L U K U 1 1

ActiveX-kontrollien luominen

Tässä luvussa

Luvussa 8 tutustuit muutamiin ActiveX-kontrollien piirteisiin, jotka erottavat ne muista COM-objekteista ja ActiveX-kontrollisäilöön. Säilöhän toteuttaa yleensä käyttöliittymä ja odottaa kontrollin toimivan osana käyttöliittymäänsä. Kontrollilla on rajapinta, jonka kautta säilö voi asettaa sen ominaisuuksia ja kutsua sen metodeja. Kontrolli kommunikoi säilön kanssa laukaisemalla tapahtumia. ActiveX-kontrolleilla on yleensä ominaisuussivu, jonka avulla käyttäjä voi muuttaa kontrollin ominaisuuksia. Kontrollisäilöt pystyvät yleensä tallentamaan sisältämiensä ActiveX-kontrollien ominaisuudet.

Tässä luvussa opit kaksi suosituinta tapaa tehdä ActiveX-kontrolleja Microsoft Visual C++:ssa. Oppitunnilla 1 teet yksinkertaisen ActiveX-kontrollin käyttämällä MFC:tä. Oppitunnilla 2 teet saman kontrollin käyttämällä ATL:ää. Tekemällä saman kontrollin eri tekniikoilla pystyt vertailemaan ja arvioimaan näitä menetelmiä ja päättämäänm milloin käytät mitäkin menetelmää.

Ennen kuin aloitat

Ennen tämän luvun aloittamista sinun tulisi lukea luvut 2-10 ja suorittaa lukuihin liittyvät harjoitukset.

Oppitunti 1: ActiveX-kontrollin tekeminen MFC:llä

MFC yksinkertaistaa ActiveX-kontrollin tekemistä. Käyttämällä MFC:n ActiveX ControlWizardia voit luoda helposti täysin toimivan ActiveX-kontrollin. Tällä oppitunnilla käytetään MFC:tä ActiveX-kontrollin tekemiseen. Kontrollilla on viestirajapinta, se laukaisee tapahtumia ja sillä on ominaisuussivu, jonka avulla pysyviä ominaisuuksia voidaan asettaa ja tutkia.

Tämän oppitunnin jälkeen:

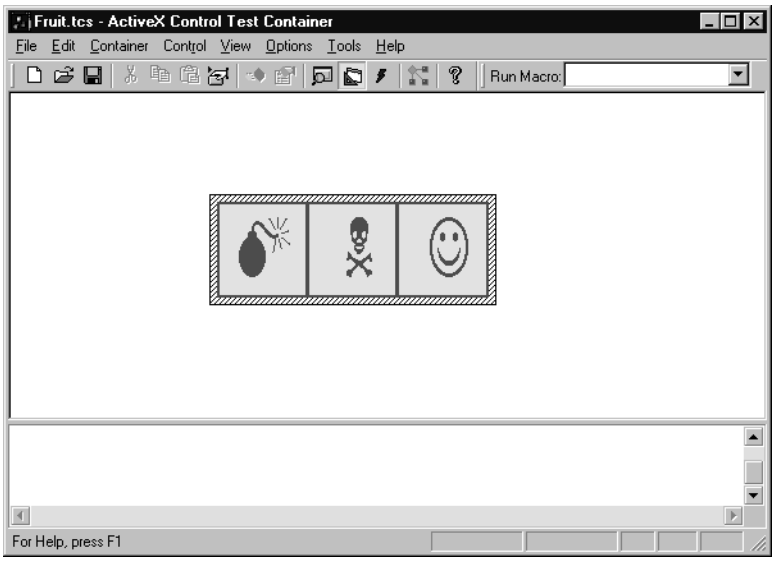
- Tiedät, kuinka MFC:n ActiveX ControlWizardia käytetään ActiveX-kontrollin luomiseen.
- Tiedät, kuinka ClassWizardilla lisätään ominaisuuksia, metodeja ja tapahtumia kontrolliin.
- Tiedät, kuinka kontrollin ominaisuussivu tehdään.
- Tiedät, kuinka ominaisuuksista tehdään pysyviä.
- Tiedät, kuinka ActiveX Control Test Containeria käytetään kontrollin testaamiseen.

Oppitunnin arvioitu kesto: 50 minuuttia

MFC:n ActiveX-kontrolliprojektin luominen

Tällä oppitunnilla tehdään OneArmedBandit ActiveX-kontrolli, joka on ohjelmallinen versio hedelmäpelin tapaisesta peliautomaatista. Kontrollilla on yksi metodi **Play()**, joka arpoo kolmen satunnaisen symbolin rivin kontrollin “rulliin”. Kun kontrollissa on kolmen samanlaisen symboolin yhdistelmä, laukaistaan **Jackpot**-tapahtuma. Kun käyttäjä napauttaa kontrollia, laukaistaan **Click**-tapahtuma. Kuvassa 11.1 seuraavalla sivulla nähdään OneArmedBandit-kontrolli ActiveX Control Test Containerissa.

OneArmedBandit-kontrollilla on ominaisuussivu, jonka avulla käyttäjä voi asettaa **ForeColor** ja **BackColor** ominaisuuksien arvot, ja asettaa oman kontrollikohtaisen **NumberOfSymbols**-ominaisuuden. Tämän ominaisuuden avulla asetetaan kontrollin käyttämien symbolien määrä, eli lisätään tai pienennetään voittomahdollisuuksia. Kaikki kolme ominaisuutta ovat pysyviä.

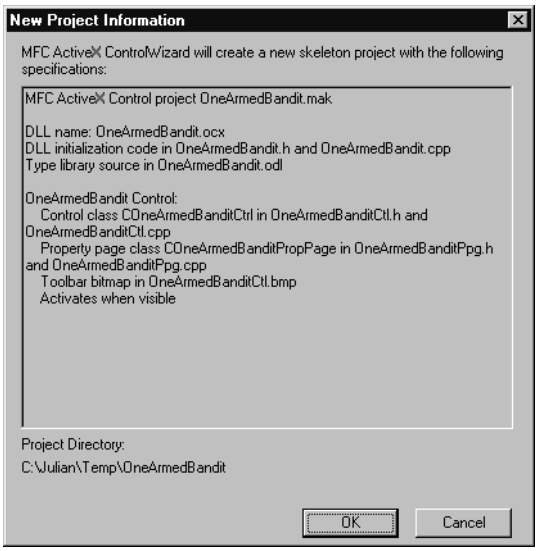


Kuva 11.1 OneArmedBandit ActiveX-kontrolli

Kuten aina MFC:tä käytettäessä ensimmäinen vaihe ActiveX-kontrollin tekemisessä on kehitysprojektin luominen. Seuraavassa harjoituksessa tehdään OneArmedBandit-projekti.

► **OneArmedBandit-projektin luominen**

1. Valitse **File**-valikosta **New** ja napauta sitten **Projects**-välilehteä.
2. Napauta **MFC ActiveX ControlWizard**. Kirjoita **Project name** -ruutuun **OneArmedBandit** ja napauta **OK**.
3. ControlWizardin vaihe 1 avautuu. Silmäile valinnat läpi ja hyväksy oletukset napauttamalla **Next**.
4. Poista **Has an "About" box** -valinta ControlWizardin vaiheessa 2. Napauta **Finish**.
5. **New Project Information** -dialogi avautuu kuten seuraavan sivun kuvassa 11.2.

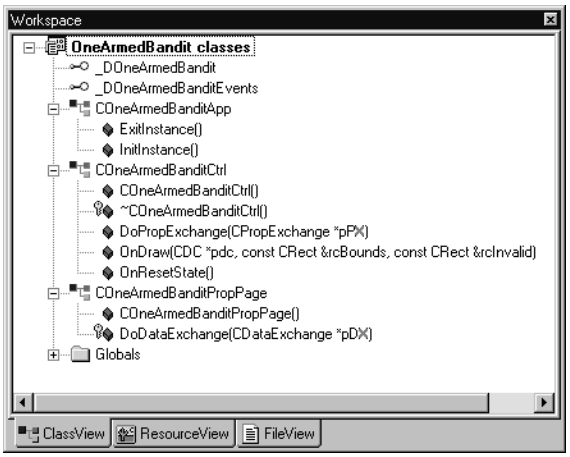


Kuva 11.2 OneArmedBandit-projektin luominen

Dialogissa nähtävissä tiedoissa on muutamia mielenkiintoisia kohtia. Ensinnäkin luodulle DLL:lle annettiin tarkentimeksi .ocx. .ocx-tarkentimen käyttö ei ole ActiveX-kontrolleille pakollista, vaan se on jäänne ajalta, jolloin ActiveX-kontrolleja kutsuttiin OLE-kontrolleiksi. OLE-kontrollit korvasivat vanhemmat Visual Basic Extension (VBX) -kontrollit. Toiseksi tyyppikirjaston lähdekoodi on tiedostossa nimeltä OneArmedBandit.odl. Tämä tiedosto sisältää Object Description Language (ODL) -koodin. ODL oli Interface Definition Languagen (IDL) edeltäjä, ja sen syntaksi on hyvin samankaltainen kuin IDL:n. Microsoft IDL (MIDL) -kääntäjä pystyy kääntämään ODL-koodia.

