiluokka, joka ilmoittaa sovellukselle, kun tapahtuma on syntynyt.

Synkronointiluokkia käytetään yhdessä synkronoinnin saantiluokkien **CSingleLock** ja **CMultiLock** kanssa varmistamaan, että globaalien tietojen ja resurssien käsittely on säieturvallista. Näitä luokkia suositellaan käyttämään seuraavasti:

- Kääri globaalien tieto-objektien ja resurssien käsittelyfunktiot luokan sisälle. Suojaa käsiteltävät tiedot ja säätele niihin pääsyä julkisten funktioiden avulla.
- Luo luokan sisällä sopivan tyyppinen synkronointiobjekti. Jos haluat esimerkiksi vain yhden säikeen kerrallaan käyttävän tietoja, käytät **CCriticalSection**-objektia tai jos haluat, että resurssi ilmoittaa valmiudestaan ottaa tietoja vastaan, käytät **CEvent**-objektia.
- Luo synkronoinnin saantiobjektin ilmentymä jäsenfunktiossa, jonka kautta tietoa tai resurssia käsitellään. Käytä **CSingleLock**-objektia, kun odotetaan vain yhtä objektia kerrallaan. Käytä **CMultiLock**-objektia, kun useita objekteja saatetaan käyttää samanaikaisesti.

- Kutsu synkronoinnin saantiobjektin **Lock()**-jäsenfunktiota ennen kuin funktiossa yritetään käsitellä suojattuja tietoja. **Lock()**-funktio voidaan asettaa odottamaan määrätty aika (tai ikuisesti) siihen liittyvän synkronointiobjektin tulemista saapuville. **CCriticalSection**-objekti on esimerkiksi saatavilla, jos se onnistuu saamaan yksinoikeudella turvallisen käyttöoikeuden nykyiselle säi-keelle. Tapahtuman saatavuus asetetaan ohjelmakoodissa kutsumalla funktiota **CEvent::SetEvent()**.
- Kun suojattujen tietojen käsittely on saatu tehtyä, kutsu saantiobjektin **Unlock()**-funktiota tai salli objektin tuhoutuminen.

Globaalien resurssien ja synkronointikoodin kapseloiminen säieturvalliseen luok-kaan auttaa keskittämään ja kontrolloimaan yhteyttä resursseihin ja suojaa *luk-kiuma* (deadlock) -tilanteelta. Lukkiuma syntyy, kun kaksi tai useampia säikeitä odottaa yhden säikeen vapauttavan jaetun resurssin ennen kuin jatkavat suori-tustaan. Lukkiumatilanteet ovat tunnetusti vaikeasti toistettavia ja jäljitettäviä, joten on ehdottoman tärkeää, että monisäikeiset sovellukset tutkitaan mahdol-listen lukkiumatilanteiden varalta ja suoritetaan niiden estämiseksi tarvittavat toi-menpiteet.

Seuraavassa harjoituksessa havannollistetaan tapaa, jolla globaalia tietoa voidaan käsitellä säieturvallisesti. **CCriticalSection**-objektia ja **CSingleLock**-objektia käyttäen suojataan pääsy edellisessä harjoituksessa tehdyn **CTimer**-luokan **CTime**-objektiin.

► CTimer-luokan käsittely säieturvallisesti

1. Lisää seuraava rivi MainFrm.cpp-tiedoston alkuun muiden #include-lauseiden joukkoon:

```
#include <afxmt.h>
```

2. Lisää seuraava rivi **CTimer**-luokan määrittelyn **protected**-osaan:

```
CCriticalSection m_CS;
```

3. Poista **CTimer::GetTime()** ja **CTimer::SetTime()** funktioiden inline-määrittelyt (muista lisätä puolipisteet niiden paikoille).
4. Lisää **CTimer**-luokan määrittelyn alle seuraava **CTimer::GetTime()**-funktion määrittely:

```
CTime CTimer::GetTime()
{
    CSingleLock csl(&m_CS);
    csl.Lock();
    CTime time = m_time;
    csl.Unlock();
    return time;
}
```

5. Lisää seuraava **CTimer::SetTime()**-funktion määrittely vaiheessa 4 lisäämiesi rivien alle:

```
void CTimer::SetTime(CTime time)
{
    CSingleLock csl(&m_CS);
    csl.Lock();
    m_time = time;
    csl.Unlock();
}
```

CSingleLock::Unlock()-kutsut näissä funktioissa eivät tarkkaan ottaen ole täysin välttämättömiä — ne vain kuuluvat meidän mielestämme hyvään ohjelmointitapaan.

Voit nyt kääntää ja käynnistää sovelluksen. Et huomaa sovelluksen toiminnassa mitään eroa aikaisempaan, mutta nyt voit olla varma, että **CTimer**-luokan ilmentymiin varastoidut tiedot ovat turvallisesti useiden säikeiden käytettävissä.

Oppitunnin yhteenveto

Monisäikeisyyttä voidaan hyödyntää sovelluksissa aina, kun niiden suorituskykyä voidaan parantaa suorittamalla useita erillisiä tehtäviä rinnakkain. Asynkroniset operaatiot, jotka toimivat omassa aikakehyksessään, ajastimen aikatauluttamat tehtävät ja operaatiot, jotka eivät tarvitse käynnistyäkseen toiselta säikeeltä tule-vaa tapahtumaa, ovat potentiaalisia ehdokkaita sovelluksen prosessin toissijaisiksi säikeiksi.

Kaikki MFC-toimintoja käyttävät säikeet tulisi luoda MFC:n **CWinThread**-luokan ilmentymien avulla. MFC sisältää globaalin funktion **AfxBeginThread()**, jo-ka auttaa ohjelmoijaa säikeiden luomisesssa MFC-sovellukseen. MFC jakaa säi- keet kahteen eri tyyppiin. Työsäikeitä käytetään yleensä sellaisten taustatehtävien tekemiseen, jotka eivät vaadi käyttäjän toimenpiteitä. Käyttöliittymäsäikeet käsit- televät käyttäjän syötteitä ja ne toteuttavat sanomasilmukan, jonka avulla vasta-taan käyttäjän ohjelmaa käyttäessään aiheuttamiin sanomiin.

Työsäie luodaan tekemällä ensin kontrolloiva funktio, joka suorittaa säikeen tehtäväksi määrätyt tehtävät ja välittämällä sitten kontrolloivan funktion osoite **AfxBeginThread()**-funktion työsäieversiolle. Käyttöliittymäsäiettä luotaessa periytetään ensin oma luokka **CWinThread**-luokasta, ylikuormitetaan tarvittavat jäsenfunktiot ja välitetään suorituksenaikaiset tiedot luokasta **AfxBeginThread()** -funktion käyttöliittymäversiolle.

Toissijaiset säikeet suorittavat yleensä *asynkronisia* toimintoja. On kuitenkin useita tilanteita, joissa niiden toimintaa täytyy synkronoida. Ne voivat joutua odottamaan signaalia toiselta säikeeltä tai niiden täytyy lähettää signaali toiselle

säikeelle. Säikeiden, jotka käsittelevät globaalia tietoa, täytyy toimia synkronoidusti niin, että ne eivät yritä käsitellä tietoja samanaikaisesti.

MFC sisältää joukon kantaluokasta **CSyncObject** periytettyjä synkronointiluokkia, joita käytetään asynkronisten säikeiden koordinointiin. Näitä luokkia käytetään yhdessä synkronoinnin saantiobjektien **CSingleLock** ja **CMultiLock** kanssa.

Kun säikeen pääsyä jaettuun resurssiin ohjataan, on suositeltavaa kapseloida resurssit luokan sisälle ja toteuttaa synkronointiobjektit saman luokan suojattuina jäseninä. Käytä synkronoinnin saantifunktioita luokan jäsenfunktioissa varmistamaan, että säikeet käsittelevät resursseja turvallisella tavalla — eli vain yksi säie käsittelee tietoja kerralla.

Oppitunti 4: Tilanteenmukainen ohje

Windows-sovellukset sisältävät usein tilanteenmukaisen ohjeen (context-sensitive Help), jonka kautta käyttäjä saa ohjeita sovelluksen tietystä toiminnosta, kuten dialogeista, komennoista ja työkalurivin painikkeista.

Tällä hetkellä Windowsissa on ohjeen tekemiseen kaksi tapaa: perinteinen Windowsohjejärjestelmä WinHelp, joka perustuu RTF (rich text format) -dokumentteihin ja uudempi HTML-ohjejärjestelmä, joka perustuu käännettyihin HTML-dokumentteihin.

Tällä oppitunnilla opit, kuinka voit tehdä tilanteenmukaisen ohjeen sovellukseesi MFC AppWizardin sisältämiä toimintoja käyttäen. Opit myös HTML-ohjeen tekemisen perusvaiheet.

Tämän oppitunnin jälkeen:

- Tiedät, kuinka Windows-sovellukseen tehdään tilanteenmukainen ohje.
- Tiedät, kuinka tilanteenmukainen ohje lisätään MFC AppWizard -sovellukseen.
- Tiedät WinHelpiä käytettäessä tarvittavat komponentit.
- Tiedät, kuinka HTML-pohjainen ohje lisätään sovellukseen.

Oppitunnin arvioitu kesto: 40 minuuttia

WinHelp

Sovellukset, jotka käyttävät Windows Help -järjestelmää, lataavat järjestelmäkansiossa sijaitsevan WinHelp-sovelluksen (Winhlp32.exe) käyttäjän pyytäessä ohjetta.

WinHelp näyttää sovelluksen ohjetiedostossa olevia sivuja. Sovelluksen ohjetiedosto sijaitsee samassa kansiossa sovelluksen exe-tiedoston kanssa ja sen nimi on .hlp-tarkenninta lukuun ottamatta sama kuin ohjelmatiedostolla. Näytettävä sivu määritellään sovelluksen WinHelpille välittämän help context -parametrin avulla.

Käyttäjä voi avata ohjeen useilla tavoilla:

- **Painamalla F1-näppäintä** Käyttäjä voi painaa F1-näppäintä aktiivisessa ikkunassa, dialogissa, viesti-ikkunassa tai valittuaan työkalurivin, komennon tai työkalurivin painikkeen, käynnistääkseen valittuun kohteeseen liittyvän ohjeen. Komennoissa ohje liitetään valittuna olevaan kohteeseen.

  Huomaa, että voit itse määritellä näppäimen, joka käynnistää ohjeen. F1 on kuitenkin yleisesti käytetty standardi.
- **Siirtymällä ohjetilaan** Käyttäjä voi siirtyä aktiivisen sovelluksen “ohjetilaan” painamalla SHIFT+F1 tai napauttamalla **Lisätietoja**-painiketta.

  Ohjetilassa hiiriosoitin muuttuu nuolen ja kysymysmerkin yhdistelmäksi. Kun sovellus on tässä tilassa, käyttäjä saa kohteeseen liittyvän ohjeen napauttamalla mitä tahansa ikkunaa, dialogia, sanomaikkunaa, komentoa tai työkalupalkin painiketta. Ohjetila sulkeutuu, kun ohje on näytetty. Ohjetila sulkeutuu myös painamalla ESC tai siirtymällä pois sovelluksesta ja takaisin sovellukseen.
- **Käyttämällä Ohje-valikkoa** Useimmissa sovelluksissa käyttäjälle on tarjolla yksi tai useampi ohjekomento. Useimmissa Windows-sovelluksissa on esimerkiksi komento, josta avautuu sovelluksen ohjetiedosto. Ohje-valikossa voi lisäksi olla komentoja, jotka avaavat **Etsi**-dialogin tai tarjoavat linkin lisätietoja sisältävälle Web-sivulle.

