L U K U 9

COM-komponenttien luominen

Tässä luvussa

Tässä luvussa kerrotaan, kuinka yksinkertainen COM-komponentti luodaan käyttämällä Microsoft ActiveX Template Librarya (ATL). Tutkit ATL-velhojen luoman COM-objektin koodiin liittyviä asioita ja tutustut muihin tapoihin, joilla voidaan luoda COM-komponentteja.

Ennen kuin aloitat

Ennen kuin aloitat tämän luvun läpikäymisen sinun tulisi lukea luvut 2-8 ja tehdä niihin liittyvät tehtävät.

Oppitunti 1: COM-komponenttien luominen ATL:llä

ATL on joukko C++-malliluokkia, joiden pohjalta voit luoda helposti COM-komponentteja. Se tukee erityisesti COM:n tärkeimpiä ominaisuuksia mukaan lukien **IUnknown** ja **IDispatch** rajapinnat ja kaksoisrajapinnat. ATL:ää voidaan käyttää myös luotaessa ActiveX-kontrolleja. ATL-koodin avulla voidaan luoda myös single-threaded, apartment-threaded, tai free-threaded-objekteja.

ATL:n lisäksi Microsoft Visual Studio sisältää ohjatun toiminnon, joka yksinkertaistaa ATL:n käyttämistä sovelluskehyksen osana.

Tällä oppitunnilla nähdään, kuinka ATL:llä luodaan yksinkertainen COM-komponentti.

Tämän oppitunnin jälkeen:

- Tiedät, kuinka ATL COM AppWizardia käytetään COM-palvelimen luomisessa COM-objekteille.
- Tiedät, kuinka ATL Object Wizardilla tehdään COM-objekti.
- Tiedät, kuinka ATL COM-objektille lisätään ominaisuus ja kuinka se toteutetaan.
- Tiedät, kuinka ATL COM-objektille lisätään metodi ja kuinka se toteuttetaan.

Oppitunnin arvioitu kesto: 30 minuuttia

ATL:n käyttö

COM-objekteja tehteässä voidaan hyödyntää suurta määrää valmista koodia. ATL:n avulla Visual Studio pystyy tarjoamaan helpon tavan luoda COM-objekteja. Joukko ATL-velhoja huolehtii valmiin koodin generoimisesta, jolloin ohjelmoijan vastuulle jää vain komponenttikohtaisen, komponentin varsinaisen tehtävän hoitavan koodin kirjoittaminen.

ATL-velhojen muodostama koodi perustuu ATL:n muodostaviin ydinluokkiin. Mallien käyttäminen (vastakohtana MFC:n käyttämälle syvälle perintärakenteelle) antaa ATL:lle mahdollisuuden komponentteihin ja kontrolleihin soveltuvan nopean, kevyen koodin tuottamiseen.

COM-objekti tehdään ATL:llä seuraavasti:

1. Luo ATL COM -projekti käyttämällä ATL COM AppWizardia. Luotavan projektin tyyppi määrittää COM-objekteja isännöivän palvelimen tyypin (prosessin sisäinen tai ulkoinen).
2. Lisää uusi ATL-objekti käyttämällä ATL Object Wizardia.
3. Lisää objektiin metodit käyttämällä Add Method to Interface Wizardia.
4. Lisää objektiin ominaisuudet käyttämällä Add Property to Interface Wizardia.
5. Tee objektin metodien toteutukset.