6. Luo OneArmedBandit-projekti napauttamalla **OK**. Avaa ClassViewin kohteet ja tutki projektiin luotuja luokkia, jotka näet kuvassa 11.3 seuraavalla sivulla.

ActiveX ControlWizard luo luokat, jotka edustavat kontrollin DLL-palvelua, kontrollia itseään ja kontrollin ominaisuussivua. **COneArmedBanditCtrl**-kontrolliluokka periytetään MFC:n luokasta **COleControl**. Tämä tehokas luokka perii luokkien **CWnd** ja **CCmdTarget** toiminnot ja sillä on suuri joukko funktioita, jotka liittyvät ActiveX-kontrollin toimintaan. Näitä funktiota käyttämällä voit käsitellä kontrollin ominaisuuksia, hakea ympäristön ominaisuudet säilöltä, laukaista tapahtumia, toteuttaa ominaisuuksien tallennukset ja hoitaa joukon kontrollin asettamiseen ja näyttämiseen liittyviä toimintoja.



Kuva 11.3 OneArmedBandit-projektin luokat

Kontrollin rajapinnan määrittäminen

ControlWizard määrittelee kontrollille kaksi rajapintaa: **_DOneArmedBandit** ja **_DOneArmedBanditEvents**. Ensimmäiseen lisätään metodit ja ominaisuudet ja toiseen tapahtumat. Vaikka voitkin tehdä nämä lisäykset napauttamalla rajapinnan osaa hiiren kakkospainikkeella ja valitsemalla sopivan komennon pikavalikosta, on suositeltavampaa käyttää ClassWizardia.

Ominaisuuksien lisääminen

Seuraavassa harjoituksessa lisätään ominaisuudet **ForeColor** ja **BackColor**, sekä **NumberOfSymbols** luokkaan **COneArmedBanditCtrl**.

► **Ominaisuuksien lisääminen ActiveX-kontrolliin**

1. Avaa ClassWizard painamalla CTRL+W. Valitse **Automation**-välilehti.
2. Avaa **Add Property** -dialogi napauttamalla **Add Property**.
3. Avaa **External name** -alasvetovalikko, josta näet **COleControl**-luokan tukemat ominaisuudet. Valitse luettelosta **BackColor**.
4. Huomaa, että **Stock**-valintanappi **Implementation**-ryhmästä valitaan automaattisesti. Dialogi ilmoittaa myös, että **GetBackColor()** ja **SetBackColor()** -funktiot luodaan. Lisää **BackColor**-ominaisuus, sekä **Get** ja **Set** -funktiot napauttamalla **OK**.
5. Lisää **ForeColor**-ominaisuus toistamalla samat toimet.
6. Luo erikoisominaisuus avaamalla **Add Property**-dialogi ja kirjoittamalla nimi **NumberOfSymbols** luetteloon **External name**. Huomaa, että ClassWizard luo m_numberOfSymbols-jäsenmuuttujan ominaisuuden arvon

tallentamista varten, ja että se luo myös funktion **OnNumberOf SymbolsChanged()**, jota kutsutaan muutettaessa ominaisuuden arvoa.

7. Valitse **Type**-alasvetovalikosta **short**, ja luo sitten ominaisuus napauttamalla **OK**.
8. Lopeta ominaisuuksien tekeminen napauttamalla **OK**.

ClassViewissä näet uusien ominaisuuksien luettelon **_DOneArmedBandit**-rajapintayksikön alla. Huomaa myös, että jäsenmuuttuja ja käsittelyfunktio on lisätty **COneArmedBanditCtrl**-luokkaan. Kaksoisnapauttamalla **OnNumberOfSymbolsChanged()**-funktiota pääset tutkimaan tiedostossa OneArmedBanditCtl.cpp olevaa lähdekoodia. Funktion oletusversio kutsuu vain yksinkertaiseti **COleControl::SetModifiedFlag()**-funktiota.

Lähempänä OneArmedBanditCtl.cpp-tiedoston alkua näet seuraavan koodin:

```
BEGIN_DISPATCH_MAP(COneArmedBanditCtrl, COleControl)
    //{{AFX_DISPATCH_MAP(COneArmedBanditCtrl)
    DISP_PROPERTY_NOTIFY(COneArmedBanditCtrl, "NumberOfSymbols",
        m_numberOfSymbols, OnNumberOfSymbolsChanged, VT_I2)
    DISP_STOCKPROP_BACKCOLOR()
    DISP_STOCKPROP_FORECOLOR()
    //}}AFX_DISPATCH_MAP

END_DISPATCH_MAP()
```

Tämä koodi toteuttaa yhdessä vastaavan header-tiedostossa olevan **DECLARE_DISPATCH_MAP**-makron kanssa luokan MFC *viestikartan* (dispatch map). Viestikartta on hyvin samantyypinen kuin luvussa 3 käsitelty sanomakartta, mutta se ohjaa Windows-sanomien sijasta Automation-asiakkaiden palvelupyyntöjä luokan ominaisuuksiin ja metodeihin. Jos esimerkiksi Visual Basic -asiakas pyytää **NumberOfSymbols**-ominaisuutta seuraavalla koodilla, kontrolli käyttää viestikarttaa (erityisesti **DISP_PROPERTY_NOTIFY**-makroa) hakiessaan m_numberOfSymbols-muuttujaan talletettua arvoa ja välittäessään sitä takaisin asiakkaalle:

```
Dim myobj As OneArmedBandit
Set myobj = New OneArmedBandit
MsgBox myobj.NumberOfSymbols
```

Kuten voit viestikartan koodista nähdä, perusominaisuudet käyttävät omia makrojaan.

Ominaisuuksien pysyvyys

MFC:n avulla kontrollien ominaisuuksien säilyttäminen on helppoa. **COleControl::DoPropExchange()**-funktio serialisoi kontrollin ominaisuudet tallennuslaitteelle ja takaisin — yleensä hyödynnetään kontrollin säilöä. ActiveX ControlWizard ylikuormittaa kontrolliluokan **DoPropExchange()**-funktion.

Framework välittää tämän funktion osoittimen **CPropExchange**-objektille, joka kapseloi ominaisuuksien välittämiseen liittyvät seikat, kuten välityksen suunnan. **DoPropExchange()**-luokan ylikirjoitettu versio kutsuu kantaluokan versiota serialisoidakseen kontrollin toteuttamat perusominaisuudet. Erikoisominaisuuksien serialisointiin tarvittavan koodin kirjoittaminen on ohjelmoijan vastuulla.

MFC sisältää joukon funktioita, joiden avulla voit serialisoida eri tietotyyppejä. Näiden funktioiden nimet alkavat etuliitteellä *PX_*. Voit sarjoittaa **NumberOfSymbols**-ominaisuuden korvaamalla kontrolliluokan //TODO-komentin kontrolliluokan **DoPropExchange()**-funktiossa **PX_Short()**-funktion kutsulla seuraavasti:

```
void COneArmedBanditCtrl::DoPropExchange(CPropExchange* pPX)
{
    ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
    COleControl::DoPropExchange(pPX);

    PX_Short(pPX, "NumberOfSymbols", m_numberOfSymbols, 3);

}
```

PX_Short()-funktion neljäs argumentti määrittelee ominaisuuden oletusarvon, jota käytetään, mikäli varastoitua arvoa ei pystytä käyttämään. Ylläolevassa esimerkkikoodissa varmistetaan, että uuden kontrollin **NumberOfSymbols**-ominaisuus alustetaan oletusarvolla 3.

Metodien lisääminen

Viestikartta ohjaa myös asiakkaiden metodikutsut luokan jäsenfunktioille. Käytetään ClassWizardia metodin lisäämiseen.

► **Metodin lisääminen ActiveX-kontrolliin**

1. Napauta ClassWizardissa **Automation**-välilehteä ja avaa **Add Method** -dialogi napauttamalla **Add Method**.
2. Kirjoita **External name** -luetteloon **Play**.
3. Valitse **Return type** -alasvetovalikosta **void**.
4. Napauta **OK**, ja luo sitten metodi napauttamalla ClassWizardissa **OK**.

Huomaa, että **Play()**-funktio on lisätty **COneArmedBanditCtrl**-luokkaan. Jos tutkit viestikarttaa huomaat, että sinne on lisätty seuraava rivi:

```
DISP_FUNCTION(COneArmedBanditCtrl, "Play", Play, VT_EMPTY, VTS_NONE)
```

Tapahtumien lisääminen

ClassWizard automatisoi myös kontrollin laukaisemien tapahtumien määrittelyn. Seuraavassa harjoituksessa toteutetaan **Click**-perustapahtuma ja **Jackpot**-erikoistapahtuma.

► **ActiveX-kontrollin tapahtuman määrittely**

1. Napauta ClassWizardissa **ActiveX Events**-välilehteä. Avaa **Add Event** -dialogi napauttamalla **Add Event**.
2. Valitse **External name** -alasvetovalikosta **Click**-perustapahtuma. Lisää tapahtuma napauttamalla **OK**.
3. Napauta **Add Event** kerran uudelleen. Kirjoita **Add Event** -dialogin **External name** -luetteloon **Jackpot**. Napauta **OK**.
4. Lopeta tapahtuman luominen napauttamalla ClassWizardissa **OK**.