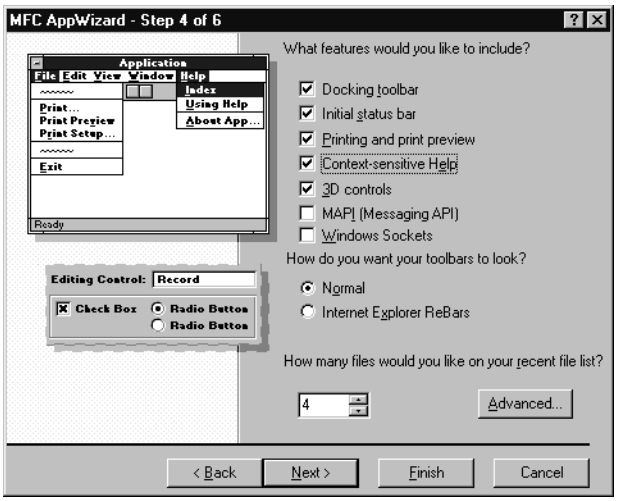
AppWizardin tuki tilanteenmukaiselle ohjeelle

MFC:n sovellusrunko sisältää laajan tuen WinHelpille. Kun teet context-sensitive Help -valinnan, MFC AppWizard asentaa projektiisi kaiken WinHelpin luomiseen tarvittavan niin, että selviät ohjeen tekemisestä mahdollisimman pienellä vaivalla.

Seuraavassa harjoituksessa näet, kuinka AppWizardia käyttäen luodaan projekti, jossa on tuki tilanteenmukaiselle ohjeelle ja kuinka sovelluksen ohje saadaan näkyviin.

► **MyHelpApp-projektin aloittaminen**

1. Valitse **New**-komento **File**-valikosta ja tee uusi MFC AppWizard (exe) -projekti **MyHelpApp**.
2. Valitse MFC AppWizardin vaiheessa 1 **Single document**.
3. Vaiheissa 2 ja 3 valitaan oletusvaihtoehdot. Valitse vaiheessa 4 **Context-sensitive Help** -valintaruutu kuten kuvassa 5.9.



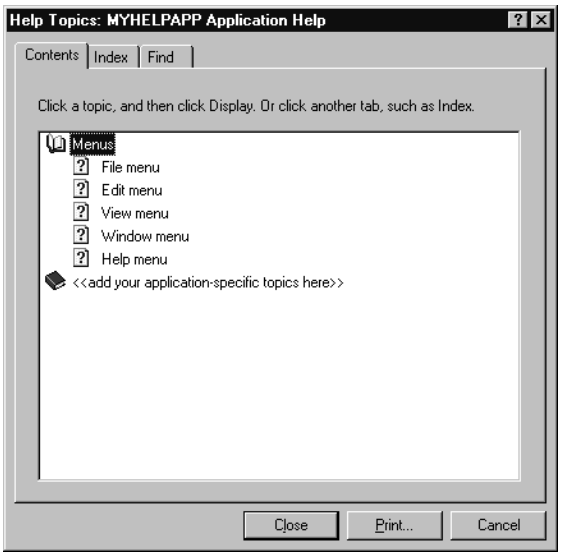
Kuva 5.9 AppWizardin context-sensitive Help -valinta

4. Valitse oletusarvot lopuissa AppWizardin vaiheissa ja luo MyHelpApp-projekti.

Seuraavassa harjoituksessa tutkitaan sovellukseesi luotua oletusohjetta.

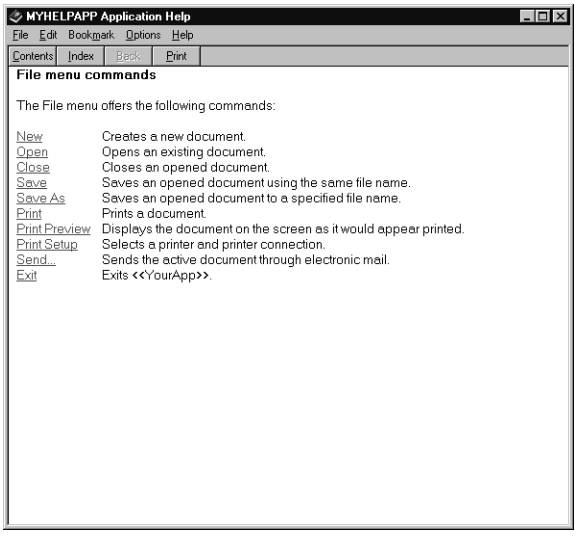
► **MyHelpApp-ohjeen tutkiminen**

1. Käännä MyHelpApp-sovellus painamalla F7.
2. Käynnistä sovellus painamalla CTRL+F5.
3. Valitse **Help**-valikosta **Help Topics**. MyHelpApp-sovelluksen ohje avautuu.
4. Napauta **Contents**-välilehteä. Kaksoisnapauta kohtaa **Menus** niin, että Help-ikkuna näyttää samalta kuin kuvassa 5.10.



Kuva 5.10 MyHelpApp-sovelluksen ohje

5. Valitse ohjeaihe **File menu**. **Close**-painike muuttuu **Display**-painikkeeksi.
6. Napauta **Display**-painiketta ja pääset tutkimaan **File**-valikon ohjetta. **File**-valikon ohje näyttää samanlaiselta kuin kuvassa 5.11.



Kuva 5.11 **File**-valikon ohjesivu

Käytä hetki Ohje-järjestelmän tutkimiseen. Valitse kokeeksi jokin hyperlinkeistä ja kokeile työkalurivin ja valikoiden komentoja. Sulje Ohje, kun olet valmis.

► **Tilanteenmukaisen ohjeen avaaminen**

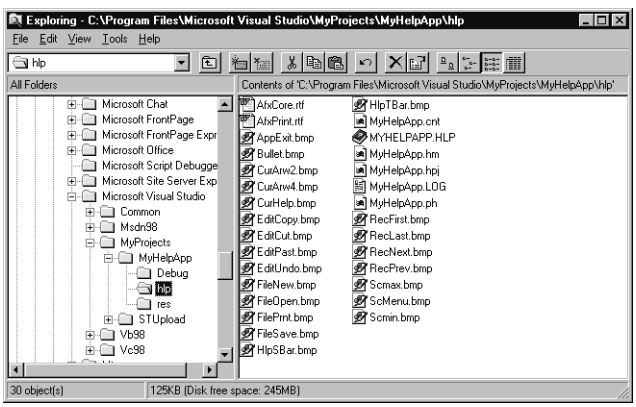
1. Avaa MyHelpApp-sovelluksen **File**-valikko napauttamalla **File**. Osoita **Print**-komentoa hiirellä, mutta älä valitse sitä.
2. Paina F1-näppäintä. Ohje avautuu ja näyttää **Print**-komennon ohjeen.
3. Sulje ohje. Sulje **File**-valikko napauttamalla sovelluksen ikkunaa valikon ulkopuolella. Siirrä osoitin **Context Help** -painikkeen päälle niin, että painike nousee ylös ja työkaluvihje tulee näkyviin. **Context Help** -painikkeessa on kuva mustasta osoittimesta ja sinisestä kysymysmerkistä ja se sijaitsee työkalurivin oikeassa reunassa.
4. Paina F1-näppäintä. Ohje avautuu ja näyttää **Context Help** -ohjeaiheen.
5. Sulje ohje.

► **Ohjetilan ohje**

1. Napauta **Context Help** -painiketta. Huomaa, kuinka hiiren osoitin muuttuu ilmaisten, että olet ohjetilassa.
2. Saat minkä tahansa komennon tai työkalurivin painikkeen ohjeen näkyviin napauttamalla sitä. Kokeile.
3. Sulje ohje.

Ohjetiedoston komponentit

Kun valitset context-sensitive Help -vaihtoehdon luodessasi sovellusta, AppWizard tekee hlp-nimisen alikansion projektisi pääkansioon. Tämä alikansio sisältää tiedostot, joita käytetään sovelluksesi ohjetiedoston luomiseen. Kuvassa 5.12 näet MyHelpApp-projektin hlp-kansion sisällön avattuna Resurssienhallintaan (projektin luomisen jälkeen).



Kuva 5.12 MyHelpApp\hlp-kansio

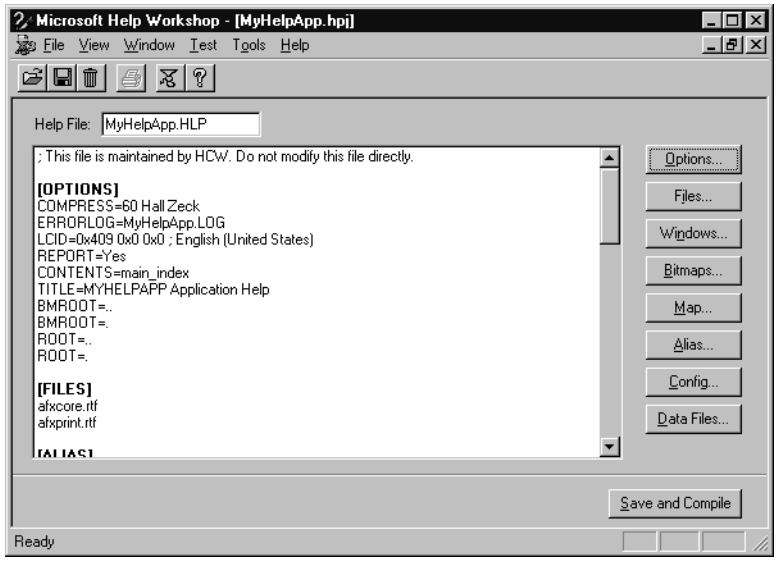
Seuraavissa jaksoissa kerrotaan, kuinka näitä tiedostoja käytetään ohjetiedostoa tehtäessä.

.hpj-tiedostot

.hpj-tiedosto on ohjeprojektitiedosto (Help project file), joka sisältää Windows-ohjekääntäjän sovelluksesi ohjeen rakentamista varten tarvitsemat tiedot. Ohjeprojektin tiedostoja hallitsee Microsoft Help Workshop, joka on sovellus, jonka avulla voit tehdä sovelluksen ohjeen visuaalisesti — .hpj ovat Help Workshopille samat kuin .dsw tiedostot ovat Visual Studiolle. Kuten Visual Studio, Help Workshop pystyy käsittelemään usean tyyppisiä tiedostoja.

Microsoft Help Workshop (hcw.exe) on asennettu \Program Files\Microsoft Visual Studio\Common\Tools -kansioon Visual Studio -perusasennuksen yhteydessä. Voit käynnistää Help Workshopin kaksoisnapauttamalla ohjeprojektitiedostoa. Seuraavan sivun kuvassa 5.13 näet MyHelpApp.hpj tiedoston avattuna Help Workshopiin.

.hpj sisältää useita lohkoja, joita voidaan muokata käyttämällä ruudun oikeassa reunassa olevia painikkeita. Ohjekääntäjän käyttämät valinnat on määritelty Options-lohkossa, jonka näet kuvassa 5.13. Muut lohkot sisältävät viittauksia muun tyyppisiin tiedostoihin, joita voidaan kääntää mukaan ohjeprojektiin.



Kuva 5.13 MyHelpApp.hpj avattuna Workshopiin

.rtf-tiedostot

.rtf-tiedosto on rich text format -muotoinen tiedosto, josta muodostuu sivu ohjetiedostoon. Ohjetiedostoon sisällytettävät .rtf-tiedostot on lueteltu projektin .hpj-tiedoston [FILES] lohkossa.

AppWizard tarjoaa joukon .rtf-tiedostoja, jotka sisältävät käyttöliittymän yleisten osien kuten **File** ja **Edit** -valikoiden ohjeaiheet. Voit muokata näissä tiedostoissa olevaa tekstiä haluamaksesi ja voit lisätä omaan sovellukseesi liittyvät ohjeaiheet. .rtf-tiedostojen muokkaamiseen tarvitset tekstinkäsittelyohjelman, kuten Micro-soft Wordin, joka osaa käsitellä rich text format -muodossa olevia tiedostoja.

.bmp-tiedostot

.rtf-tiedostot sisältävät viittauksia kuvituksena käytettäviin .bmp-tiedostoihin.