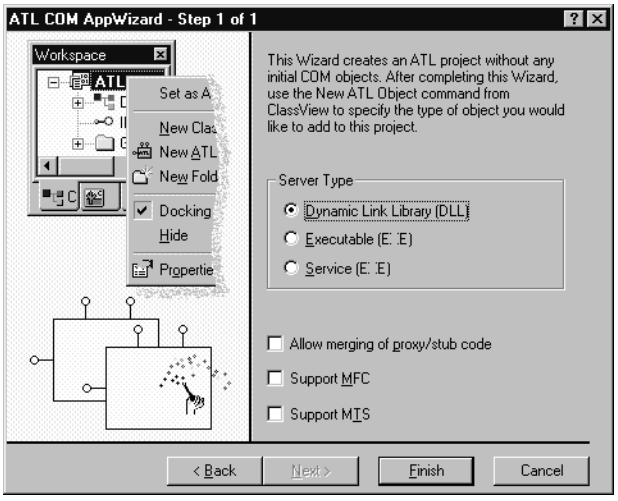
Tällä oppitunnilla teet näitä vaiheita seuraten yksinkertaisen ATL-pohjaisen COM-objektin, joka tarjoaa salakirjoituspalveluja. Tämä COM-objekti, nimeltään **Encoder**, tukee yhtä **IEncoder**-nimistä rajapintaa. Tämä rajapinta julkistaa yhden metodin, **EncodeString()**, joka palauttaa yksinkertaisella salausalgoritmilla salatun merkkijonon. Salaus lisää tai vähentää merkkijonon merkkiä vastaavaa numeroarvoa määrätyllä lukemalla. Tämä lukema määritellään ominaisuuden **Key** avulla.

ATL COM -projektin luominen

Tässä harjoituksessa tehdään ATL COM AppWizardia käyttäen ATL COM -projekti COM-palvelinta varten.

► **EncodeServer-projektin luominen**

1. Valitse **File**-valikosta **New**, napauta sitten **Projects**-välilehteä.
2. Projektikategoriat on lueteltu dialogin vasemmalla puolella olevassa ikkunassa. Valitse **ATL COM AppWizard**.
3. Kirjoita **Project name** -ruutuun **EncodeServer**. Napauta **OK**.
4. Varmista **ATL COM AppWizard** -dialogissa, että **Dynamic Link Library (DLL)** on valittu kuten kuvassa 9.1, ja napauta sitten **Finish**.

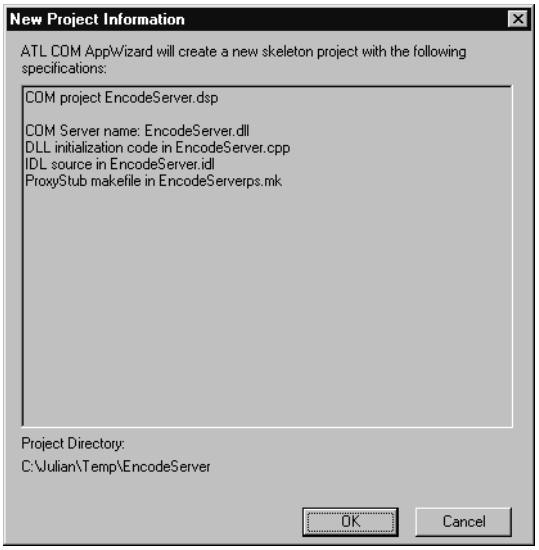


Kuva 9.1 **ATL COM AppWizard** -dialogi DLL:n luomista varten konfiguroituna

Jos päätät luoda DLL:n isännöimään COM-komponenttejasi, määrität samalla, että COM-palvelin on prosessin sisäinen. Jos valitset **Executable (EXE)**-vaihtoehdon, luodaan palvelin, joka voi olla prosessin ulkoinen tai etäpalvelin riippuen tavasta, jolla se toteutetaan. Nämä termit on määritelty luvun 8 oppitunnilla 2. **Service**-vaihtoehto antaa mahdollisuuden *Windows*

NT palvelun luomiseen. Palvelu on ohjelma, joka suoritetaan taustalla Windows NT:n käynnistyessä.

5. **New Project Information** -dialogi kertoo tietoja COM-projektia varten luoduista tiedostoista, kuten kuvassa 9.2. Jatka napautamalla **OK**.