Huomaa ClassViewissä, että **FireJackpot()**-funktio on lisätty **COneArmedBanditCtrl**-luokkaan. Voit laukaista koodissasi **Jackpot**-tapahtuman tätä funktiota käyttämällä. ClassWizard on lisännyt merkinnän myös luokan *tapahtumakarttaan* (event map). Tapahtumakartta on rakenne, joka muistuttaa huomattavasti ActiveX-kontrollien viestikarttaa ja sitä käytetään tapahtumien toteuttamiseen. Seuraavassa näet **COneArmedBanditCtrl**-luokan tapahtumakartan:

```
BEGIN_EVENT_MAP(COneArmedBanditCtrl, COleControl)
    //{{AFX_EVENT_MAP(COneArmedBanditCtrl)
    EVENT_CUSTOM("Jackpot", FireJackpot, VTS_NONE)
    EVENT_STOCK_CLICK()
    //}}AFX_EVENT_MAP
END_EVENT_MAP()
```

Ominaisuussivun tekeminen

Nyt, kun ActiveX-kontrollin rajapinnat on tehty, voidaan lisätä ominaisuussivu, jonka kautta kontrollin käyttäjä voi tutkia ja asettaa kontrollin ominaisuuksia. Framework käyttää **PROPPAGEID**-makrojen sarjaa määritellessään ominaisuussivujen tunnistetaulukkoa kontrollin ominaisuussivua varten. ActiveX ControlWizardin OneArmedBandit -projektiin tekemä oletuskoodi on OneArmedBandit.cpp-tiedostossa ja se näyttää seuraavalta:

```
BEGIN_PROPPAGEIDS(COneArmedBanditCtrl, 1)
    PROPPAGEID(COneArmedBanditPropPage::guid)
END_PROPPAGEIDS(COneArmedBanditCtrl)
```

Huomaa, että tämä rakenne ei, poiketen viesti- ja tapahtumakartoista, ole ClassWizardin ylläpitämä. Sinun täytyy huolehtia itse koodin ylläpidosta.

Mukautetut ominaisuussivut

Edellä esitetyssä koodissa taulukon **PROPPAGEID**-merkintä viittaa kontrollille luotuun oletusominaisuussivuun. ActiveX ControlWizard luo dialogimalli-resurssin ja dialogiluokan (joka pohjautuu **COlePropertyPage**-luokkaan), joita voit muokata niin, että myös kontrollin erityisominaisuuksia voidaan käsitellä niiden avulla.

Seuraavassa harjoituksessa tehdään ominaisuussivu, jonka avulla käyttäjä voi asettaa **NumberOfSymbols**-ominaisuuden.

► **Oman ominaisuussivun tekeminen**

1. Avaa ResourceViewissä Dialog-kansio. Aloita dialogimallin muokkaaminen kaksoisnapauttamalla kohtaa **IDD_PROPPAGE_ ONEARMEDBANDIT**.
2. Poista //TODO-teksti. Lisää seliteteksti- ja muokkausruutukontrollit niin, että dialogi näyttää samalta kuin kuvassa 11.4.

Kuva 11.4 Oman ominaisuussivun tekeminen.

   Anna muokkausruudulle tunniste **IDC_NUMSYMBOLS**. Valitse **Edit Properties** -dialogin **Styles**-välilehdeltä **Number**-valintaruutu.
3. Avaa ClassWizard painamalla CTRL+W. Valitse **Member Variables** -välilehti.
4. Lisää Value kategorian jäsen **m_numsymbols** napauttamalla **Add Variable**. Muuttujan tulee olla tyypiltään **short**. Kirjoita **Optional property name** -ruutuun **NumberOfSymbols** ja napauta sitten **OK**.
5. Aseta sallituksi alarajaksi **3** ja ylärajaksi **7**. Napauta **OK**.

Tutki **COneArmedBanditPropPage**-luokan **DoDataExchange()**-funktiota. Huomaa, että DDX ja DDV -funktioiden lisäksi ClassWizard on lisännyt seuraavan rivin:

```
DDP_Text(pDX, IDC_NUMSYMBOLS, m_numsymbols, _T("NumberOfSymbols"));
```

Tämä on yksi MFC:n tarjoamista monista *DDP_*-etuliitteisistä funktioista, jotka siirtävät tietoa ActiveX-kontrollin ominaisuuksien ja ominaisuussivun välillä. Huomaa, että kontrollin ominaisuuden viestinimeä on käytetty niin, että luokan **COneArmedBanditPropPage** ja **COneArmedBanditCtrl**-luokan välistä tiedonsiirtoa varten ei enää tarvita lisäkoodia.

Perusominaisuussivut

Koska kontrollille on lisätty **ForeColor** ja **BackColor** -ominaisuudet, täytyy myös niitä vastaavat ominaisuussivut toteuttaa, jotta käyttäjät pääsevät muuttamaan niiden ominaisuuksia. MFC sisältää kolme perusominaisuussivua, joita voidaan käyttää ActiveX-kontrollien kanssa: **CLSID_ CColorPropPage**, **CLSID_CFontPropPage** ja **CLSID_CPicturePropPage**. Nämä sivut sisältävät mainitussa järjestyksessä käyttöliittymän väri-, kirjasin- ja kuvaominaisuuksille. Perusominaisuussivun saat lisättyä lisäämällä toisen **PROPPAGEID**-makron OneArmedBandit.cpp-tiedoston koodiin seuraavalla tavalla:

```
BEGIN_PROPPAGEIDS(COneArmedBanditCtrl, 2)
    PROPPAGEID(COneArmedBanditPropPage::guid)
    PROPPAGEID(CLSID_CColorPropPage)
END_PROPPAGEIDS(COneArmedBanditCtrl)
```

Huomaa, että toinen **BEGIN_PROPPAGEIDS**-makron parametreistä täytyy muuttaa niin, että se vastaa ActiveX-kontrollille määriteltyjen ominaisuussivujen määrää.

OnDraw()-Funktio

MFC ActiveX ControlWizardilla luodut kontrollit ovat käyttöliittymäkontrolleja. Tästä suuri osa kontrollikohtaisesta koodista liittyy kontrollin ulkoasun piirtämiseen. Kuten MFC:n dokumentti/näkymä-sovelluksessa, kaikki kontrollin piirtokoodi on sijoitettu yhteen **OnDraw()**-nimiseen funktioon. Seuraavassa ControlWizardin tekemä oletusfunktio:

```
void COneArmedBanditCtrl::OnDraw(
    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    // TODO: Replace the following code with your own drawing code.
    pdc->FillRect(rcBounds,
    CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
    pdc->Ellipse(rcBounds);

}
```

COleControl::OnDraw()-funktio saa osoittimen MFC:n laiterajapintaobjektiin, ja kontrollin näytöllä näkyvää aluetta vastaavan suorakaiteen määrittävän loogisen yksikön. Piirtokoodissa tulisi välttää kiinteiden arvojen käyttämistä ja kaikki piirtotapahtumat tulisi skaalata suorakaiteen mittasuhteisiin. Näin toimien varmistetaan, että kontrolli on aina näkyvissä ja että sen sisäiset mittasuhteet pysyvät kunnossa. **OnDraw()**-funktiolle väitetään myös suorakaide, joka edustaa kontrollin epäkelpoa aluetta ja jota voidaan näin käyttää piirtokoodin optimoimiseen.

► **COneArmedBanditCtrl::OnDraw()-funktion toteutus**

Korvaa projektissasi oleva koodi seuraavalla:

(Tämä koodi on asennettu oheisrompulta tiedostoon CH11_01.cpp.)

```
void COneArmedBanditCtrl::OnDraw(
        CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
        // Get colors of stock properties
        COLORREF colorBack = TranslateColor(GetBackColor());
        COLORREF colorFore = TranslateColor(GetForeColor());

        // Get dimensions of control
        float ctrlWidth = float(rcBounds.Width());
        float ctrlHeight = float(rcBounds.Height());

        // Setup DC
        CBrush brush(colorBack);
        CBrush * pOldBrush = pdc->SelectObject(&brush);

        CPen pen(PS_SOLID, 3, colorFore);
        CPen *pOldPen = pdc->SelectObject(&pen);

        CFont SymbolFont;
        CFont *pOldFont;

        if(SymbolFont.
                CreateFont(long(ctrlHeight / 1.1), long(ctrlWidth/6),
                0, 0, 0, 0, 0, 0, SYMBOL_CHARSET, 0, 0, 0,
                0,"WingDings"));

                pOldFont = pdc->SelectObject(&SymbolFont);
        else
                pOldFont = SelectStockFont(pdc);

        // Draw bounding rectangle
        pdc->Rectangle(rcBounds);
        pdc->SetBkMode(TRANSPARENT);

        // Draw text
        pdc->SetTextColor(colorFore);
        RECT rect;
        ::CopyRect(&rect, rcBounds);
        CString strDisplay;
        strDisplay.Format("%c %c %c",
                m_symbols[0], m_symbols[1], m_symbols[2]);

        pdc->DrawText(strDisplay, &rect, DT_SINGLELINE | DT_CENTER
                | DT_VCENTER );
```

```
    // Draw vertical bars
    long onethird = long(ctrlWidth / 3);
    CPoint ptTop(rcBounds.TopLeft());
    CPoint ptBtm(rcBounds.left, rcBounds.bottom);

    ptTop.x += onethird; ptBtm.x += onethird;
    pdc->MoveTo(ptTop);
    pdc->LineTo(ptBtm);

    ptTop.x += onethird; ptBtm.x += onethird;
    pdc->MoveTo(ptTop);
    pdc->LineTo(ptBtm);

    // Restore device context
    pdc->SelectObject(pOldFont);
    pdc->SelectObject(pOldPen);
    pdc->SelectObject(pOldBrush);
}
```

Tämä funktio simuloi hedelmäpelin ruutua näyttämällä kolme Wingdings-tyyppistä kirjainta, kuten kuvassa 11.1 näkyy. Itse merkit on sijoitettu kiinteäpituiseen merkkijonoon **m_symbols**, joka on **COneArmedBanditCtrl**-luokan jäsenmuuttuja. Tämä jäsenmuuttuja täytyy lisätä ennen koodin kääntämistä.