.hm-tiedostot

.hm-tiedoston luo MakeHm-työkalu. MakeHm lukee Resource.h tiedostosi ja luo ohjeen aihetunnisteet (Help context ID), jotka vastaavat sovelluksen resurssitunnisteita niin, että jokainen dialogi, komento tai muu resurssi saa oman ohjetunnisteen. Pidä mielessä, että MakeHm ei tarkista tunnisteiden ainutlaatuisuutta, joten niissä saattaa ilmetä ristiriitoja.

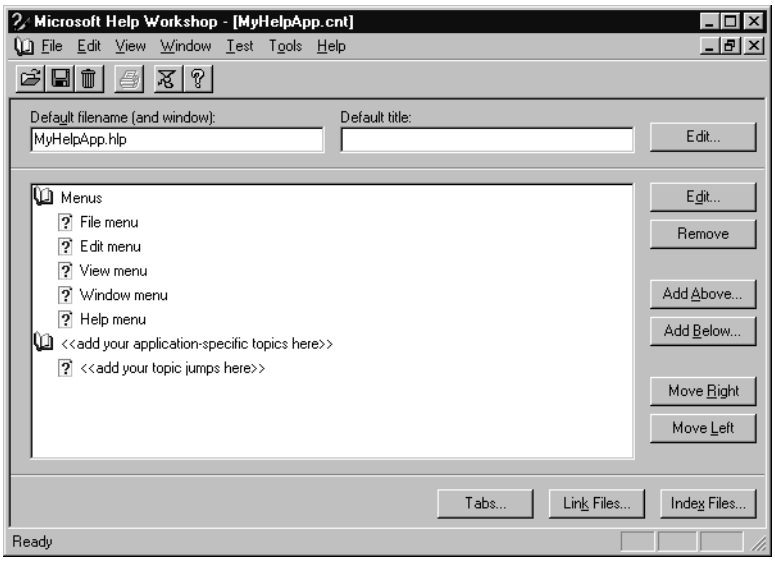
Projektin .hpj-tiedoston [MAP]-osassa on lause, jossa määritellään mukaan projektin .hm-tiedosto ja MFC:n lisäämä standardi .hm-tiedosto. Ohjekääntäjä hyödyntää näiden .hm-tiedostojen sisältämiä ohjetunnisteita määritellessään, mikä ohjeen aihe liittyy mihinkin dialogiin, komentoon tai muuhun resurssiin.

Kun lisätään uusia resurssitunnisteita, täytyy muistaa käyttää oikeita etuliitteitä, jolloin MakeHm osaa tehdä ohjaukset oikein. Visual C++:n resursseille tarjoamat oletusnimet on varustettu oikeilla etuliitteillä. Lisää tietoja saat hakemalla Visual C++:n ohjeesta fraasilla “Preferred Resource ID Prefixes”.

Aina kun lisäät uuden resurssin projektiisi, sinun täytyy lisätä .rtf-tiedostoon uusi ohjeen aihe näille resursseille. AppWizard tekee räätälöidyt käännösohjeet projektiasia varten niin, että projektitiedosto käännetään uudelleen lähdekoodin muututtua. Tämä pitää sisällään MakeHm.exe:n kutsumisen, kun Resource.h-tiedostoa on muutettu.

.cnt-tiedostot

.cnt-tiedosto sisältää ohjeen Sisältö-ruudun tarvitsemat tiedot. Voit muokata .cnt-tiedostoja Help Workshopissa, kuten kuvassa 5.14.



Kuva 5.14 MyHelpApp.cnt-tiedosto Help Workshopissa

.ph-tiedosto

.ph-tiedosto on ohjekääntäjän luoma taulukko avainfraaseista. Se luodaan, kun pakkausvaihtoehto on määritelty.

.hlp-tiedostot

Kuten aikaisemmin mainittiin, .hlp-tiedostot ovat sovelluksen ohjetiedostoja, joita Windowsin ohjekääntäjä tuottaa. Ohjekääntäjä tekee .hlp-tiedoston sovelluksesi lähdekoodikansioon. Jos käännöskansio (esimerkiksi Debug tai Release kansio) on olemassa, .hlp-tiedosto kopioidaan sinne, jotta ajantasainen versio sovelluksen ohjeesta olisi saatavilla aina kun suoritat sovelluksen.

.log-tiedostot

.log-tiedostossa ovat ohjekääntäjän tuottamat ilmoitukset. Niitä voidaan tutkia Help Workshop ohjelmalla.

Lähdekoodin osat

Sen lisäksi, että AppWizard luo sovelluksesi ohjetiedoston tekemiseen tarvittavat tiedostot, se lisää sovelluksen lähdekoodiin osat, joiden avulla tilanteenmukainen ohje toteutetaan. **Help Topics** -valinta (resurssi ID_HELP_FINDER) lisätään **Help**-valikkoon, ja **Context Help** -painike (ohjetilaohje, jonka resurssi **ID_ CONTEXT_HELP**) lisätään sovelluksen työkaluriville. F1 lisätään sovelluksen pikanäppäintaulukkoon ja se määritellään

tilanteenmukaisen ohjeen pikanäppäi-meksi (resurssi **ID_HELP**) ja SHIFT+F1 määritellään ohjetilan pikanäppäimeksi.

Nämä resurssitunnisteet ohjataan kantaluokan käsittelijöille kantaikkunan sano-makartassa, kuten seuraavasta koodista ilmenee:

```
BEGIN_MESSAGE_MAP(CMainFrame,  CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
    // Global Help commands
    ON_COMMAND(ID_HELP_FINDER, CFrameWnd::OnHelpFinder)
    ON_COMMAND(ID_HELP, CFrameWnd::OnHelp)
    ON_COMMAND(ID_CONTEXT_HELP,  CFrameWnd::OnContextHelp)
    ON_COMMAND(ID_DEFAULT_HELP,  CFrameWnd::OnHelpFinder)
END_MESSAGE_MAP()
```

Kaikki käsittelijät kutsuvat lopulta **CWnd::WinHelp()**-funktiota, joka käynnistää WinHelp sovelluksen. **OnContextHelp()** luo ohjetunnisteen (Help context ID) resurssitunnisteesta, jonka se välittää WinHelpille **WinHelp()**-funktion paramet-rinä. Jos framework ei pysty luomaan käypää ohjetunnistetta, lähetetään sovel-lukselle **ID_DEFAULT_HELP** komentosanoma.

Ohjeaiheen luominen

Seuraavissa harjoituksissa lisäät MyHelpApp-sovellukseen uuden valikon ja valik-koa sekä sen komentoja vastaavat ohjeaiheet. Opit, kuinka lisäät ohjeen asiayh-teydestä kertovan informaation niin, että tilannekohtainen ohje toimii oikein.

Näiden harjoitusten tekeminen edellyttää, että olet asentanut tekstinkäsittelyohjelman, jolla voit muokata rtf-muodossa olevaa tekstiä. Esimerkeissä käytämme Microsoft Wordiä.

► ID_IMPORT_TEXTFILE-komennon lisääminen

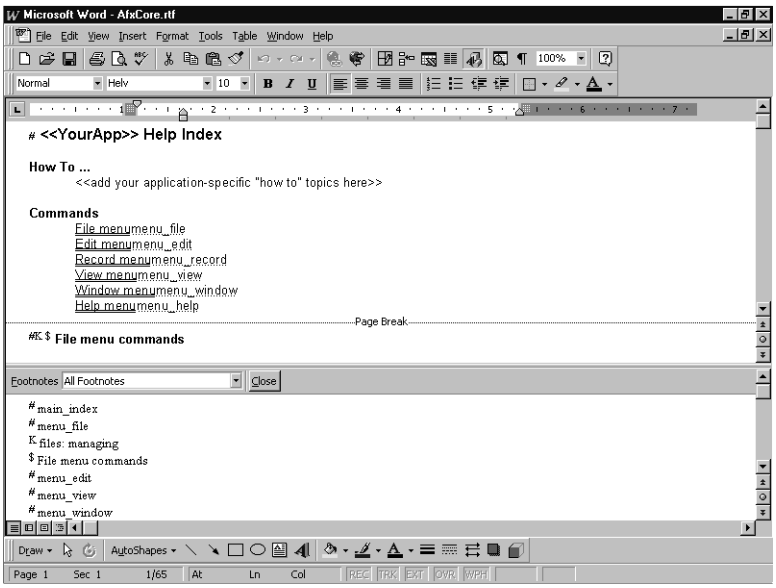
1. Avaa MyHelp-projektin ResourceViewissä **IDR_MAINFRAME** valikko menueditoriin. Lisää uusi valikko, jonka otsikossa lukee **&Import**. Sijoita tähän valikkoon **&Text File** -niminen komento. Sulje ja avaa uudelleen Menu Item Properties varmistaaksesi, että editori on luonut komennolle tunnisteen **ID_IMPORT_TEXTFILE**.
2. Käännä MyHelpApp-sovellus. Huomaa, että seuraava ilmoitus ilmestyy tulosteikkunaan:

   ```
   Making Help include file...
   Making Help file...
   ```

3. Etsi MyHelpApp.hm-tiedosto hlp-alikansiosta. Tutki tiedostoa Muistiolla ja varmista, että MakeHm-toiminto on lisännyt ohjeviitetunnisteen **HID_IMPORT_TEXTFILE**. Sulje MyHelpApp.hm-tiedosto.

Seuraavassa harjoituksessa lisäät ohjeaiheet sovelluksen .rtf-tiedostoon. Ymmärtääksesi paremmin tiedostomuotoa, avaa AfxCore.rtf-tekstieditoriisi. Jos editorissa on toiminto, jolla piilotettu teksti saadaan näkyviin, valitse se. Wordissä, tämä tehdään valitsemalla **Asetukset** (Options) **Työkalut**-valikosta (Tools) ja asettamalla valinta **Näkymä**-välilehden (View) **Piiloteksti**-ruutuun (Hidden text). Ota myös alaviitteet käyttöön — Wordissä valitsemalla **Näytä**-valikosta (View) Alaviitteet (Footnotes).

Kuvassa 5.15 näet miltä AfxCore.rtf näyttää Wordin normaalinäkymässä.



Kuva 5.15 AfxCore.rtf Microsoft Wordissä

Jokainen ohjeaihe .rtf-tiedostossa alkaa pakotetulla sivunvaihdolla. Jokainen aihe voi sisältää hypertekstilinkkejä muihin ohjeaiheisiin. Linkit merkitään kaksoisalleviivauksella ja heti niiden jäljessä on linkin kohde piilotekstiksi muotoiltuna. Linkkien kohteet merkitään ruutumerkillä (#) alaviitteiksi. Alaviite sisältää asiayhteyden nimen, joka voi olla ohjetiedoston sisäinen tai jokin ohjeprojektin .hm-tiedostossa määritellyistä aihetunnisteista. Ohje käyttää upotettuja aihetunnisteita oi keiden ohjeaiheiden löytämiseen tilannekohtaisesta ohjeesta.

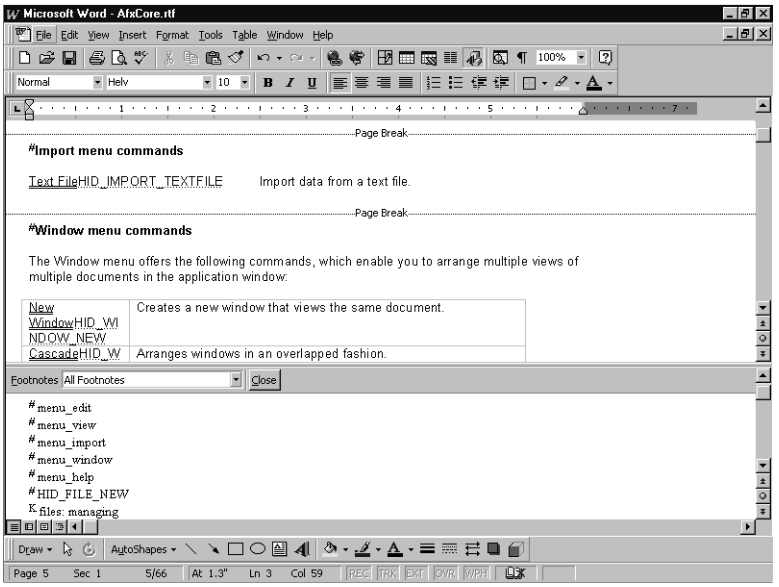
Muilla erityisillä alaviitemerkinnöillä merkitään etsittävät avainsanat (K) ja aiheiden nimet ($).