Kuva 9.2 New Project Information -dialogi

Olet luonut DLL-isännän prosessinsisäiselle COM-palvelimelle, ja lisännyt kaiken COM-objektin rekisteröimiseen vaadittavan koodin. Tämäntyyppisestä COM-objektista käytetään nimitystä *itserekisteröityvä* (self-registering) komponentti.

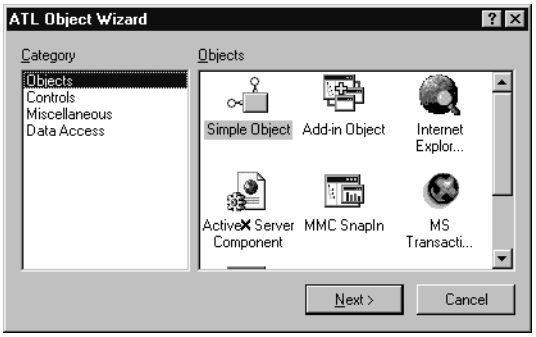
Uuden COM-komponentin lisääminen

Nyt, kun isäntäympäristö on luotu ja määritelty, voidaan lisätä varsinainen COM-objekti. Tämän tehtävän suorittamiseen käytetään ATL Object Wizardia.

► **Encoder COM-komponentin lisääminen**

1. Valitse **Insert**-valikosta **New ATL Object**.
2. Valitse **Category**-luettelosta **Objects**.

3. Valitse **Objects**-ruudusta **Simple Object** -kuvake kuten kuvassa 9.3 ja napauta **Next**.



Kuva 9.3 ATL Object Wizard -dialogi

4. Valitse **ATL Object Wizard Properties** -dialogin **Names**-välilehti.
5. Kijoita **Short Name** -ruutuun palvelinluokan nimi — **Encoder**. Kaikki muut kentät päivitetään automaattisesti **Short Name** -ruudun perusteella, kuten kuvassa 9.4 nähdään.

Kuva 9.4 ATL Object Wizard Properties -dialogi

Dialogin vasen osa kertoo, että velho luo **CEncoder**-nimisen C++-luokan (määritelty tiedostoissa Encoder.h ja Encoder.cpp). COM-komponentin määrittelemiseksi tarvittava koodi sijoitetaan tähän luokkaan. Oikealla oleva osa kertoo, että komponentin nimeksi tulee **Encoder,** ja että se julkistaa oletusrajapinnan **IEncoder**. Huomaa myös ProgID **EncodeServer.Encoder**. Tämän nimen avulla voit selvittää objektin GUID:n käyttämällä **CLSIDFromProgID()** -funktiota.

6. Valitse **ATL Object Wizard Properties** -dialogissa **Attributes**-välilehti. Aseta seuraavat kuvassa 9.5 nähtävät attribuutit:
   - Valitse **Threading Model** kohdasta **Single**.
   - Valitse **Interface** kohdasta **Custom**.
   - Valitse **Aggregation** kohdasta **No**.

Kuva 9.5 ATL Object Wizard Properties -dialogin **Attributes**-välilehti

ATL noudattaa suositusta sanomarajapintojen toteuttamisesta kaksoisrajapintoina. Valitsemalla mukautetun rajapinnan valitsit, että COM-komponenttiin *ei* toteuteta kaksoisrajapintaa. Vain ne asiakkaat, jotka voivat käsitellä komponentin vtablea, voivat kutsua sen rajapinnan metodeja. (Kokoonpanoa (aggregation) käsitellään tarkemmin luvun 11 oppitunnilla 3.)

7. Varmista, että kaikki valintaruudut ovat tyhjiä ja lisää sitten Encoder-objekti napauttamalla **OK**.

   ClassViewissä näet, että olet lisännyt **CEncoder**-komponenttiluokan ja **IEncoder**-rajapinnan, kuten kuvasta 9.6 nähdään.