► **COneArmedBanditCtrl::m_symbols-merkkijonon lisääminen**

1. Napauta ClassWizardissa hiiren kakkospainikkeella **COneArmedBanditCtrl**-luokkaa. Napauta **Add Member Variable**. Luo seuraava protected-jäsenmuuttuja:

   ```
   TCHAR m_symbols[3];
   ```

2. Etsi **COneArmedBanditCtrl**-muodostin ja lisää seuraava koodirivi, joka alustaa muuttujan:

   ```
   _tcscpy(m_symbols, _T("JJJ"));
   ```

Control-metodin toteutus

Tekämättä on vielä **Play()**-metodi, joka simuloi pelin renkaiden pyörittämisen ja näyttää renkaisiin arvotut symbolit. **Play()**-funktiolle tehtiin tynkäfunktio samalla, kun metodi lisättiin kätyttöliittymään. Nyt täytyy kirjoittaa koodia, joka arpoo merkit **COneArmedBanditCtrl::m_symbols**-merkkijonoon.

► **COneArmedBanditCtrl::Play()-funktion toteutus**

1. Etsi **Play()**-funktion tynkä ja korvaa se seuraavalla koodilla:

   (Tämä koodi on asennettu oheisrompulta tiedostoon CH11_02.cpp.)

```
void COneArmedBanditCtrl::Play()
{
    srand((unsigned)time(NULL));

    _tcscpy(m_symbols, _T("JJJ"));

    for(int i = 0; i < 3; i++)
        m_symbols[i] += UINT(rand() % m_numberOfSymbols);

    InvalidateControl(); // repaints control

    if(m_symbols[0] == m_symbols[1] &&
        m_symbols[1] == m_symbols[2])
        FireJackpot();

}
```

   Modulo-operaattoria (%) käyttäen varmistetaan, että jokaisen m_symbols-merkkijonon merkin arvoa lisätään satunnaisluvulla, joka on nollan ja *m_numberOfSymbols*-arvon välillä. Kun kaikki kirjaimet ovat samoja, laukaistaan **Jackpot**-tapahtuma.

2. Käännä OneArmedBandit ActiveX-kontrolli painamalla F7. Kääntäjä ja linkkeri tekevät .ocx-päätteisen DLL:n output-hakemistoosi. Jos käännös onnistuu, kontrolli rekisteröidään tietokoneellesi.

Kontrollin testaaminen

ActiveX Control Test Container on hyödyllinen Visual Studion mukana toimitettava työkalu, jonka avulla voit ajaa ja testata koneellesi rekisteröityjä ActiveX-komponentteja. Seuraavissa harjoituksissa käytetään ActiveX Control Test Containeria OneArmedBandit ActiveX-kontrollin ominaisuussivujen, tapahtumien ja **Play()**-metodin toimivuuden kokeilemiseen.

► **OneArmedBandit-kontrollin kokeileminen**

1. Valitse Visual C++:n **Tools**-valikosta **ActiveX Control Test Container**. Valitse Test Containerin **Edit**-valikosta **Insert New Control**.
2. Valitse **Insert Control** -dialogista **OneArmedBandit Control** ja napauta **OK**.

3. Muuta kontrollin koko sopivaksi käyttämällä kontrollin reunoilla olevia kahvoja.
4. Napauta **Edit**-valikosta **Properties**. Muuta **OneArmedBandit Control Properties** -dialogin **General**-välilehdellä **Number of symbols** -ominaisuuden arvoksi **5**.
5. Napauta **Colors**-välilehteä ja valitse kontrollin **BackColor** (taustaväri) ja **ForeColor** (edustan väri) arvot.
6. Napauta **OK** ja varmista, että kontrollin värit ovat oikeat.
7. Napauta kontrollia ja tarkista, että **Click**-tapahtuma huomiodaan Test Container -ikkunan alareunassa.
8. Valitse **Control**-valikosta **Invoke Methods**. Siirrä tarvittaessa **Invoke Methods** -dialogia niin, että näet kontrollin kokonaan ja tulosteet Test Container -ikkunan alareunassa.
9. **Play (Method)** olessa valittuna **Method Name** -alasvetovalikosta napauta **Invoke**, jolloin pääset pelaamaan peliä. Pelaa kunnes saat kolme samaa symbolia. Kun näin käy, pitäisi **Jackpot**-tapahtuman näkyä Test Container -ikkunan alareunassa.

► **Ominaisuuksien säilyvyyden varmistaminen**

1. Sulje ActiveX Control Test Container. Tallenna kehotettaessa istunto nimellä **oab.tcs**.
2. Avaa ActiveX Control Test Container uudelleen. Valitse **File**-valikon viimeksikäytettyjen tiedostojen luettelosta **oab.tcs**. OneArmedBandit-kontrollin pitäisi avautua sen kokoisena ja värisenä kuin se on tallennettu. (**COleControl**-luokka huolehtii kontrollin mittasuhteiden serialisoinnista ilman ohjelmoijan toimenpiteitä.)
3. Valitse kontrolli napauttamalla sen reunusta. Varmista kontrollin ominaisuuksia tutkimalla, että **NumberOfSymbols**-ominaisuuden arvo on edelleen **5**.
4. Sulje ActiveX Control Test Container.

Oppitunnin yhteenveto

Tällä oppitunnilla näit, kuinka helppoa ActiveX-kontrollin tekeminen on MFC:tä käyttäen. ActiveX ControlWizard luo joukon luokkia, jotka toteuttavat DLL:n, kontrollin ja ominaisuussivun. ControlWizard määrittelee erilliset rajapinnat kontrollin ominaisuuksille, metodeille ja tapahtumille. Nämä rajapinnat suunnitellaan ClassWizardilla.

MFC ActiveX -kontrollit perustuvat **COleControl**-luokkaan, joka sisältää joukon jäsenfunktioita, jotka toteuttavat perusominaisuudet ja -tapahtumat, hakevat ulkoamäärätyt ominaisuudet säilöltä, toteuttavat ominaisuuksien tallentamisen ja yhdistävät kontrollin ominaisuudet ominaisuussivulle. Riippumatta mukautettujen metodien toteutuksesta suurin osa tarvittavasta lisäkoodista on kontrollin piirtokoodia, joka sijoitetaan kontrollin **OnDraw()**-funktioon.

Oppitunti 2: ActiveX-kontrollin tekeminen ATL:llä

Tällä oppitunnilla tehdään OneArmedBandit ActiveX-kontrolli uudelleen käyttämällä ATL:ää. Näin voit vertailla MFC ja ATL -kontrolliohjelmoinnin eroja ja pystyt päättämään, kumpi tekniikoista sopii paremmin tarpeisiisi.

Tämän oppitunnin jälkeen:

- Tiedät, kuinka ActiveX-kontrolli ohjelmoidaan ATL:llä.
- Tiedät, kuinka kontrollille lisätään ominaisuuksia ja metodeja.
- Tiedät, kuinka kontrollille tehdään ominaisuusikkuna ja kuinka kontrollin ominaisuudet saadaan tallennettua.
- Tiedät, kuinka ATL:n muunnosta COM:n yhteyspistearkkitehtuurista käytetään tapahtumien liittämiseen kontrolliin.

Oppitunnin arvioitu kesto: 50 minuuttia

Kontrollin lisääminen ATL COM-projektiin

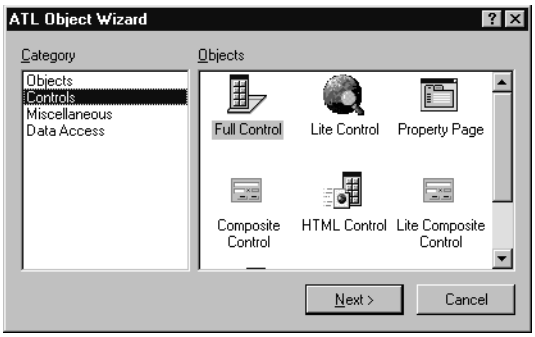
Kuten aina ATL COM-ohjelmoinnissa, luodaan ensin ATL-projekti ja lisätään sitten yksittäiset COM-objektit ATL:n Object Wizardilla. Seuraavissa harjoituksissa luodaan kontrollia varten DLL-projekti ja lisätään sitten ATL ActiveX-kontrolli.