Kaksinkertaisissa kulmasuluissa oleva teksti (esimerkiksi <<YourApp>> kuvassa 5.15) on AppWizardin luoma paikanvaraaja, joka korvataan sovelluskohtaisella tekstillä.

Seuraavassa harjoituksessa teet ohjeaiheen **Import**-valikolle, jonka lisäsit MyHelpApp-sovellukseen.

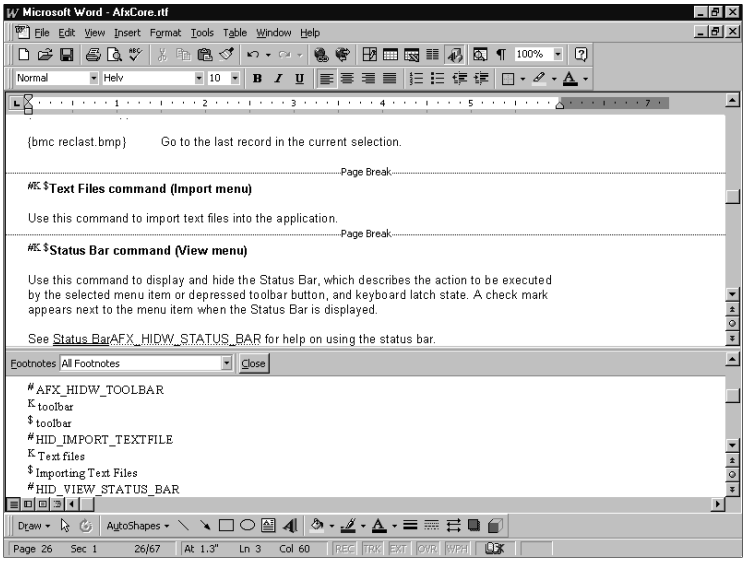
► **Import-valikko-ohjeaiheen lisääminen**

1. Korvaa AfxCore.rtf-tiedostossa olevat <<YourApp>>-tekstit tekstillä **MyHelpApp**.
2. Muuta Help Index aiheen (tiedostossa ensimmäisenä) hypertekstilinkin **Record menu** tilalle **Import menu**. (MyHelpApp sovelluksessa ei ole **Record**-valikkoa, joten sitä varten ei myöskään tarvita ohjeaihetta.)
3. Muuta linkin kohdetekstiksi (välittömästi linkkitekstin jälkeen oleva piiloteksti) **menu_import**.
4. Lisää sopivaan kohtaan uusi aihe otsikolla **Import Menu Commands**. Varmista, että aihe on omalla sivullaan. Otsikko tulee merkitä #-alaviitteellä, joka osoittaa tekstiin **menu_import**. Aihesivulla tulisi olla **Text File** -niminen hyperlinkki, jota seuraa viite kohteeseen **HID_IMPORT_TEXTFILE**. Hyperlinkin jälkeen tulisi seurata lyhyt kuvaus perustekstimuodossa kuten kuvassa 5.16.



Kuva 5.16 **Import**-valikon ohjeaihe

5. Lisää sopivaan kohtaan toinen uusi aihe otsikolla **Text Files command (Import menu)**. Otsikko tulee merkitä #-alaviitteellä, joka osoittaa tekstiin **HID_IMPORT_TEXTFILE**. Lisää K-alaviite, joka osoittaa tekstiin **Text Files** ja $-alaviite, joka osoittaa tekstiin **Importing Text Files**. Lisää aiheen lyhyt kuvaus. Käytä kuvaa 5.17 mallina.



Kuva 5.17 Importing Text Files -ohjeaihe

6. Tallenna ja sulje AfxCore.rtf-tiedosto.

Huomio Todellisessa kehitysprojektissa nimeäisit todennäköisesti AfxCore.rtf ja AfxPrint.rtf -tiedostot uudelleen sen jälkeen, kun niistä tulee sovellukseesi sovitettuja. Sinun pitää tämän jälkeen muuttaa viittaukset Visual Studiossa ja Microsoft Help Workshop -projektissa. Olemme ohittaneet nämä vaiheet harjoituksessa pitääksemme sen lyhyenä.

---

7. Käännä MyHelpApp-sovellus uudelleen ja käynnistä se. Katsele muokkaamaasi MyHelpApp hakemistosivua painamalla F1. Hae Importing Text Files aihe etsimällä ”Import menu” ja sitten “Text files”. Sulje MyHelpApp-sovelluksen ohje.
8. Tarkista, että tilanteenmukainen ohje toimii **Import**-valikon **Text Files** –toiminnon kanssa.
9. Avaa **Help Topics** dialogi ja etsi “Text files”-aihe **Index**-välilehdeltä, ja tee aiheesta avainsanahaku **Find**-välilehdellä.

Huomio Tämä on mahdollista, koska lisäsit K ja $-alaviitteet AfxCore.rtf-tiedostoon.

HTML-ohje

HTML-ohje on Microsoftin uuden sukupolven ohjekehitysympäristö, joka hyödyntää Microsoft Internet Explorerin komponentteja ohjeen näyttämisessä. Voit käyttää HTML, ActiveX, Java, skripti (JavaScript ja Microsoft Visual Basic Scripting Edition) ja HTML-kuvaformaatteja (.jpeg, .gif), jolloin sovelluksen ohjeesta saadaan täysin toimivan Web-palvelun kaltainen. Ohjeesi voi sisältää linkkejä ulkoisiin tietolähteisiin kuten yhtiösi teknisen tuen Web-sivulle.

HTML-ohjeen käyttämä katseluohjelma sisältää työkalurivin ja sisällys/indeksi-kontrollin, jotta ohjeen käyttäminen olisi käyttäjälle helpompaa. Sinun tulisi nyt tuntea HTML-ohjeen käyttöliittymä, koska Visual C++ 6.0:n ohjejärjestelmä hyödyntää sitä.

Jossain vaiheessa Microsoft tulee todennäköisesti tarjoamaan MFC-sovellusrungossa HTML-ohjeelle samanlaisen tuen kuin WinHelpille. Ennen sitä sinun täytyy toteuttaa sovelluksesi HTML-ohje itse, jos haluat sitä käyttää. Tehtävää helpottamaan on kuitenkin olemassa HTML Help Workshop, kehitystyökalu, jolla voidaan tehdä HTML-ohjeita ja joka muistuttaa WinHelpin Help Workshopia. Parasta on, että HTML Help Workshopin avulla voit muuntaa WinHelp-projektin (.hpj-tiedosto) HTML Help -projektiksi (*.hhp*-tiedosto). WinHelp-projektin rtf-tiedostot muunnetaan HTML-tiedostoiksi, .bmp-tiedostot .gif-tiedostoiksi ja WinHelpin .cnt-tiedostot HTML-ohjeen vastaaviksi *.hhc*-tiedostoiksi. HTML-ohjekääntäjä tuottaa *.chm*-tiedoston — pakatun HTML-tiedoston.

Tässä kirjassa ei käydä HTML-ohjeen tekemistä yksityiskohtaisesti läpi. Mukaan on kuitenkin ottettu muutamia harjoituksia, joiden avulla pääset alkuun. Ensim-mäisessä harjoituksessa katsotaan, kuinka voit muuntaa HTML Help Workshopia käyttäen MyHelpApp:iin luomasi WinHelp-projektin HTML-ohjeprojektiksi MyHHelp.hhp. Voit tämän jälkeen kääntää MyHHelp-ohjetiedoston ja lisätä sen näyttämisessä tarvittavan koodin MyApp-sovellukseen.

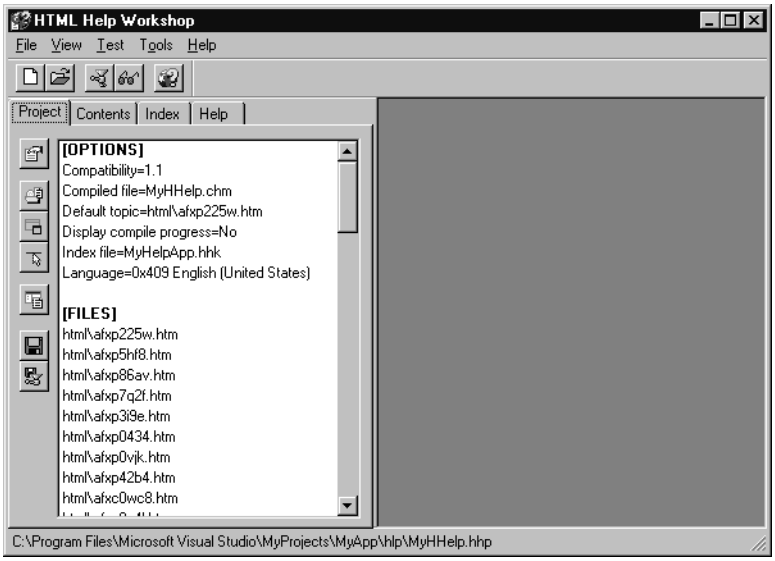
Ennen näiden harjoitusten tekemistä koneellesi täytyy asentaa HTML-ohje.

► **HTML-ohjeen asentaminen**

1. Käynnistä **Htmlhelp.exe**, joka on Visual C++:n tai Visual Studion ensimmäisellä CD:llä kansiossa \HtmlHelp.
2. Asenna ohjeita seuraten HTML-ohje oletussujaintiin: **c:\Program Files\HTML Help Workshop**.

► **WinHelp-projektin MyHelpApp muuntaminen**

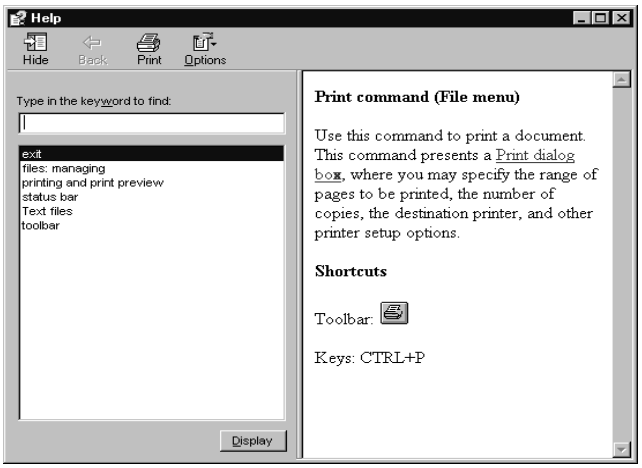
1. Sulje kaikki avoimet Visual Studio -projektit. Tee Windowsin Resurssienhallintaa käyttämällä **hlp**-niminen alikansio MyApp-kansiolle.
2. Valitse **Käynnistä**-valikosta **Programs**, josta edelleen **HTML Help Workshop**, ja napauta sieltä **HTML Help Workshop**.
3. Valitse HTML Help Workshopin **File**-valikosta **New**.
4. Määrää, että luot uuden projektin napauttamalla **OK**.
5. Valitse New Project Wizardin ensimmäisessä vaiheessa **Convert WinHelp Project** -valintaruutu ja napauta **Next**.
6. Kirjoita New Project Wizardin toisessa vaiheessa MyHelpApp.hpj-projektin täudellinen polku ensimmäiseen muokkausruutuun (esimerkiksi: **c:\Program Files\Microsoft Visual Studio\MyProjects\MyHelpApp\hlp\ MyHelpApp.hpj**).
7. Kirjoita toiseen muokkausruutuun kohdassa 1 luomasi **hlp**-kansion täydellinen polku. Aseta luotavan HTML-ohjeprojektin nimeksi **MyHHelp** (esimerkiksi: **c:\Program Files\Microsoft Visual Studio\MyProjects\MyApp\hlp\MyHHelp**).
8. Jatka napauttamallla **Next** ja napauta sitten **Finish**. MyHHelp.hhp-tiedosto luodaan ja se avautuu HTML Help Workshopiin, kuten kuvassa 5.18.