Kuva 9.6 **CEncoder**-luokka ja **IEncoder**-rajapinta ClassView-näkymässä

CEncoder-komponenttiluokan mukana saat malliversion luokkatehtaasta, jota käytetään COM-komponenttien luomiseen. Samalla periytettiin myös **IUnknown** metodien **QueryInterface()**, **AddRef()**, ja **Release()**, perusversiot, joiden avulla asiakkaat käsittelevät asiakkaiden pääsyä rajapintojen osoittimiin ja säädellään COM-objektin elinaikaa.

Metodien lisääminen komponentin rajapintaan

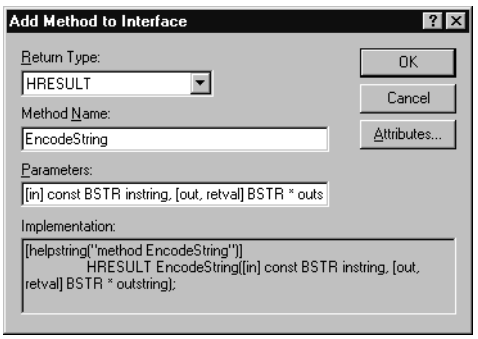
Kun COM-komponentille on ensin lisätty isännöintiympäristö, luokkatehdas ja **IUnknown**-metodit, lisätään komponenttikohtaiset metodit. Asiakas voi näiden metodien avulla käyttää COM-objektin palveluja. Metodien lisäämisessä komponenttiin käytetään Add Method to Interface Wizardia.

► **EncodeString()-metodin lisääminen**

1. Napauta ClassViewissä hiiren kakkospainikkeella **IEncoder**-rajapintaa.
2. Valitse pikavalikosta **Add Method**. **Add Method to Interface** -dialogi avautuu.
3. Valitse **Return Type** -ruudusta **HRESULT**.
4. Kirjoita **Method Name** -ruutuun **EncodeString**.
5. Sijoita seuraava koodi **Parameters**-ruutuun.

```
[in] const BSTR instring, [out, retval] BSTR * outstring
```

Implementation-ruudussa, Add Method to Interface Wizard näyttää syöttämiesi tietojen pohjalta muodostetun kutsurajapinnan MIDL listauksen kokonaisuudessaan, kuten kuvassa 9.7.



Kuva 9.7 Add Method to Interface -dialogi

Kun määrittelet parametrejä käyttämällä Add Method to Interface Wizardia, täytyy parametrilista antaa Interface Definition Languagen (IDL) käyttämässä muodossa. Tämä muoto vaatii, että käytät attribuutteja, joilla osoitetaan tiedonsiirron suunta. Määrittämällä suunnan [**in**] osoitat, että ensimmäiset parametrit välittävät tietoa metodille. [in]-parametri syrjäytetään arvolla.

EncodeString()-funktio ottaa [in]-parametrin ja luo koodatun merkkijonon käyttämällä yksinkertaista salausalgoritmia. Osoitin salattuun osoittimeen välitetään asiakkaalle [out]-parametrillä.

Huomaa Automaation merkkijonotyypin BSTR:n käyttö. Automaation kanssa yhteensopivien tietotyyppien käyttäminen rajapintojen metodien määrittämiseen antaa sellaisille kielille kuin Microsoft Visual Basic tai Microsoft Visual J++ mahdollisuuden käyttää komponentin tarjoamia palveluja.

Useimpien COM-rajapintojen metodien palauttama arvo on HRESULT — COM:n erityinen 32-bittinen arvo, joka ilmaisee onnistumisen tai epäonnistumisen tilan. HRESULTS on kuvattu tarkemmin luvun 13 oppitunnilla 2. Kaikki asiakaskielet (Visual Basic esimerkiksi) eivät pysty käyttämään HRESULT tietotyyppiä suoraan. [**retval**]-attribuutti osoittaa parametrin, jota käytetään tällaisissa tapauksissa. Visual Basicin suorituksenaikainen virheenkäsittelyjärjestelmä prosessoi funktion palauttaman HRESULT arvon, mutta Visual Basic -koodissa **EncodeString()** -funktion paluuarvoon viitataan [retval]-parametrillä. Näin ollen seuraava Visual Basic koodin pätkä näyttää sanan “Hello” salatussamuodossa sanomaikkunassa:

```
Dim comobj As Encoder

Set comobj = New Encoder

MsgBox comobj.EncodeString(“Hello”)
```