► **OneArmedBanditATL-projektin luominen**

1. Valitse **File**-valikosta **New**. Valitse **Projects**-välilehdeltä **ATL COM AppWizard**.
2. Kirjoita **Project name** -ruutuun **OneArmedBanditATL** ja napauta **OK**.
3. Varmista **ATL COM AppWizard** dialogista, että **Dynamic Link Library (DLL)** on valittu ja napauta **Finish**.
4. Tutki uudelleen **New Project Information** -dialogi ja napauta **OK**.

► **ATLBandit-kontrollin lisääminen**

1. Valitse **Insert**-valikosta **New ATL Object**.
2. Valitse **Category**-luettelosta **Controls**. **ATL Object Wizardin** pitäisi avautua kuten kuvassa 11.5 seuraavalla sivulla.

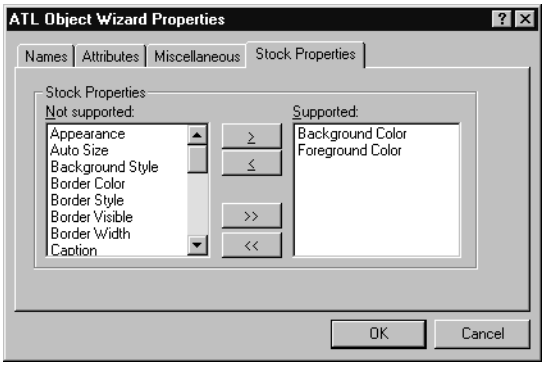


Kuva 11.5 ATL Object Wizard

ATL Object Wizard osaa näyttää useita erityyppisiä kontrolleja, kuten:

- **Full controls** Kontrollit, jotka voidaan sijoittaa mihin tahansa ActiveX:ää tukevaan säilöön.
- **Lite controls** Kontrollit, jotka voidaan sijoittaa Internet Exploreriin, mutta joista puuttuvat monien muiden säilöjen vaatimat rajapinnat.
- **Composite controls** Kontrollit, jotka voivat sisältää muita ActiveX-kontrolleja. Tällaisia kontrolleja ovat esimerkiksi dialogit.
- **HTML controls** Kontrollit, joissa on HTML-sivujen näyttämistä varten upotettu Web-selainkontrolli. Näitä kontrolleja käsitellään tarkemmin luvussa 12.

3. Valitse **Objects**-luettelosta **Full Control**. Napauta **Next**.
4. Napauta **Names**-välilehteä. Kirjoita **Short Name** -ruutuun **ATLBandit**.
5. Valitse **Attributes**-välilehdeltä **Support Connection Points** -vaihtoehto.
6. Siirrä **Stock Properties** -välilehdellä **>**-painiketta käyttämällä **Background Color** ja **Foreground Color** -perusominaisuudet **Not Supported** -luettelosta **Supported**-luetteloon, kuten kuvassa 11.6.



Kuva 11.6 Perusominaisuuksien valinta ATL Object Wizardissa

7. Lisää **ATLBandit** ActiveX-kontrolli projektiin napauttamalla **OK**.

ClassViewissä nähdään, että wizard on lisännyt **IATLBandit**-rajapinnan ja **CATLBandit**-toteutusluokan. **CATLBandit**-luokka on periytetty ATL:n **CComControl**-luokasta (samoin kuin moni muukin luokka). Avaa ATLBandit.h-tiedosto ja etsi COM-karttarakenne. Näet, että **CATLBandit**-luokka tukee monia ActiveX-kontrollisäilöjen vaatimia rajapintoja.

Heti COM-kartan jälkeen näet seuraavan ominaisuussivurakenteen:

```
BEGIN_PROP_MAP(CATLBandit)
    PROP_DATA_ENTRY("_cx", m_sizeExtent.cx, VT_UI4)
    PROP_DATA_ENTRY("_cy", m_sizeExtent.cy, VT_UI4)
    PROP_ENTRY("BackColor", DISPID_BACKCOLOR, CLSID_StockColorPage)
    PROP_ENTRY("ForeColor", DISPID_FORECOLOR, CLSID_StockColorPage)
    // Example entries
    // PROP_ENTRY("Property Description", dispid, clsid)
    // PROP_PAGE(CLSID_StockColorPage)
END_PROP_MAP()
```

ATL luo ominaisuuskarttarakenteen, jotta ominaisuuksien tallentaminen olisi helpommin toteutettavissa. Edellisestä koodista nähdään, että perusominaisuuksia, myös kontrollin mittasuhteita, koskevat merkinnät on jo tehty. Huomaa, että **PROP_ENTRY**-makrojen avulla on mahdollista liittää ominaisuussivu ominaisuuteen, ja että väriominaisuussivu on jo liitetty **ForeColor** ja **BackColor** ominaisuuksiin.

Ominaisuuksien lisääminen

Seuraavaksi lisätään luokkaan mukautettu **NumberOfSymbols**-ominaisuus.

► **NumberOfSymbols-ominaisuuden lisääminen**

1. Napauta ClassViewissä hiiren kakkospainikkeella **IATLBandit**. Napauta **Add Property**.
2. Valitse **Add Property To Interface** -dialogissa ominaisuustyypiksi **short**. Kirjoita **Property Name** -ruutuun **NumberOfSymbols**.
3. Luo ominaisuuden toteuttava funktio napauttamalla **OK**.

Muista, että ATL:ssä täytyy määritellä ominaisuuden arvon sisältävä jäsenmuuttuja sekä tehdä **Get** ja **Put** -funktiot, joiden avulla tietoja haetaan ja viedään jäsenmuuttujiin.

► **NumberOfSymbols-ominaisuuden toteutus**

1. Lisää suojattu **short**-tyyppinen jäsenmuuttuja **m_numberOfSymbols** **CATLBandit-**luokkaan.
2. Avaa ClassViewissä **IATLBandit-**rajapinta **CATLBandit-**luokan alta ja etsi **get_NumberOfSymbols()** ja **put_NumberOfSymbols()** -funktiot. Tee funktiot seuraavassa koodissa esitellyllä tavalla:

```
STDMETHODIMP CATLBandit::get_NumberOfSymbols(short *pVal)
{
    *pVal = m_numberOfSymbols;

    return S_OK;
}

STDMETHODIMP CATLBandit::put_NumberOfSymbols(short newVal)
{
    newVal = newVal < 3 ? 3 : newVal;
    newVal = newVal > 7 ? 7 : newVal;

    m_numberOfSymbols = newVal;
    SetDirty(TRUE);

    return S_OK;
}
```

Huomaa, että **put_ NumberOfSymbols()**-funktio tarkastaa, että **NumberOfSymbols**-ominaisuuteen asetetaan käypä arvo. Huomaa myös **SetDirty()-**funktion kutsu. Tätä funktiota tulee kutsua talletettavan ominaisuuden muuttuessa, jolloin säilö voi kysyä käyttäjältä, talletetaanko muutos vai hylätäänkö kontrollin tilaan tehdyt muutokset. **NumberOfSymbols**-ominaisuuden tallennustoiminto tehdään sen jälkeen, kun on ensin lisätty ominaisuussivu, joka mahdollistaa ominaisuuden arvon muuttamisen.

Tapahtumien lisääminen

Seuraavaksi toteutetaan kontrollin **Click** ja **Jackpot** tapahtumat. COM-tapahtumien ymmärtäminen on helpompaa, jos avaat projektin IDL-tiedoston, OneArmedBanditATL.idl, ja tutkit tyyppikirjaston määrittelyjä, jotka näet seuraavalla sivulla.

```
[
    uuid(39623002-6FAA-11D3-935D-0080C7FA0C3E),
    version(1.0),
    helpstring("OneArmedBanditATL 1.0 Type Library")
]
library ONEARMEDBANDITATLLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");

    [
        uuid(39623010-6FAA-11D3-935D-0080C7FA0C3E),
        helpstring("_IATLBanditEvents Interface")
    ]
    dispinterface _IATLBanditEvents
    {
        properties:
        methods:
    };

    [
        uuid(3962300F-6FAA-11D3-935D-0080C7FA0C3E),
        helpstring("ATLBandit Class")
    ]
    coclass ATLBandit
    {
        [default] interface IATLBandit;
        [default, source] dispinterface _IATLBanditEvents;
    };
};
```

ONEARMEDBANDITATLLib-tyyppikirjaston määrittely sisältää erillisen **_IATLBanditEvents**-rajapinnan määrittelyn. Tämä rajapinta määritellään **ATLBandit** coclass-lohkon sisällä — osoittaen, että ATLBandit-kontrolli julkistaa **_IATLBanditEvents**-rajapinnan. Huomaa kuitenkin, että rajapinnan määrittelyssä on mukana IDL-attribuutti ***[source]***, joka määrittelee, että **_IATLBanditEvents**-rajapinta on tapahtumien lähde. Tällaista rajapintaa kutsutaan *yhteyspisteeksi* (connection point). COM-objekti, joka julkistaa yhteyspisteen, toteuttaa **IConnectionPointContainer**-rajapinnan, joka huolehtii lähderajapintojen yhteykisistä vastaavaan asiakasobjektiin, josta käytetään nimitystä *sink*. Sink-objekti toteuttaa lähdeobjektin määrittelemät toiminnot. Sink-objektin rajapinnan osoitin välitetään lähdeobjektille yhteyspistemekanismin kautta. Tämän osoittimen avulla lähde voi muodostaa yhteyden metodiensa toteutuksiin. Laukaistessaan tapahtuman lähdeobjekti kutsuu vastaavaa sink-rajapinnan metodia.