Kuva 5.18 HTML Help Workshop

9. Luo ohjetiedosto napauttamalla **Save all project files and compile** -painiketta. Se on vasemmassa reunassa pystyssä olevan työkalurivin alin painike.

Kun käännös on valmis, sulje HTML Help Workshop. Hae Windowsin Resurssienhallintaa käyttämällä MyHHelp.chm-tiedosto ..\MyApp\hlp kansiosta. Avaa kaksoisnapauttamalla MyHHelp.chm-tiedosto HTML Help sovellukseen. Näet kuinka MyHelpApp-sovellukseen luomasi WinHelp-tiedosto on muutettu siististi HTML Help -muotoon kuten kuvassa 5.19.



Kuva 5.19 MyHHelp.chm-tiedosto

Seuraavassa harjoituksessa opit, kuinka voit HTML Help API:a käyttämällä kutsua MyHHelp.chm Help -tiedostoa MyApp-sovelluksesta. Jotta tämä olisi mahdollista sinun täytyy linkittää HTML Help -kirjasto sovellukseesi.

► **HTML-ohjeen kutsuminen MyApp-sovelluksesta**

1. Avaa Visual C++:ssa MyApp-projekti. Valitse **Project**-valikosta **Settings**.
2. Napauta **Link**-välilehteä. Napauta **Category**-ruudusta **Input**.
3. Kirjoita **Object/library modules** -ruutuun **htmlhelp.lib**.
4. Kirjoita **Additional library path** -ruutuun sen hakemiston polku, johon htmlhelp.lib-tiedosto on asennettu (esimerkiksi: **c:\Program Files\HTML Help Workshop\lib**).
5. Napauta **C/C++**-välilehteä. Valitse **Category**-ruutuun **Preprocessor**.
6. Kirjoita **Additional include directories** -ruutuun sen hakemiston polku, johon htmlhelp.h-tiedosto asennetaan (esimerkiksi: **c:\Program Files\HTML Help Workshop\include**).
7. Tallenna asetukset napauttamalla **OK**.
8. Etsi ja avaa FileViewissä StdAfx.h-tiedosto. Lisää seuraava koodirivi muiden #include-lauseiden perään:

```
#include <htmlhelp.h>
```

9. Avaa **IDR_MAINFRAME**-valikko menueditoriin. Lisää uusi komento **&Help Topics** valikkoon **Help**. Varmista, että komento on luotu tunnisteella **ID_HELP_HELPTOPICS** sulkemalla ja avaamalla uudelleen **Menu Item Properties** -dialogi.
10. Lisää ClassWizardia käyttäen **OnHelpHelptopics()**-käsittelijä **ID_HELP_HELPTOPICS**-sanomalle **CMainFrame**-luokkaan.
11. Lisää funktioon seuraava koodirivi:

```
::HtmlHelp(NULL,
    C:\\Program Files\\Microsoft Visual
    Studio\\MyProjects\\MyApp\\hlp\\MyHHelp.chm",
    HH_DISPLAY_TOPIC,  0);
```

    Jos .chm-tiedosto sijaitsee eri kansiossa, sinun täytyy hieman muuttaa koodia. Huomaa, että merkkijonoon kuuluvat kenoviivat täytyy korvata kahdella kenoviivalla \\. Jos merkkijono jakautuu useammalle riville, muista lisätä \
    -jatkomerkki.

    HtmlHelp()-funktio kuuluu HTML API:n ytimeen. Voit sen avulla näyttää .chm-tiedostoja, HTML-tiedostoja, URL-kohteita ja puhdasta, muotoilematonta tekstiä ponnahdusikkunassa. **HtmlHelp()**-funktiolla on monia parametrejä, jotka määrittelevät avautuvan ikkunan ominaisuuksia ja näytettäviä tietoja. Lisätietoja saat Visual C++:n ohjeesta kohdasta “HTML Help API reference”.
12. Käännä MyApp-sovellus. Jos linkittäjä antaa varoituksia kirjastojen välisistä ristiriidoista, voit jättää ne huomiotta — ne johtuvat htmlhelp.lib-kirjaston linkittämisestä, eivätkä vaikuta sovellukseesi. Käynnistä sovellus ja valitse **Help**-valikosta **Help Topics**. HTML Help -sovelluksen tulisi avautua ja MyHHelp.chm-ohjetiedoston tulisi olla siinä näkyvissä.

Oppitunnin yhteenveto

Windows-sovelluksissa on yleensä tilanteen mukainen ohje, jota voidaan käyttää, kun halutaan lisätietoja sovelluksen käyttöliittymästä.

Windows-sovellukset ovat siirtymässä perinteisestä Windowsin WinHelp järjestelmästä, joka pohjautuu RTF-dokumentteihin, uudempaan HTML ohjeeseen, joka pohjautuu käännettyihin HTML-dokumentteihin. Vaikka Visual C++ sisältää kehitysympäristön HTML Help -tiedostojen tekemistä varten, MFC AppWizardin automaattiset toiminnot tukevat edelleen WinHelp-järjestelmän käyttämistä sovelluksissa.

Tavallisesti käyttäjät avaavat tilanteenmukaisen ohjeen painamalla F1-näppäintä. Jos jokin käyttöliittymän osa kuten valikkokomento, työkalurivin painike tai dialogin kontrolli on valittuna, kun F1-näppäintä painetaan, sovelluksen tulisi näyttää tähän osaan liittyvä tilanteenmukainen ohje. Sovelluksessa tulisi olla myös ohjetilaohje. Sovellus siirtyy ohjetilaan, kun käyttäjä painaa SHIFT+F1 tai valitsee **Help**-painikkeen työkaluriviltä. Kun sovellus on ohjetilassa, käyttäjä saa ohjeen käyttöliittymän elementeistä yksinkertaisesti napauttamalla haluamaansa elementtiä hiirellä.

MFC-sovellusrunko sisältää laajan tuen WinHelpille. Valitsemalla vaihtoehdon **Context-sensitive Help** MFC AppWizardin vaiheessa 4 saat projektiisi kaikki WinHelpin tekemiseen tarvittavat osat. AppWizard luo hlp-nimisen kansion projektikansiosi alikansioksi ja luo Help projektitiedoston ja sovelluksesi ohjetiedoston lähdetiedostot. Näihin lähdetiedostoihin kuuluvat .rtf-tiedostot, jotka sisältävät Windows-käyttöliittymän yleisten elementtien kuten **File** ja **Edit** -valikkojen ohjeet. Voit muokata näitä tiedostoja ja muuttaa valmista tekstiä vastaamaan oman sovelluksesi toimintoja.

Ohjeprojektin tiedostoja hallitsee Microsoft Help Workshop -sovellus, joka on asennettu \Program Files\Microsoft Visual Studio\Common\Tools kansioon osana Visual Studion perusasennusta. AppWizard luo sovelluksellesi räätälöidyt käännösohjeet niin, että ohjetiedosto käännetään uudelleen aina, kun sen lähdetiedostoja muutetaan.

Käyttöliittymän toiminnot yhdistetään sovelluksen ohjetiedoston aiheisiin ohjetunnisteiden avulla. Ne luodaan sovelluksen resurssitunnisteiden pohjalta MakeHm-työkalulla ja talletetaan osaksi ohjeprojektia .hm-päätteisiin tiedostoihin.

Sovelluksen ohjetiedoston tunniste on .hlp. Ohje käynnistetään sovelluksen lähdekoodista kutsumalla **CWnd::WinHelp()**-funktiota. Sovellusrunko luo sopivan ohjetunnisteen, jonka se välittää ohjejärjestelmälle **WinHelp()**-funktion parametrinä.

HTM-ohje on Microsoftin seuraavan sukupolven ohjekehitysympäristö, joka hyödyntää Microsoft Internet Explorerin komponentteja ohjeen sisällön näyttämisessä. Voit käyttää hyväksesi HTML-, ActiveX-, Java- ja skriptiformaatteja, sekä HTML:n tukemia kuvatiedostoja toteuttaessasi sovelluksesi ohjejärjestelmää, joka sisältää linkkejä intranetissä tai Internetissä oleviin osoitteisiin.

Tällä hetkellä AppWizard ei osaa luoda automaattisesti HTML ohjejärjestelmää sovellukseesi. Voit luoda järjestelmän käyttämällä HTML Help -kehitysympäristöä, joka toimitetaan Visual C++ 6.0:n mukana. Se sisältää HTML Help Workshopin, joka muistuttaa läheisesti the Microsoft Help Workshopia, ja joka mahdollistaa WinHelp-projektin muuttamisen HTML Help -projektiksi. WinHelp projektin rtf-tiedostot muutetaan HTML-tiedostoiksi. Sovelluksen HTML-ohje sijoitetaan .chm-tiedostoon — pakattuun HTML-tiedostoformaattiin. HTML-ohje saadaan käyttöön sovelluksen lähdekoodista kutsumalla HTML Help API:n funktiota **HTMLHelp()**.

Laboratorio 5: STUploadin tietojen näyttäminen

Laboratoriossa 3 opit, kuinka tehdään STUpload-sovellukseen yksinkertainen **OnDraw()**-funktio, joka tulostaa sovelluksen dokumenttiobjektissa olevat tiedot tekstiriveinä. Sovellus näyttää kaikki tietueet, jotka dokumenttiin on talletettu -tässä vaiheessa kovakoodatun tietojoukon, joka sisältää tiedot kolmen osakkeen kursseista.

Laboratoriossa 5 sovellusta muutetaan kahdella tavalla. Ensiksi tehdään **Select Fund** -dialogi, modaaliton dialogi, josta käyttäjä voi valita sovelluksen dokumenttiobjektiin talletettujen osakkeiden luettelosta haluamansa. Tämän jälkeen luot uuden **CSTUploadView::OnDraw()**-funktion, joka näyttää valitun osakkeen tiedot graafisessa muodossa.

Select Fund -dialogin näyttäminen

Select Fund -dialogin luomiseen käytetään laboratoriossa 4 tehtyä dialogimallia IDD_FUNDDIALOG ja luokkaa **CFundDialog**. Dialogi ei tässä vaiheessa ole vielä valmis. Lopullisessa versiossa ei ole **OK** eikä **Cancel** -painikkeita, koska käyttäjän sallitaan ottaa dialogi esille ja piilottaa se vain **View**-valikon **Select Fund** -komentoa käyttäen (tai työkalupainikkeella). Sinun täytyy kuitenkin ylikuormittaa **OnOK()** ja **OnCancel()** tapahtumakäsittelijät. Koska IDOK ja IDCANCEL sanomat lähetetään käyttäjän painaessa ENTER tai ESC näppäintä, täytyy käsittelijöistä tehdä **Select Fund** -dialogiluokkaan versiot, jotka eivät tee mitään; muussa tapauksessa kutsutaan kantaluokan (**CDialog**) versioita. Muista, että kantaluokan versiot käsittelijöistä kutsuvat **EndDialog()**-funktiota sulkeakseen dialogin.

On paljon helpompi lisätä käsittelijät **OK** ja **Cancel** sanomille, kun painikkeet ovat edelleen mukana dialogissa, joten luodaan **OnOK()** ja **OnCancel()** -käsittelijät tässä vaiheessa.