Lisää **EncodeString()**-metodi napauttamalla **OK**. Add Method to Interface Wizard sijoittaa antamiisi kuvaustietoihin perustuvan merkinnän projektin IDL-tiedostoon. Velho lisää myös C++-metodin **CEncoder**-komponenttiluokkaan.

Ominaisuuksien lisääminen komponentin rajapintaan

Ominaisuudet ovat COM-objektien julkisia tietojäseniä. Kielet, jotka tukevat COM:n ominaisuuksia voivat saada ja asettaa komponentin ominaisuuksien arvoja samaan tapaan kuin C++-luokan jäsenmuuttujienkin arvoja. Seuraava Visual Basic -koodi esimerkiksi näyttää ensin **Key**-ominaisuuden nykyisen arvon ja asettaa sitten uudeksi arvoksi 3:

```
Dim comobj As Encoder

Set comobj = New Encoder

MsgBox comobj.Key

comobj.Key = 3
```

Koska COM-rajapinta on itseasiassa funktioiden osoittimien taulukko, C++ toteuttaa ominaisuudet funktioparin avulla — toisella funktiolla asetetaan arvo ja toinen funktio antaa arvon ulospäin. Add Property to Interface Wizard luo automaattisesti **Get** ja **Put** metodit jokaiselle määriteltävälle ominaisuudelle, vaikka voitkin jättää **Put**-metodin toteuttamatta ja luoda vain-luku-tyyppisen ominaisuuden.

► **Key-ominaisuuden lisääminen**

1. Napauta ClassViewissä hiiren kakkospainikkeella **IEncoder**-rajapintaa.
2. Valitse pikavalikosta **Add Property**.
3. Valitse **Return Type** -ruudusta **HRESULT** ja **Property Type** -ruudusta **short**.
4. Kirjoita **Property Name** -ruutuun **Key**. Huomaa, että **get_Key** ja **put_Key()** funktioiden MIDL-kutsurajapinnat ilmestyvät **Implementation**-ruutuun.
5. Luo ominasuuden toteuttavat funktiot painamalla **OK**.

COM-objektin toteuttavassa luokassa täytyy määritellä jäsenmuuttuja, johon arvo sijoitetaan. Myös muuttujan arvon asettavan **Put** ja noutavan **Get**-metodin toteutukset täytyy tehdä.

► **Key-ominaisuuden toteuttaminen**

1. Lisää **short**-tyyppinen protected-jäsenmuuttuja **m_Key** luokkaan **CEncoder**.
2. Lisää **CEncoder::CEncoder()** muodostimeen seuraava rivi, joka alustaa m_Key-muuttujan arvon:

```
m_Key = 1;
```

3. Avaa ClassViewissä **IEncoder**-rajapinta **CEncoder**-luokan alta ja etsi **get_Key()** ja **put_Key()** -funktiot. Toteuta funktiot seuraavalla tavalla:

```
STDMETHODIMP CEncoder::get_Key(short *pVal)
{
    *pVal = m_Key;

    return S_OK;
}

STDMETHODIMP CEncoder::put_Key(short newVal)
{
    newVal = newVal > 5 ? 5 : newVal;
    newVal = newVal < -5 ? -5 : newVal;
    m_Key = newVal;

    return S_OK;
}
```