Ensimmäinen tehtävä tapahtumaa määriteltäessä on tapahtumaa edustavan metodin lisääminen lähderajapintaan.

► **Click() ja Jackpot() -tapahtumien metodien lisääminen**

1. Napauta ClassViewissä hiiren kakkospainikkeella **_IATLBanditEvents**. Valitse pikavalikosta **Add Method**.
2. Valitse **void** paluuarvon tyypiksi. Anna metodin nimeksi **Click**.
3. Luo samalla tavalla **Jackpot**-niminen **void**-metodi.

Kun metodit on lisätty lähderajapintaan, käytetään ATL Wizardia yhteyspisteen tekemiseen. ATL käyttää tyyppikirjaston tietoja tämän toteutuksen muodostamisessa, joten tyyppikirjasto täytyy ensin kääntää.

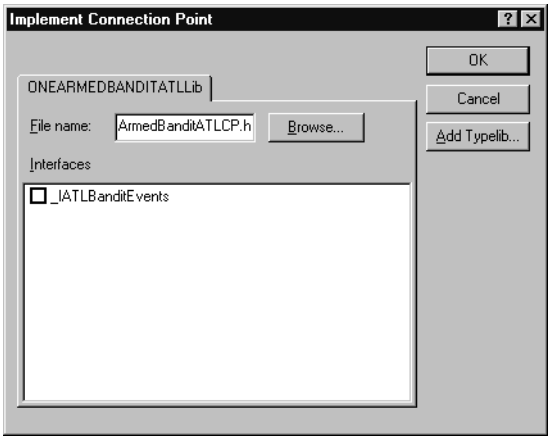
► **Tyyppikirjaston kääntäminen**

1. Siirry FileViewiin ja etsi OneArmedBanditATL.idl-tiedosto.
2. Napauta hiiren kakkospainikkeella **OneArmedBanditATL.idl** ja valitse sitten **Compile OneArmedBanditATL.idl**.
3. Kun käännös on suoritettu palaa ClassViewiin.

Nyt voidaan yhteyspiste toteuttaa ATL Wizardilla.

► **ATLBandit-kontrollin yhteyspisteen tekeminen**

1. Napauta hiiren kakkospainikkeella **CATLBandit**-luokkaa ja valitse **Implement Connection Point**. Implement Connection Point Wizard avautuu kuten kuvassa 11.7.



Kuva 11.7 Implement Connection Point Wizard

2. Valitse Implement Connection Point Wizardissa **_IATLBanditEvents**-valintaruutu ja napauta **OK**.

Wizard luo *proxy-luokan* (jolla ei ole mitään tekemistä parametrien järjestelyssä käytettävien proxy-objektien kanssa), joka sisältää proxyfunktiot, jotka toteuttavat tapahtumametodit, jotka lisättiin tapahtumarajapintaan. Jos katsot ClassViewiin, näet, että projektiisi on lisätty **CProxy_IATLBanditEvents**-luokka, ja että se sisältää funktiot **Fire_Click()** ja **Fire_Jackpot()**, joiden avulla kontrollin tapahtumat laukaistaan. ATL muuttaa **CATLBandit**-luokan toteutusta niin, että se periytyy **CProxy_IATLBanditEvents class**-luokasta. Näin kontrolliluokkasi jäsenet pystyvät käsittelemään **Fire_Click()** ja **Fire_Jackpot()** metodeja.

Implement Connection Point Wizard luo myös seuraavan yhteyspistekartan luokkaasi.

```
BEGIN_CONNECTION_POINT_MAP(CATLBandit)
    CONNECTION_POINT_ENTRY(IID_IPropertyNotifySink)
    CONNECTION_POINT_ENTRY(DIID__IATLBanditEvents)
END_CONNECTION_POINT_MAP()
```

Varmista, että yhteyspistekartta näyttää täsmälleen samanlaiselta kuin mallissa —ATL ei aina pysty muodostamaan tätä karttaa oikein, mistä aiheutuu käännösvirheitä.

Voit nyt käyttää tapahtumien proxyfunktioita kontrolliesi tapahtumien laukaisemiseen. Tämän havainnollistamiseksi toteutetaan käsittelijä, joka laukaisee **Click()**-tapahtuman, kun kontrollia napautetaan.

► **OnLButtonDown()-käsittelijän lisääminen**

1. Napauta ClassViewissä hiiren kakkospainikkeella **CATLBandit**-luokkaa. Valitse pikavalikosta **Add Windows Message Handler**.
2. Valitse **New Windows messages/events** -luettelosta **WM_LBUTTONDOWN**.
3. Lisää **OnLButtonDown()**-funktio napauttamalla **Add and Edit**. Samalla siirryt funktion toteuttavaan koodiin, joka on määritelty **CATLBandit**-luokan määrittelyssä. Huomaa, että ATL ylläpitää omanlaistaan sanomakartaa, joka on sijoitettu yhteyspistekartan alle.
4. Korvaa **OnLButtonDown()**-funktion sisällä oleva //TODO-kommentti seuraavalla koodirivillä:

```
Fire_Click();
```

Metodien lisääminen

Seuraavaksi lisätään kontrollin rajapintaan **Play()**-metodi ja lisätään metodin toteuttamiseen tarvittava koodi.

► **Play()-metodin lisääminen**

1. Napauta ClassViewissä hiiren kakkospainikkeella **IATLBandit**-rajapintaa. Valitse pikavalikosta **Add Method**.
2. Kirjoita **Add Method to Interface** -dialogissa **Method Name** -ruutuun **Play**.
3. Luo metodi napauttamalla **OK**.

► **Play()-metodin toteuttaminen**

1. Samoin kuin teit MFC OneArmedBandit-kontrollille oppitunnilla 1, lisää seuraava suojattu jäsenmuuttuja **CATLBandit**-luokkaan:

```
TCHAR m_symbols[3];
```

Lisää seuraava alustuskomento CATLBandit-muodostimeen:

```
_tcscpy(m_symbols, _T("JJJ"));
```

2. Avaa ClassViewissä **IATLBandit**-rajapinta **CATLBandit**-luokan alta.
3. Kaksoisnapauta **Play()**-metodia **IATLBandit**-rajapinnan alta, jolloin pääset muokkaamaan **CATLBandit::Play()**-funktiota.
4. Lisää tarvittava koodi **CATLBandit::Play()**-funktion runkoon niin, että se näyttää seuraavalta:

(Tämä koodi on asennettu oheisrompulta tiedostoon CH11_03.cpp.)

```
STDMETHODIMP CATLBandit::Play()
{
    srand((unsigned)time(NULL));

    _tcscpy(m_symbols, _T("JJJ"));

    for(int i = 0; i < 3; i++)
        m_symbols[i] += UINT(rand() % m_numberOfSymbols);

    // repaint control
    m_spInPlaceSite->InvalidateRect(NULL, TRUE);

    if(m_symbols[0] == m_symbols[1] &&
        m_symbols[1] == m_symbols[2])
        Fire_Jackpot();

    return S_OK;
}
```

5. Lisää seuraava rivi ATLBandit.cpp-tiedoston alkuun:

```
#include <time.h>
```

Ominaisuussivun luominen

Seuraavaksi tehdään **Symbols**-ominaisuussivu, jonka avulla käyttäjä voi asettaa **NumberOfSymbols**-ominaisuuden arvon. ATL asettaa kontrollin ominaisuusikkunan jokaisen sivun erillisiksi objekteiksi, joiden toteutuksesta huolehtii **IPropertyPage**-luokasta periytetty luokka. Seuraavassa harjoituksessa projektiin lisätään uusi ominaisuusikkuna luokka.

► **Symbols-ominaisuussivun lisääminen**

1. Valitse **Insert**-valikosta **New ATL Object**. Valitse **Category**-luettelosta **Controls**.
2. Valitse **Objects**-luettelosta **Property Page**. Napauta **Next**.
3. Napauta **Names**-välilehteä. Kirjoita **Short Name** -ruutuun **BanditPP**.
4. Kirjoita **Strings**-välilehden **Title**-ruutuun **Symbols**. Poista tekstit kahdesta muusta ruudusta.
5. Lisää ominaisuusikkuna napauttamalla **OK**. Sivun dialogimalli avautuu dialogieditoriin.
6. Tee dialogimalli aivan kuin MFC-kontrollillekin (palaa kuvaan 11.4). Lisää selitetekstikontrolli ja numeerinen muokkausruutu, jonka tunnisteeksi tulee **IDC_NUMSYMBOLS**.

Jos siirryt takaisin ClassViewiin, näet, että **CBanditPP**-luokka on lisätty projektiin. ATL Object Wizard on tehnyt **IPropertyPageImpl::Apply()**-funktion tähän luokkaan. **Apply()** suoritetaan ominaisuusikkunan käyttäjän napautettua **OK** tai **Apply** -painiketta. Seuraavaksi lisätään tämän funktion toteuttamiseen tarvittava koodi.