► **OnOk()-käsittelijän tekeminen**

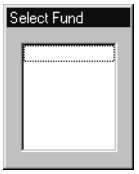
1. Avaa ClassWizard painamalla CTRL+W. Napauta **Class name** -ruudusta **CFundDialog**.
2. Napauta **Object IDs** -ruudusta **IDOK**.
3. Napauta **Messages**-ruudusta **BN_CLICKED**.
4. Napauta **Add Function** ja hyväksy sitten ehdotettu nimi **OnOK** napauttamalla **OK**.
5. Siirry **CFundDialog::OnOK()**-funktion toteutukseen napauttamalla **Edit Code.**
6. Poista tämän funktion rungosta seuraava rivi:

```
CDialog::OnOK();
```

7. Lisää tyhjä **OnCancel()**-funktio käsittelemään IDCANCEL:n BN_CLICKED sanoma toistamalla aiemmat toimenpiteet. Voit nyt poistaa **OK** ja **Cancel** -painikkeet dialogista.

► **Select Fund -dialogi mallin viimeistely**

1. Avaa **IDD_FUNDDIALOG**-resurssi dialogieditorilla.
2. Napauta **OK**. Poista painike painamalla DELETE-näppäintä.
3. Napauta **Cancel**. Paina DELETE.
4. Muuta dialogin kokoa niin, että luetteloruutu mahtuu juuri sinisen ohjausviivoituksen sisään. Dialogin tulisi nyt muistuttaa kuvaa 5.20.



Kuva 5.20 Select Fund -dialogi

Seuraavaksi kirjoitetaan ohjelmakoodi ohjaamaan **Select Fund** -dialogin toimintaa. Yksi **CfundDialog**-objekti luodaan sovelluksen alkuvaiheessa, sovelluksen kantaikkunaobjektin **CMainFrame**:n jäsenenä m_wndFundDialog. Dialogi on sovelluksen kantaikkunan lapsi-ikkuna ja se on näkyvissä tai piilotettuna riippuen CMainFrame-luokan Boolean-jäsenmuuttujan

m_bFundsVisible-tilasta. m_bFundsVisible-muuttujan tila asetetaan käyttämällä valikon tai työkalurivin komentoja.

► **m_wndFundDialog-jäsenmuuttujan lisääminen**

1. Napauta ClassView:ssä hiiren oikealla painikkeella **CMainFrame**-luokan kuvaketta.
2. Napauta pikavalikosta **Add Member Variable**.
3. Kirjoita **Add Member Variable** -dialogin **Variable Type** -ruutuun **CFundDialog**.
4. Kirjoita **m_wndFundDialog-**ruutuun **Variable Name**.
5. Määrää suojaustasoksi **Protected**. Lisää muuttuja napauttamalla **OK**.
6. Lisää protected BOOL-tyyppinen jäsenmuuttuja **m_bFundsVisible** luokkaan **CMainFrame** toistamalla samat toimenpiteet kuin edellä.
7. Avaa otsikkotiedosto kaksoisnapauttamalla **CMainFrame**-luokan kuvaketta ClassView:ssä ja siirry luokanmäärittelyn alkuun.
8. Varmista, että ClassView on lisännyt seuraavan rivin tiedoston alkuun:

```
#include “FundDialog.h”
```

► **m_bFundsVisible-muuttujan alustaminen muodostimessa**

1. Kaksoisnapauta kuvaketta, joka edustaa **CMainFrame-**luokan muodostina.
2. Lisää seuraava rivi funktion runkoon:

```
m_bFundsVisible = FALSE;
```

► **Saantifunktion lisääminen m_bFundsVisible-jäsenmuuttujalle**

1. Palaa **CMainFrame**-luokan määrittelyyn, joka on tiedostossa **MainFrm.h**. Lisää seuraavat koodirivit public-osaan:

(Tämä koodi on asennettu oheisrompulta tiedostoon CH5_01.cpp.)

```
BOOL AreFundsVisible() {return m_bFundsVisible;}
void SetFundsVisible(BOOL bSet)
{
    m_bFundsVisible = bSet;
    if(bSet) m_wndFundDialog.ShowWindow(SW_SHOW);
    else m_wndFundDialog.ShowWindow(SW_HIDE);
}
```

2. Seuraavaksi täytyy alustaa dialogiobjekti kutsumalla sen **Create()**-funktiota ja välittämällä parametrinä sen resurssitunniste. **Create()**-funktio hyväksyy myös toisen parametrin, jolla voidaan määrittää dialogin kantaikkuna. Tällä kertaa käytetään oletusarvoa **NULL**, joka määrittää kantaikkunaksi sovelluksen pääikkunan.

► **m_wndFundDialog-objektin alustaminen**

1. Avaa ClassViewissä **CMainFrame**-luokan kuvake.
2. Siirry funktion toteutusosaan kaksoisnapauttamalla **OnCreate()**-funktiota edustavaa kuvaketta.
3. Lisää funktion loppuun, *ennen* return-komentoa, seuraavat rivit:

```
// Create the fund dialog window
m_wndFundDialog.Create(IDD_FUNDDIALOG);
```

Muista, että dialogilla ei ole WS_VISIBLE-omianisuutta, joten se täytyy näyttää kutsumalla suoraan **CWnd::ShowWindow()**-funktiota.

Dialogin näyttämistä ja kätkemistä ohjataan **View**-valikon **Selection Fund** -komennolla. Seuraavassa harjoituksessa tehdään käsittelijä komennolle **ID_VIEW_FUNDSELECTION**.

► **CMainFrame::OnViewFundselection()-funktio**

1. Avaa ClassWizard painamalla CTRL+W. Napauta **Message Maps** -välilehteä.
2. Napauta **Class Name** -ruudussa **CMainFrame**.
3. Napauta **Object IDs** -ruudussa **ID_VIEW_FUNDSELECTION**.
4. Napauta **Messages**-ruudussa **COMMAND**.
5. Napauta **Add Function**. Hyväksy ehdotettu nimi **OnViewFundselection()** napauttamalla **OK**.
6. Napauta **Edit Code**. MainFrm.cpp-tiedosto avautuu ja kohdistin on sijoitettuna funktion alkuun.
7. Lisää seuraava rivi **OnViewFundselection()**-funktion loppuun:

```
SetFundsVisible(m_bFundsVisible ? FALSE : TRUE);
```

► **CMainFrame::OnUpdateViewFundselection()-funktio**

1. Luo ClassWizardilla **OnUpdateViewFundselection**-funktio **UPDATE_COMMAND_UI**-sanomaa ja **ID_VIEW_ FUNDSELECTION**-objektitunnusta varten.
2. Valitse **Settings**-komento **Project**-valikosta. Napauta **C/C++**-sivulla **C++ Language**-asetus **Category**-ruudusta. Varmista, että **Enable Run-Time Type Information (RTTI)** on valittu. Sulje **Project Settings** -dialogi napauttamalla **OK**.
3. Lisää seuraavat koodirivit **OnUpdateViewFundselection()**-funktion runko-osaan:

   (Tämä koodi on asennettu oheisrompulta tiedostoon CH5_02.cpp.)

```
// Enable the View Funds Selection dialog if
// the document CStockDataList is not empty.
// If enabled, then toggle button state checked/unchecked
// according to whether the window is displayed or hidden
BOOL bEnable = FALSE;
CSTUploadDoc * pDoc =
    dynamic_cast<CSTUploadDoc *>(GetActiveDocument());
if(pDoc)
    bEnable = pDoc->GetDocList().GetCount() ? TRUE : FALSE;
pCmdUI->Enable(bEnable);
if(bEnable)
    pCmdUI->SetCheck(m_bFundsVisible ? 1 : 0);
```

4. Lisää seuraava rivi MainFrm.cpp-tiedoston alkuun:

```
#include "STUploadDoc.h"
```

5. Voit nyt kääntää ja käynnistää STUpload-sovelluksen. Kokeile **Select Fund** -dialogin avaamista ja sulkemista käyttäen valikkokomentoja ja työkalurivin painikkeita.

Osakkeiden nimien esittäminen Funds List -ruudussa

Nyt kun **Select Fund** -dialogi on näkyvissä, täytyy kirjoittaa funktio, joka lataa osakkeiden nimet luetteloruutuun. Koska luettelossa, jota pidetään **CSTUploadDoc::m_DocList**-objektissa, osakkeet ovat järjestettyinä, on helppoa käydä lista läpi ja poimia sieltä eri osakkeiden nimet.

Ennen tämän funktion luomista täytyy luoda MFC:n CListBox-tyyppiä oleva muuttuja.

► **CFundDialog::m_listBox-jäsenmuuttujan lisääminen**

1. Avaa ClassWizard painamalla CTRL+W ja napauta **Member Variables** -välilehteä.
2. Napauta **Class Name** -luettelosta **CFundDialog**-luokkaa.
3. Napauta **Add Variable**. Kirjoita **Member variable name** -ruutuun **m_listBox**.
4. Napauta **Category**-ruudussa **Control**. **Variable Type** -ruudussa lukee **CListBox**.
5. Luo muuttuja napauttamalla **OK** ja sulje sitten ClassWizard napauttamalla uudelleen **OK**.

Seuraavaksi lisätään **UpdateFundList()**-jäsenfunktio **CMainFrame**-luokkaan. Sen tehtävänä on ladata osakkeiden nimet luetteloruutuun. Funktio saa kaksi

parametriä — viittauksen *CStockDataList*-luetteloon, josta haetaan osakkeiden nimet ja merkkijonon, jossa määritellään ensin valittavan osakkeen nimi. Jos toisen muuttujan sisältö on tyhjä merkkijono (oletus) tai määriteltyä merkkijonoa ei löydy listasta, mitään valintaa ei tehdä.

► CMainFrame::UpdateFundList()-funktion toteutus

1. Avaa FileView-näkymässä Header Files -kansio ja avaa **CMainFrm.h**-tiedosto kaksoisnapauttamalla sen kuvaketta.
2. Lisää tiedoston alkuun muiden #include-lauseiden joukkoon seuraava rivi:

```
#include “StockDataList.h”
```

3. Palaa ClassViewiin ja napauta hiiren oikealla painikkeella **CMainFrame**-kuva-ketta. Lisää funktion kanta luokan määrittelyn **public**-osaan käyttämällä **Add Member Function** -toimintoa. Funktion määrittelyn tulisi olla:

```
void UpdateFundList(const CStockDataList & pList,
    CString strCurrentFund = “”);
```

4. Lisää seuraava koodi **UpdateFundList()**-funktion runkoon:

   (Tämä koodi on asennettu oheisrompulta tiedostoon CH5_03.cpp.)

```
// Function to add one entry per fund to fund view list box.
// CStockDataLists are sorted by fund name so this is easy.
CListBox *pListBox = &m_wndFundDialog.m_listBox;
// Empty current contents of list box
pListBox->ResetContent();
CString strLastFund;
POSITION pos = pList.GetHeadPosition();
while(pos)
{
    CStockData sd = pList.GetNext(pos);
    CString strFund = sd.GetFund();
    if(strFund != strLastFund)
        pListBox->AddString(strFund);
    strLastFund = strFund;
}
// Set list box selection to strCurrentFund parameter.
// No selection if parameter empty or not found.
int iPos = pListBox->FindStringExact(-1, strCurrentFund);
pListBox->SetCurSel(iPos);
```

Tutki koodia ja varmista, että ymmärrät tavan, jolla funktio toteuttaa toimintonsa.

UpdateFundList()-funktiota kutsutaan **LoadData()**-funktiosta. Lopullisessa muodossaan **LoadData()**-funktio hoitaa tietueiden lataamisen tekstitiedostosta.

Tässä vaiheessa tehdään funktio, joka yksinkertaisesti lisää muutamia kovakoodattuja tietueita.