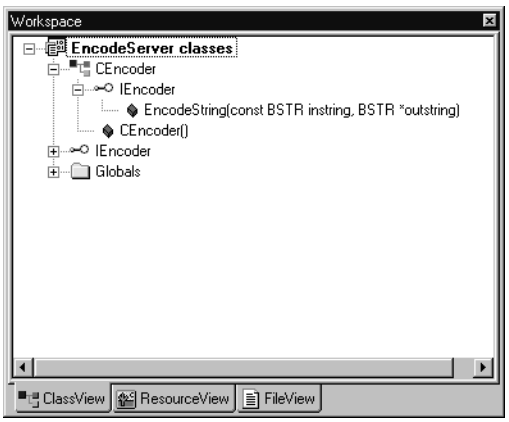
Huomaa **put_Key()**-funktioon lisätty yksinkertainen arvojen tarkistus.

Komponentin metodien toteuttaminen

Nyt kun **Key**-ominaisuus on toteutettu, voidaan toteuttaa **EncodeString()**-metodi.

► **EncodeString()-metodin toteutus**

1. Valitse **View**-valikosta **Workspace**, ja napauta sitten **ClassView**-välilehteä.
2. Avaa luokan sisältö näkyviin napauttamalla **CEncoder** vieressä olevaa plusmerkkiä.
3. Napauta **IEncoder** kohdan vieressä olevaa plusmerkkiä **CEncoder**-luokassa, jolloin näet tämän rajapinnan sisällön kuten kuvassa 9.8.



Kuva 9.8 **IEncoder**-rajapinnan metodit avattuna ClassViewissä

4. Kaksoisnapauta **CEncoder**-luokan **IEncoder**-rajapinnan alla olevaa **EncodeString()**-metodia. **EncodeString()**-funktion tiedostossa Encoder.cpp oleva toteutus avautuu muokkausikkunaan.
5. Lisää **EncodeString()**-funktion runkoon koodia niin, että se näyttää seuraavalta:

```
STDMETHODIMP CEncoder::EncodeString(const BSTR instring, BSTR *outstring)
{
    BSTR tempstring = ::SysAllocString(instring);
    wcscpy(tempstring, instring);

    for(UINT i = 0; i < ::SysStringLen(tempstring); i++)
        tempstring[i] += m_Key;

    *outstring = ::SysAllocString(tempstring);

    ::SysFreeString(tempstring);

    return S_OK;
}
```

Edellistä koodia on ehkä helpompi tulkita, jos tunnet hieman BSTR:ää. BSTR on itseasiassa osoitin merkkijonoon (**wchar_t*). Edellinen koodi käyttää BSTR:ää **wscpy()**-standardikirjastofunktion parametrinä. BSTR on kuitenkin enemmän kuin tavallinen merkeistä koostuva taulukko — sen alussa on 4-bittinen kokonaisluku, joka ilmaisee merkkijonon sisältämien bittien määrän. Tämän takia BSTR:n tarvitsema tila *täytyy* allokoida käyttämällä Win32 -funktiota **SysAllocString()**. Edellisessä esimerkissä BSTR allokoidaan

olemassaolevasta BSTR:stä. Toinen vaihtoehto on allokoinnin suorittaminen *wchar_t*-taulukosta kuten seuraavassa:

```
wchar_t wszPeas[] = L"Visualize Whirled Peas";
BSTR bstrPeas = ::SysAllocString(wszPeas);
```

Kun BSTR:ää ei enää tarvita, se deallokoidaan käyttämällä **SysFreeString()**-funktiota. Huomaa, että esimerkkimme BSTR *outstring* jätetään deallokoimatta, koska se täytyy palauttaa asiakkaalle. COM:ssa palvelimen asiakkaan pyynnöstä varaamien resurssien vapauttaminen on asiakkaan vastuulla.

Huomaa myös, että **EncodeString()**-funktiossa käytetään **SysStringLen()**-funktiota BSTR:n pituuden (merkeissä) selvittämiseen.

Voit nyt kääntää EncodeServer-projektin. Jos käännös ja linkitys onnistuvat, Visual Studio rekisteröi vielä lisäksi **Encoder-**objektin paikalliseen tietokoneeseen, jolloin asiakkaat voivat käytää **Encoder-**objektin palveluja. Luvussa 11 luotava asiakasohjelma tekee juuri näin.