► **CBanditPP::Apply()-funktion toteutus**

Etsi **CBanditPP::Apply()**-funktio BanditPP.h-tiedostosta. Korvaa esimerkkikoodi seuraavalla toteutuksella:

```
STDMETHOD(Apply)(void)
{
    CComQIPtr<IATLBandit> pIBandit(m_ppUnk[0]);

    pIBandit->put_NumberOfSymbols(GetDlgItemInt(IDC_NUMSYMBOLS));

    m_bDirty = FALSE;
    return S_OK;
}
```

Tämä funktio asettaa kontrollin **NumberOfProperties**-ominaisuudeksi muokkausruudusta saadun numeroarvon. Huomaa kuinka ATL:n osoitinluokkaa **CComQIPtr** käytetään **IATLBandit**-osoittimen hakemiseen. Jäsenmuuttuja m_ppUnk on **IUnknown**-osoittimista muodostuva taulukko, jossa olevat osoittimet viittaavat ominaisuussivuun yhdistettyihin objekteihin.

Apply()-funktio tallettaa muokkausruutukontrollin arvon pysyvään **IATLBandit: :NumberOfSymbols**-ominaisuuteen. Muokkausruutukontrollin alustamiseen tarvittava funktio täytyy tehdä myös. Tämä funktio on oikeastaan **Apply()**-funktio takaperin.

► **CBanditPP::OnInitDialog()-funktion toteutus**

1. Napauta ClassWizardissa hiiren kakkospainikkeella **CBanditPP**-luokkaa. Valitse pikavalikosta **Add Windows Message Handler**.
2. Valitse **New Windows messages/events**-luettelosta **WM_INITDIALOG**-sanoma.
3. Lisää **OnInitDialog()**-dialogi napauttamalla **Add and Edit**. Siirryt samalla funktion toteutukseen.
4. Tee funktion toteutus seuraavassa koodissa esitetyllä tavalla:

```
LRESULT OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM lParam,
    BOOL& bHandled)
{
    CComQIPtr<IATLBandit> pIBandit(m_ppUnk[0]);

    short i = 0;
    pIBandit->get_NumberOfSymbols(&i);
    SetDlgItemInt(IDC_NUMSYMBOLS, i);
    return 0;
}
```

Tässä vaiheessa täytyy lisätä ilmoitusfunktio, joka ilmaisee, että ominaisuussivua on muokattu. Tämän funktion tulee käsitellä **EN_CHANGE**-tapahtuma, joka laukeaa käyttäjän muuttaessa muokkausruudussa olevaa arvoa.

► **CBanditPP::OnChangeNumsymbols()-funktion toteutus**

1. Napauta ClassWizardissa hiiren kakkospainikkeella **CBanditPP**-luokkaa. Valitse pikavalikosta **Add Windows Message Handler**.
2. Valitse **Class or object to handle** -luettelosta **IDC_NUMSYMBOLS**-objekti.
3. Valitse **New Windows messages/events**-luettelosta **EN_CHANGE**-tapahtuma.
4. Napauta **Add and Edit** ja sitten **OK**. Näin lisätään **OnChangeNumsymbols()**-funktio ja siirrytään samalla sen toteuttavaan koodiin.

5. Korvaa //TODO-kommentti seuraavalla koodirivillä:

```
SetDirty(TRUE);
```

Nyt, kun ominaisuussivu on kokonaisuudessaan valmis, voidaan lisätä koodi, joka tallentaa **NumberOfSymbols**-ominaisuuden arvon.

► **NumberOfSymbols-ominaisuuden tekeminen pysyväksi**

1. Etsi ominaisuuskartasta ATLBandit.h-tiedosto.
2. Lisää uusi **NumberOfSymbols**-ominaisuutta koskeva merkintä niin, että ominaisuuskartta näyttää lopulta seuraavalta:

```
BEGIN_PROP_MAP(CATLBandit)
    PROP_DATA_ENTRY("_cx", m_sizeExtent.cx, VT_UI4)
    PROP_DATA_ENTRY("_cy", m_sizeExtent.cy, VT_UI4)
    PROP_ENTRY("NumberOfSymbols", 1, CLSID_BanditPP)
    PROP_ENTRY("BackColor", DISPID_BACKCOLOR, CLSID_StockColorPage)
    PROP_ENTRY("ForeColor", DISPID_FORECOLOR, CLSID_StockColorPage)
END_PROP_MAP()
```

PROP_ENTRY-makron toinen parametri on *DISPID*, joka voidaan löytää IDL-tiedostosta. **CLSID_BanditPP**-vakio on määritelty OneArmedBanditATL_i.c-tiedostossa.

Kontrollin piirtäminen

Jäljellä on vain piirtokoodin toteutus. Kuten MFC:n **COleControl**-luokassakin, **CComControl**-luokka sisältää **OnDraw()**-funktion, joka sisältää piirtokoodin. **CComControl::OnDraw()**-funktio saa viittauksen **ATL_DRAWINFO**-rakenteeseen, joka sisältää, monien muiden asioiden lisäksi, piirtopinnan kahvan ja osoittimen **RECT**-rakenteeseen, joka osoittaa kontrollin suorakaiteen reunat. Piirtopinta sisältyy **HDC**:hen, joka on MFC-luokkaan **CDC**-kiedottu Windowsin alkuperäinen tietotyyppi. Tämä tarkoittaa sitä, että kontrollin muodostamiseen täytyy käyttää GDI API -funktioita. GDI-funktiot ovat samanlaisisa kuin vastaavat **CDC**-funktiot, tärkein ero on, että ne ottavat ensimmäiseksi argumentikseen **HDC**:n. Jos vertaat **CATLBandit::OnDraw()**-funktiota ja edellisen oppitunnin **COneArmedBanditCtrl::OnDraw()**-funktioita keskenään, saat hyvän käsityk-sen siitä, kuinka tämä kaikki toimii.

► **CATLBandit::OnDraw()-funktion toteutus**

Korvaa ATLBandit.h-tiedostossa oleva funktio seuraavalla versiolla:

(Tämä koodi on asennettu oheisrompulta tiedostoon CH11_04.cpp.)

```
HRESULT OnDraw(ATL_DRAWINFO& di)
{
    const RECT& rc = *reinterpret_cast<const RECT*>(di.prcBounds);
    HDC dc = di.hdcDraw;
```

```
COLORREF colorBack, colorFore;
OleTranslateColor(m_clrForeColor, NULL, &colorFore);
OleTranslateColor(m_clrBackColor, NULL, &colorBack);

// Get dimensions of control
float ctrlWidth = float(rc.right - rc.left);
float ctrlHeight = float(rc.bottom - rc.top);

// Set up DC
HBRUSH brush = CreateSolidBrush(colorBack);
HBRUSH oldBrush = static_cast<HBRUSH>(SelectObject(dc, brush));

HPEN pen = CreatePen(PS_SOLID, 3, colorFore);
HPEN oldPen = static_cast<HPEN>(SelectObject(dc, pen));

HFONT SymbolFont = CreateFont(long(ctrlHeight / 1.1),
    long(ctrlWidth/6), 0, 0, 0, 0, 0, 0, SYMBOL_CHARSET, 0, 0, 0,
    0,"WingDings");

HFONT oldFont = static_cast<HFONT>(SelectObject(dc, SymbolFont));

// Draw bounding rectangle
Rectangle(dc, rc.left, rc.top, rc.right, rc.bottom);
SetBkMode(dc, TRANSPARENT);

// Draw text
SetTextColor(dc, colorFore);
RECT rect;
CopyRect(&rect, &rc);

TCHAR strDisplay[5];
_stprintf(strDisplay, "%c %c %c",
    m_symbols[0], m_symbols[1], m_symbols[2]);

DrawText(dc, strDisplay, 5, &rect, DT_SINGLELINE | DT_CENTER |
    DT_VCENTER );

// Draw vertical bars
long onethird = long(ctrlWidth / 3);
POINT ptTop = { rc.left, rc.top };
POINT ptBtm = { rc.left, rc.bottom };

ptTop.x += onethird; ptBtm.x += onethird;
MoveToEx(dc, ptTop.x, ptTop.y, NULL);
LineTo(dc, ptBtm.x, ptBtm.y);

ptTop.x += onethird; ptBtm.x += onethird;
MoveToEx(dc, ptTop.x, ptTop.y, NULL);
LineTo(dc, ptBtm.x, ptBtm.y);
```

```
    // Restore device context
    SelectObject(dc, oldFont);
    SelectObject(dc, oldPen);
    SelectObject(dc, oldBrush);

    DeleteObject(brush);
    DeleteObject(pen);
    DeleteObject(SymbolFont);

    return S_OK;
}
```

Voit nyt kääntää **ATLBandit**-kontrollin. Kokeile kontrollin näyttöä, ominaisuussivuja, **Play()**-metodia, sekä **Click** ja **Jackpot** -tapahtumia lataamalla kontrolli ActiveX Control Test Containeriin.

MFC vai ATL?

Sinulla pitäisi nyt olla hyvä kuva MFC:n ja ATL:n eroavaisuuksista ActiveX-kontrolleja ohjelmoitaessa. MFC tekee kontrollin ohjelmoinnista helpon toimenpiteen kätkemällä lähes kaikki COM-ohjelmoinnin monimutkaisuudet. Vaikka kontrollien ohjelmoiminen ATL:llä vaatiikin enemmän työtä, opit COM-ohjelmointitaitojesi karttuessa todennäköisesti arvostamaan sen antamia vaikutusmahdollisuuksia.