► **STUploadDoc::LoadData()-funktion toteutus**

1. Napauta ClassViewissä hiiren oikealla painikkeella **STUploadDoc**-kuvaketta ja käytä **Add Member Function**-dialogia funktion esittelyn lisäämiseen luokan määrittelyn protected-osaan ja funktio kannan lisäämiseen. Funktion esittelyn tulisi olla seuraava:

```
BOOL LoadData(CStdioFile & infile)
```

2. Lisää **LoadData()**-funktion runkoon tiedostossa CH5_04.cpp (asennettu oheisrompulta) olevan koodin osa:

```
m_DocList.AddTail(CStockData(_T("ARSC"),
COleDateTime(1999, 4, 1, 0, 0, 0), 22.33));
// ... more records added here
m_DocList.AddTail(CStockData(_T("COMP"),
COleDateTime(1999, 4, 5, 0, 0, 0), 19.77));

// Update main window

UpdateAllViews(NULL);

// Update Fund Selection dialog box
CMainFrame * pWnd =
      dynamic_cast<CMainFrame *> (AfxGetMainWnd());
if(pWnd)
{
      pWnd->UpdateFundList(m_DocList);
      // Show fund window after loading new funds
      pWnd->SetFundsVisible(TRUE);
}

return TRUE;
```

3. Lisää seuraava rivi CSTUploadDoc.cpp-tiedoston alkuun:

```
#include "MainFrm.h"
```

4. Etsi CSTUploadDoc-muodostin. Poista kaikki toteutusosan koodi niin, että pelkkä kanta jää jäljelle:

```
CSTUploadDoc::CSTUploadDoc()
{
}
```

5. Etsi **CSTUploadDoc::OnDataImport()**-funktio. Lisää funktion loppuun ennen viimeistä sulkua seuraava **LoadData()**-funktiota kutsuva koodi:

```
if(nID == IDOK)
{
    CStdioFile aFile;
    LoadData(aFile);
}
```

Voit nyt kääntää STUpload-sovelluksen. Valitse **Import** komento **Data**-valikosta. Avaa **Open**-dialogia käyttäen Ch5Test.dat-tiedosto ..\Chapter 5\Data -kansiosta. ARSC, BBIC ja COMP -osakkeiden tietojen tulisi näkyä ja **Select Fund** -dialogin tulisi näyttää nämä samat osakkeet.

Select Fund -dialogin sanomien käsittely

Select Fund -dialogin avulla käyttäjän tulisi voida rajoittaa näytettävien kurssitietojen määrää niin, että vain valittuna olevan osakkeen kurssi näytetään. Olet jo täyttänyt luetteloruudun vaihtoehdoilla — nyt täytyy vain toimia käyttäjän valintojen mukaan.

Aluksi lisätään m_strCurrentFund-muuttuja tällä hetkellä valituna olevan osakkeen nimen tallentamista varten. Seuraavassa luvussa muuttujasta tehdään pysyvä niin, että sen arvo voidaan tallentaa osana dokumentin tietoja. Näin ollen m_strCurrentFund-muuttuja on **CSTUploadDoc**-luokan jäsen.

Luetteloruudun tapahtumasanomalle luodaan käsittelijä, jonka avulla varmistutaan siitä, että muuttuja sisältää aina **Select Fund** -dialogissa valittuna olevan arvon. Luetteloruutu lähettää LBN_SELCHANGE-sanoman kantaikkunalleen (**CFundDialog**-objekti) aina valinnan muuttuessa. Voit käyttää ClassWizardia tämän sanoman käsittelijän luomiseen.

Myös **CSTUploadView::OnDraw()**-funktiota muutetaan niin, että se viittaa dokumenttiobjektin “valittuna oleva osake”-muuttujaan, jolloin varmistutaan siitä, että se esittää vain valitun osakkeen tiedot.

► **CDocument::m_strCurrentFund-jäsenen lisääminen**

Napauta hiiren oikealla painikkeella **CDocument**-kuvaketta ClassViewissä ja lisää protected jäsenmuuttuja **m_strCurrentFund**.

► **m_strCurrentFund-jäsenmuuttujan alustaminen muodostimessa**

1. Kaksoisnapauta kuvaketta, joka edustaa **CSTUploadDoc**-luokan muodostinta.
2. Lisää seuraava rivi funktion runkoon:

```
m_strCurrentFund = “”;
```

► **Saantifunktion lisääminen m_strCurrentFund-jäsenfunktiota varten**

Lisää seuraavat koodirivit **CDocument**-luokan määrittelyn **public**-osaan:

```
CString GetCurrentFund () {return m_strCurrentFund;}
void SetCurrentFund (CString strSet){m_strCurrentFund= strSet;}
```

► **LBN_SELCHANGE-sanoman käsittelijän lisääminen CFundDialog-luokkaan**

1. Avaa ClassWizard painamalla CTRL+W. Napauta **Message Maps** -välilehteä.
2. Napauta **Class Name** -ruudusta **CFundDialog**.
3. Napauta **Object IDs** -ruudusta **IDC_FUNDLIST**.
4. Napauta **Messages** -ruudusta **LBN_SELCHANGE**.
5. Napauta **Add Function**. Hyväksy käsittelijälle ehdotettu nimi **OnSelchangeFundlist()** napauttamalla **OK**.
6. Napauta **Edit Code**. FundDialog.cpp-tiedosto avautuu ja kohdistin on funktion alussa.

► **CMainFrame::OnSelchangeFundlist()-funktion toteutus**

1. Lisää seuraavat koodirivit **OnSelchangeFundlist()**-funktion runko-osaan:

   (Tämä koodi on asennettu oheisrompulta tiedostoon CH5_05.cpp.)

```
CMainFrame * pWnd =
    dynamic_cast<CMainFrame *> (AfxGetMainWnd());
ASSERT_VALID(pWnd);

CSTUploadDoc * pDoc =
    dynamic_cast<CSTUploadDoc *>(pWnd->GetActiveDocument());
ASSERT_VALID(pDoc);

CString strCurFund;

int sel = m_listBox.GetCurSel();

if(sel == LB_ERR) sel = 0;

m_listBox.GetText(sel, strCurFund);

pDoc->SetCurrentFund(strCurFund);

pDoc->UpdateAllViews(NULL);
```

2. Lisää seuraavat koodirivit FundDialog.cpp-tiedoston alkuun:

```
#include "MainFrm.h"
#include "STUploadDoc.h"
```

► **CSTUploadView::OnDraw()-funktion muutokset**

1. Etsi silmukka **CSTUploadView::OnDraw()**-funktion loppuosasta. Lisää seuraava koodi **CStockDataList::GetNext()**-kutsun eteen:

```
if(sd.GetFund() != pDoc->GetCurrentFund()) continue;
```

Silmukan tulisi nyt näyttää kokonaisuudessaan seuraavalta:

```
while(pos)
{
    CStockData sd = pData.GetNext(pos);

    if(sd.GetFund() != pDoc->GetCurrentFund())continue;

    pDC->TextOut(10, yPos, sd.GetAsString());
    yPos += nTextHeight;
}
```

2. Käännä ja käynnistä STUpload-sovellus. Kokeile tietojen lataamista kuten aiemminkin. Huomaat, että nyt yhtään tietoa ei ilmesty automaattisesti näkyville, mutta heti kun valitset osakkeen **Select Fund** -dialogista, asiaan kuuluvat tiedot tulevat näyttöön.

Select Fund -dialogin asettaminen päällimmäiseksi

Select Fund -dialogi on hyvin tärkeä osa STUpload-sovelluksen käyttöliittymää. Sitä käytetään jatkuvasti vaihdettaessa osakkeesta toiseen operaattorin käydessä läpi tiedostoon tallennettuja tietoja. Nykyisessä toteutuksessa se jää kuitenkin helposti piiloon käyttäjän napauttaessa pääikkunaa — esimerkiksi valitessaan valikkokomentoja. **Select Fund** -dialogi on niin pieni, että sen saaminen takaisin näkyviin suuremman ikkunan takaa on vaivalloista.

Tämä ongelma voidaan ratkaista tekemällä **Select Fund** -dialogista päällimmäinen ikkuna — eli ikkuna, joka on sovelluksessa aina toisen ikkunan päällä. Päällimmäi-nen ikkuna on näkyvissä, vaikka sillä ei olisikaan fokusta.

Päällimmäisen ikkunan tyyli on WS_EX_TOPMOST. MFC-sovelluksessa voit tehdä tämän asetuksen kutsumalla **CWnd::SetWindowPos()**-funktiota käyttäen **wndTopMost**-vakion osoitetta ensimmäisenä parametrinä.

► **Select Fund -dialogin asettaminen päällimmäiseksi ikkunaksi**

1. Aloita koodin muokkaaminen kaksoisnapauttamalla ClassViewissä **CMainFrame::OnCreate()**-funktion kuvaketta.

2. Lisää funktion loppuun heti seuraavan rivin perään:

```
m_wndFundDialog.Create(IDD_FUNDDIALOG);
```

ja *ennen* return-komentoa seuraava koodirivi.

```
m_wndFundDialog.SetWindowPos(&wndTopMost, 0, 0, 0, 0,
    SWP_NOMOVE | SWP_NOSIZE);
```

Tässä oli kaikki mitä tarvitaan **Select Fund** -dialogin pitämiseen sovelluksen pääikkunan päällä. Valitettavasti tästä seuraa epätoivottu sivuvaikutus. Dialogi nimittäin jää myös *kaikkien muidenkin* sovellusikkunoiden päälle — jopa silloin, kun STUpload-sovellus ei ole aktiivinen. Jos käännät ja käynnistät sovelluksen ja ko-keilet sen toimintaa tässä vaiheessa, huomaat, että **Select Fund** -dialog pysyy näkyvissä vaikka STUpload-sovellus pienennettäisiin tehtäväpalkkiin.

Tämän ongelman ratkaisemiseksi täytyy dialogi piilottaa (jos se on näkyvissä) ai-na, kun *koko sovellus* menettää fokuksen, ja tuoda takaisin näkyviin, kun fokus siirtyy takaisin sovellukseen. Tämä tehdään tekemällä käsittelijä WM_ ACTIVATEAPP-sanomalle, jota kutsutaan käyttäjän siirtyessä sovelluksesta toiseen. ClassWizardia käyttäen luodaan käsittelijä, joka ylikuormittaa **CWnd:: OnActivateApp()**-metodin. Framework kutsuu tätä funktiota ja lähettää sille Boolean-parametrin, joka kertoo aktivoitiinko vai deaktivoitiinko sovellus.

► Käsittelijän lisääminen WM_ACTIVATEAPP-sanomalle

1. Avaa ClassWizard painamalla CTRL+W. Napauta **Message Maps** -välilehteä.
2. Napauta **Class Name** -ruudusta **CMainFrame**.
3. Napauta **Object IDs** -ruudusta **CMainFrame**.
4. Napauta **Messages** -ruudusta **WM_ACTIVATEAPP**.
5. Napauta **Add Function**.
6. Napauta **Edit Code**. MainFrm.cpp-tiedosto avautuu ja osoitin on sijoitettuna funktion toteutusosan alkuun.

► CMainFrame::OnActivateApp()-funktion toteutus

1. Lisää seuraavat rivit **OnActivateApp()**-funktion runkoon kantaluokan kutsun jälkeen:

(Tämä koodi on asennettu oheisrompulta tiedostoon CH5_15.cpp.)

```
if(bActive)
{
    if(AreFundsVisible())
        m_wndFundDialog.ShowWindow(SW_SHOW);
}
else
```

```
{
    if(AreFundsVisible())
        m_wndFundDialog.ShowWindow(SW_HIDE);
}
```

2. Käännä sovellus ja varmista, että **Select Fund -**dialogi pysyy STUpload-sovellusikkunan päällä, mutta jää piiloon muiden sovellusten ollessa aktiivisia.

STUpload-sovelluksen tietojen näyttäminen

Seuraavaksi voidaan toteuttaa valittuna olevan osakkeen hintatietojen esittäminen graafisessa muodossa. Ensimmäiseksi täytyy määritellä dokumentin koko niin, että **CScrollViewin** tuottamat vierityspalkit saadaan näkymään oikein. STUpload-sovelluksessa tulosteen koko pysyy vakiona, vaikka sovelluksen tietomäärä vaihteleekin tekstitiedostojen lataamisen seurauksena. Vain yksi käyrä on näkyvissä kerrallaan. Käyrän koko sovitetaan lasertulostimen vaaka-arkin sivukokoon — tässä tapauksessa leveys 11 tuumaa ja korkeus 8.5 tuumaa.