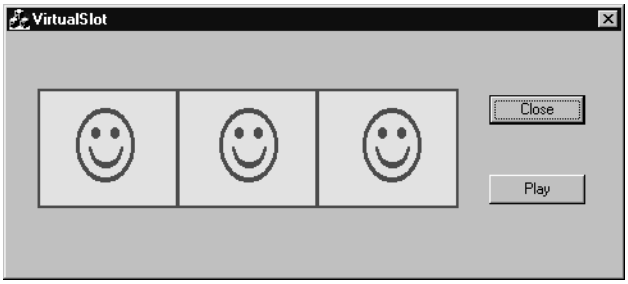
ATL ActiveX-kontrollit ovat pienempiä kuin MFC-vastineensa ja oikein ohjelmoituina toimivat paremmin. ATLBandit-projektin itsenäinen (MinDependency) käännös tuottaa kontrollin, jonka koko on 100 Kt. Staattisesti MFC-kirjastoihin linkitetty OneArmedBandit.ocx on kooltaan noin 264 Kt. Kun komponentti linkitetään MFC-kirjastoihin dynaamisesti, koko putoaa noin 36 kilotavuun, jolloin MFC:stä tulee houkutteleva vaihtoehto, mikäli kohdeympäristössä voidaan olettaa MFC:n DLL:ien olevan asennettuina.

Oppitunnin yhteenveto

ATL:ää voidaan käyttää pienten, nopeiden ja tehokkaiden kaikissa säilöissä toimivien ActiveX-kontrollien ohjelmoimiseen. ATL:ssä ActiveX-kontrollit toteuttaa **CComControl**-luokasta periytetty luokka. ActiveX-kontrollit toteuttavat myös monia ActiveX-kontrollisäilöjen vaatimia COM-rajapintoja. ATL-kontrolliluokka sisältää ominaisuuskarttarakenteen, joka toteuttaa ominaisuuksien tallettamisen, sanomakartan, joka käsittelee Windowsin sanomat, COM-yhteyspisteen, joka huolehtii ActiveX-tapahtumista. Voit toteuttaa kontrollin ominaisuussivun sijoittamalla ominaisuussivun COM-objektin projektiisi. Kontrolli piirretään käyttämällä GDI API -funktioita **CComControl::OnDraw()**-funktiossa.

Laboratorio 11: ActiveX-kontrollin käyttäminen sovelluksessa

Laboratoriossa 7 näit, kuinka STUpload-sovellus käytti kahta ActiveX-kontrollia — Microsoft ADO Data Control ja Microsoft DataGrid Control — osana käyttöliittymäänsä. STUpload-sovellukseen ei lisätä enää uusia ActiveX-kontrolleja. Sen sijaan opit, kuinka tässä luvussa luomaasi OneArmedBandit-kontrollia käytetään osana Windows-sovellusta. Tämä VirtualSlot niminen sovellus on kuvassa 11.8.



Kuva 11.8 VirtualSlot-sovellus

Sovellus kutsuu OneArmedBandit-kontrollin julkistamaa **Play()**-metodia ja käsittelee **Jackpot**-tapahtuman. Sovelluksen kehitystyön aikana asetetaan ominaisuussivun avulla käytettävien symbolien määrä ja kontrollin värit.

Jos et ole tehnyt tämän luvun harjoituksia, voit käyttää kansiossa Chapter 11\Lab olevaa OneArmedBandit-kontrollia. Tämän kontrollin rekisteröinti tapahtuu seuraavasti:

1. Kopioi OneArmedBandit.ocx-tiedosto sopivaan paikkaan kiintolevyllesi.
2. Avaa **Käynnistä**-valikosta komentokehote ja siirry hakemistoon, jossa OneArmedBandit.ocx-tiedosto on.
3. Rekisteröi kontrolli koneellesi seuraavalla komennolla:

```
regsvr32 onearmedbandit.ocx
```

Nyt olet valmis aloittamaan VirtualSlot-sovelluksen tekemiseen.

► **VirtualSlot-projektin luominen**

1. Valitse Visual C++:n **File**-valikosta **New**.
2. Valitse **MFC AppWizard (exe)** uuden **VirtualSlot**-projektin luomista varten. Napauta **OK**.
3. Napauta AppWizardin vaiheessa 1 **Dialog Based** ja sitten **Finish**.

4. Vahvista valinta napauttamalla **OK**.

IDD_VIRTUALSLOT_DIALOG-dialogimalli avautuu dialogieditoriin. Luot dialogia muokkaamalla VirtualSlot-sovelluksen käyttöliittymän, joka koostuu OneArmedBandit-kontrollista, **Play**-painikkeesta ja **Close**-painikkeesta.

► **IDD_VIRTUALSLOT_DIALOG-dialogimallin muokkaaminen**

1. Poista //TODO-teksti.
2. Muuta **OK**-painikkeen otsikoksi **Close**. Yhdistä **Cancel**-painikkeeseen uusi tunniste **ID_PLAY**, ja muuta sen otsikoksi **Play**.
3. Järjestä painikkeet kuten kuvassa 11.8.

► **OneArmedBandit-kontrollin lisääminen projektiin**

1. Valitse **Project**-valikosta **Add to Project**, ja napauta sitten **Components and Controls**.
2. Kaksoisnapauta Components and Controls Galleryn **Registered ActiveX Controls** -kansiota, jolloin näet kaikki järjestelmääsi rekisteröidyt ActiveX-kontrollit.
3. Valitse OneArmedBandit Control ja napauta **Insert**.
4. Lisää komponentti napauttamalla **OK**. Vahvista **COneArmedBandit**-luokan luominen napauttamalla **OK**.
5. Sulje Components and Controls Gallery napauttamalla **Close**.

Huomaa, että OneArmedBandit-kontrollin oletus **OCX**-kuvake on lisätty **Controls**-työkaluriville. Tässä luvussa luomasi MFC OneArmedBandit -projekti sisältää **IDB_ONEARMEDBANDIT**-bittikartan, jota muokkaamalla kontrollille voidaan tehdä erilainen työkalupainike.

Näet ClassViewissä **COneArmedBandit**-luokan, jonka Components and Controls Gallery on luonut. Tämä luokka sisältää käärefunktiot, joiden avulla voidaan käsitellä kontrollin ominaisuuksia ja metodeja. **Create()**-funktion avulla voidaa luoda kontrollin ilmentymä dynaamisesti (esimerkiksi **CView**-objektissa).

► **COneArmedBandit-kontrollin ilmentymän sijoittaminen**

1. Napauta **Controls**-työkalurivin **OneArmedBandit**-painiketta ja sijoita kontrolli dialogiin. Muuta kontrollin ja dialogin kokoa niin, että lopputulos vastaa kuvaa 11.8.

2. Siirry muokkaamaan kontrollin ominaisuuksia painamalla ENTER OneArmedBandit-kontrollin ollessa valittuna. Tuttujen ominaisuussivujen lisäksi ilmestyvät näkyviin myös säilön luomat **General** ja **All** -sivut.
3. Muuta **Control**-sivulla **Number of Symbols**:n arvoksi **7**.
4. Aseta **Colors**-sivulla **ForeColor** ja **BackColor** -ominaisuuksien värit.

Seuraavaksi muokataan **CVirtualSlotDlg**-dialogiluokkaa niin, että se sisältää **Play**-painikkeen toiminnan ja käsittelee **Jackpot**-tapahtuman.

► **Play-painikkeen toteuttaminen**

1. Avaa ClassWizard painamalla CTRL+W. Valitse **Member Variables** -välilehdeltä **IDC_ONEARMEDBANDIT1**-kontrollitunniste. Luo **m_bandit**-niminen **COneArmedBandit**-tyyppinen **Control**-jäsenmuuttuja napauttamalla **Add Variable**.
2. Valitse **Message Maps** -sivulla **ID_PLAY**-objektitunniste. Lisää BN_CLICKED-sanomalle käsittelijä **OnPlay()**.
3. Napauta **Edit Code**. Korvaa **OnPlay()**-funktion toteutuksesta //TODO-kommentti seuraavalla koodirivillä:

```
m_bandit.Play();
```

► **Jackpot-tapahtuman käsittely**

1. Valitse ClassWizardin **Message Maps** -välilehdeltä kontrollitunniste **IDC_ONEARMEDBANDIT1**.
2. Valitse **Jackpot**-sanoma ja luo **OnJackpotOnearmedbanditctrl1()**-funktio napautamalla **Add Function**.
3. Napauta **Edit Code**. Korvaa funktion toteutuksesta //TODO-kommentti koodirivillä:

```
AfxMessageBox(“Jackpot!”);
```

4. Voit nyt kääntää ja käynnistää VirtualSlot-sovelluksen. Pelaa hedelmäpeliä napauttamalla **Play**-painiketta. Pelaa kunnes saat jackpotin!

Kertaus

1. Kuinka ominaisuuksien tallentaminen tehdään MFC ActiveX-kontrollissa?

2. Mikä on DDP-funktion tehtävä, ja missä niitä käytetään?

3. Mikä on yhteyspiste?

4. Kuinka **CComControl**-luokasta periytetty objekti käsittelee Windowsin sanomia?

5. Missä tilanteissa MFC soveltuu parhaiten käytettäväksi ActiveX-kontrollin tekemiseen?