► **Vierityskoon asettaminen STUpload-sovelluksessa**

1. Avaa ClassViewissä **CSTUploadView**-luokan kuvake.
2. Aloita funktion muokkaaminen kaksoisnapauttamalla **OnInitialUpdate()**-kuvaketta.
3. Muokkaa seuraavia rivejä:

```
sizeTotal.cx = sizeTotal.cy = 100;
SetScrollSizes(MM_TEXT, sizeTotal);
```

niin, että ne näyttävät seuraavilta:

```
sizeTotal.cx = 1100;
sizeTotal.cy = 850;
SetScrollSizes(MM_LOENGLISH, sizeTotal);
```

CView::OnDraw()-funktio korvataan nyt lopullisella versiolla. Tämä funktio ko-koaa tiedostossa tietystä osakkeesta säilytettävän tiedon yhteen taulukkoon. Se käyttää tietoja laskiessaan sopivaa skaalausta päivämäärille (x-akseli) ja hinnoille (y-akseli). Tiedot esitetään käyränä, josta operaattorin tulisi pystyä huomaamaan virheelliset tiedot helposti, koska tarkat arvot esitetään käyrän taitekohdissa. Käy koodi läpi niin näet, kuinka MFC-piirtotyökaluluokkia ja GDI:n piirtofunktioita käytetään tulosteen esittämiseen näytöllä.

► **OnDraw()-metodin toteutus**

1. Avaa ClassViewissä **CSTUploadView**-luokan kuvake.
2. Aloita metodin muokkaaminen kaksoisnapauttamalla **OnDraw()**-kuvaketta.

3. Korvaa metodi *kokonaan* seuraavalla koodilla:

(Tämä koodi on asennettu oheisrompulta tiedostoon CH5_07.cpp.)

```
void CSTUploadView::OnDraw(CDC* pDC)
{
    CSTUploadDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // Save the current state of the device context
    int nDC = pDC->SaveDC();

    const CStockDataList & pData = pDoc->GetDocList();

    // Make a small array containing the
    // records for the current fund.
    // We use an array to take advantage of indexed access.
    CArray<CStockData, CStockData &> arrFundData;

    POSITION pos = pData.GetHeadPosition();

    while(pos)
    {
        CStockData sd = pData.GetNext(pos);

        if(sd.GetFund() == pDoc->GetCurrentFund())
            arrFundData.Add(sd);
    }

    int nPrices = arrFundData.GetSize();
    if(nPrices == 0)
        return;

    // Some constant sizes (in device units)
    const int AXIS_DIVIDER_LENGTH = 6;
    const int AXIS_FONT_HEIGHT = 24;
    const int HEADING_FONT_HEIGHT = 36;

    // Create font for axis labels
    CFont AxisFont;
    if(AxisFont.CreateFont(AXIS_FONT_HEIGHT, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, FF_ROMAN, 0))

        pDC->SelectObject(&AxisFont);
    else
    {
        AfxMessageBox("Unable to create Axis font");
        return;
    }
```

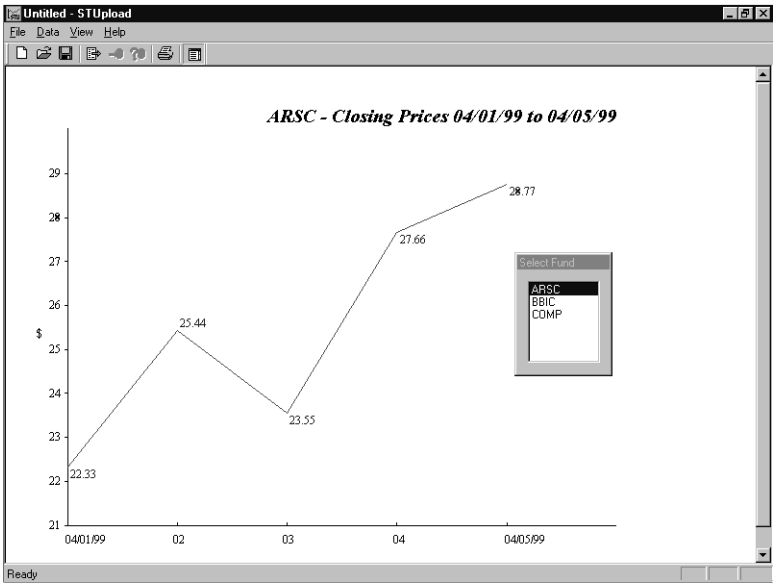
```
CPen AxisPen;
if(AxisPen.CreatePen(PS_SOLID, 1, RGB(0,0,0)))
    pDC->SelectObject(&AxisPen);
else
{
    AfxMessageBox("Unable to create Axis Pen");
    return;
}

// Array to graph coordinates as we go
CArray<CPoint, CPoint&> CoordArray;
for(int i = 0; i < nPrices; i++)
    CoordArray.Add(CPoint(0, 0));

// Set viewport origin to bottom left corner of window
CPoint ptBottomLeft(0, -850);
pDC->LPtoDP(&ptBottomLeft);
pDC->SetViewportOrg(ptBottomLeft);

// Base coordinates for axes
const CPoint ORIGIN(100, 100);
const CPoint Y_EXTENT(ORIGIN.x, ORIGIN.y + 650);
const CPoint X_EXTENT(ORIGIN.x + 900, ORIGIN.y);

// Draw axes
pDC->MoveTo(Y_EXTENT);
pDC->LineTo(ORIGIN);
pDC->LineTo(X_EXTENT);

int nLabelPos = Y_EXTENT.y + ((ORIGIN.y - Y_EXTENT.y) / 2);
pDC->TextOut(ORIGIN.x - 50, nLabelPos, '$');

// Divide x-axis into number of prices held in the file
int nXIncrement = (X_EXTENT.x - ORIGIN.x) / nPrices;

double nMaxPrice = 0;
double nMinPrice = 0;

for(i = 0; i < nPrices; i++)
{
    int xPoint = (ORIGIN.x + (i * nXIncrement));
    CoordArray[i].x = xPoint;

    pDC->MoveTo(xPoint, ORIGIN.y);
    pDC->LineTo(xPoint, ORIGIN.y + AXIS_DIVIDER_LENGTH);

    COleDateTime aDate = arrFundData[i].GetDate();
    double aPrice = arrFundData[i].GetPrice();

    nMaxPrice = max(nMaxPrice, aPrice);
```

```
        nMinPrice = nMinPrice == 0 ?
            nMaxPrice :
            min(nMinPrice, aPrice);

        CString strDate = aDate.Format("%m/%d/%y");

        if(i == 0 || i == (nPrices-1))
            pDC->TextOut(xPoint-2,
                ORIGIN.y - AXIS_FONT_HEIGHT / 2, strDate);
        else
        {
            CString strDay = strDate.Mid(
                strDate.Find('/') + 1);
            strDay = strDay.Left(strDay.Find('/'));
            pDC->TextOut(xPoint-6,
                ORIGIN.y - AXIS_FONT_HEIGHT / 2, strDay);
        }
    }

    // Divide y-axis into suitable scale based on
    // the difference between max and min prices on file
    nMaxPrice += 2.0;
    nMinPrice -= 1.0;
    int iScale = int(nMaxPrice) - int(nMinPrice);

    int nYIncrement = (ORIGIN.y - Y_EXTENT.y) / iScale;

    for(i = 0; i < iScale; i++)
    {
        int yPoint = (ORIGIN.y - (i * nYIncrement));
        pDC->MoveTo(ORIGIN.x, yPoint);
        pDC->LineTo(ORIGIN.x - AXIS_DIVIDER_LENGTH, yPoint);

        int iCurrentPrice = int(nMinPrice) + i;

        for(int j = 0; j < nPrices; j++)
        {
            double aPrice = arrFundData[j].GetPrice();
            if(aPrice >= double(iCurrentPrice) &&
                aPrice < double(iCurrentPrice) + 1.0)
            {
                double dFraction = aPrice -
                    double(iCurrentPrice);
                CoordArray[j].y =
                    yPoint - int(dFraction *
                        double(nYIncrement));
            }
        }
        CString strPrice;
        strPrice.Format("%d", iCurrentPrice);
```

```
        int nTextSize = pDC->GetTextExtent(strPrice).cx;
        nTextSize += 10;

        pDC->TextOut(ORIGIN.x - nTextSize, yPoint+12, strPrice);
}

// Graph figures stored in CoordArray
CPen GraphPen;
if(GraphPen.CreatePen(PS_SOLID, 1, RGB(255,0,0)))  // Red pen
{
        pDC->SelectObject(&GraphPen);
}
else
{
        AfxMessageBox("Unable to create Graph Pen");
        return;
}

// Draw Graph
// Label graph points with price value (in blue)
COLORREF crOldText = pDC->SetTextColor(RGB(0,0,255));

pDC->MoveTo(CoordArray[0]);

for(i = 0; i <nPrices; i++)
{
        pDC->LineTo(CoordArray[i]);
        CPoint TextPoint;
        if((i+1) <nPrices)
        {
                if(CoordArray[i + 1].y >= CoordArray[i].y)
                        TextPoint = CoordArray[i] + CPoint(5, 0);
                else
                        TextPoint = CoordArray[i] + CPoint(5,
                                AXIS_FONT_HEIGHT);
        }
        else
                TextPoint = CoordArray[i] + CPoint(5, 0);

        CString strPrice;
        strPrice.Format("%.2f", arrFundData[i].GetPrice());

        pDC->TextOut(TextPoint.x, TextPoint.y, strPrice);
}

pDC->SetTextColor(crOldText);

// Create heading
CFont HeadingFont;
```

```
    if(HeadingFont.
        CreateFont(HEADING_FONT_HEIGHT, 0, 0, 0, FW_BOLD, 1, 0, 0,
        0, 0, 0, 0, FF_ROMAN, 0));

        pDC->SelectObject(&HeadingFont);
    else
    {
        AfxMessageBox("Unable to create Heading Font");
        return;
    }

    CString strHeading = pDoc->GetCurrentFund();
    strHeading += " - Closing Prices ";

    COleDateTime aDate = arrFundData[0].GetDate();
    strHeading += aDate.Format("%m/%d/%y");
    strHeading += " to ";
    aDate = arrFundData[nPrices - 1].GetDate();
    strHeading += aDate.Format("%m/%d/%y");

    CSize sizeText = pDC->GetTextExtent(strHeading);

    pDC->TextOut(X_EXTENT.x - sizeText.cx,
        Y_EXTENT.y + sizeText.cy, strHeading);

    // Restore the original device context
    pDC->RestoreDC(nDC);
}
```

4. Käännä ja käynnistä STUpload-sovellus. Lataa testitiedot kuten ennenkin. Sovelluksen tietojen tulisi tulla näkyviin samassa muodossa kuin kuvassa 5.21.



Kuva 5.21 STUpload-sovellus

Kertaus

1. Kuinka **DoDataExchange()**-funktiota kutsutaan? Kuinka se määrittelee tiedonsiirron suunnan?

2. Kuinka kontrollin käyttäminen estetään dialogissa?

3. Mitä kahta funktiota käytetään koordinaatiston kohdistamiseen vapaassa kohdistuksessa?

4. Mikä funktio yhdistää piirtotyökalun piirtopintaan?

5. Minkälaisia säikeitä täytyy luoda **CWinThread**-objektia käyttäen?

6. Mikä ero on **CCriticalSection**-objektilla ja **CMutex**-objektilla?

7. Kuinka ohjeen aihetunnisteet (Help context ID) määritellään ohjeen projektitiedostossa?

8. Kuinka tehdään hyperlinkki ohjeprojektin .rtf-tiedostossa?