Oppitunnin yhteenveto

Kun haluat luoda COM-objektin, yksinkertaisin tapa on käyttää ATL:ää. Tämä sovelluskehys sisältää useita ohjattuja toimintoja, jotka helpottavat työlskentelyä. Näitä toimintoja ovat esimerkiksi ATL COM AppWizard, ATL Object Wizard, Add Method to Interface Wizard ja Add Property to Interface Wizard. Näiden ohjattujen toimintojen avulla saat ATL:n luomaan kaikki COM:n COM-objektin luomisessa vaadittavat standardiosat. Voit näin keskittyä komponentin ominaisuuksien ja metodien toteuttamiseen. Näin saavutetaan parempi tuottavuus.

Oppitunti 2: ATL COM -komponentin koodi

Tällä oppitunnilla tutustut ATL:n ohjattujen toimintojen COM-objektiasi varten generoimaan koodiin. Tutustut määrättyihin koodin osiin kuten luokan määrittelyyn, luokan runkoon, globaaleihin tulopistefunktioihin, rekisteriskriptiresursseihin ja interface definition language (IDL) -tiedostoon. Kun olet tutustunut ATL-lähestymistavan vaatimaan koodiin, opit muutamia muita keinoja COM-objektin ohjelmointiin.

Tämän oppitunnin jälkeen:

- Tunnet **CComObjectRootEx** ja **CComCoClass** malliperusluokkien komponenteille tarjoaman toiminnallisuuden.
- Tunnet COM-palvelimeen sisällytetyt globaalit tulopistefunktiot.
- Tunnet ATL-projektille luodun rekisteriskriptitiedoston ominaisuudet.
- Tunnet ATL-projektille luodun IDL-tiedoston piirteet.
- Tunnet muutamia vaihtoehtoisia keinoja COM-komponenttien tekemiseen.

Oppitunnin arvioitu kesto: 30 minuuttia

Komponenttiluokan määrittely

Komponenttiluokan määrittely sijaitsee Encoder.h-tiedostossa. Tämä määrittely on tavanomainen moniperintää hyödyntävän C++-luokan määrittely.

```
class ATL_NO_VTABLE CEncoder :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CEncoder, &CLSID_Encoder>,
    public IEncoder
{
public:
    CEncoder()
    {
        m_Key = 1;
    }

DECLARE_REGISTRY_RESOURCEID(IDR_ENCODER)
DECLARE_NOT_AGGREGATABLE(CEncoder)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CEncoder)
    COM_INTERFACE_ENTRY(IEncoder)
END_COM_MAP()

// IEncoder
public:
    STDMETHOD(get_Key)(/*[out, retval]*/ short *pVal);
    STDMETHOD(put_Key)(/*[in]*/ short newVal);
    STDMETHOD(EncodeString)(/*[in]*/ const BSTR instring,
    /*[out, retval]*/ BSTR * outstring);
protected:
    short m_Key;
};
```

Luokka periytetään kolmesta luokasta: **CComObjectRootEx**, **CComCoClass** ja **IEncoder**. Kaksi ensimmäistä ovat ATL:n malliluokkia. Viimeinen luokka on rajapinta, joka on määritelty EncodeServer.h-tiedostossa abstraktina kantaluokkana. Rajapintametodit, joita määrittelet, lisätään **IEncoder**-rajapintaan puhtaina virtuaalifunktioina. Näistä funktioista luokkiin periytetyt versiot sisältävät COM-palvelimen palvelut. Luokka määrittelyn lopussa olevassa public-osassa on määri telty **EncodeString()**-funktion toteutus.

CComObjectRootEx-luokka sisältää perustoteutuksen **IUnknown**-rajapinnan metodeille **QueryInterface()**, **AddRef()** ja **Release()**. Kun periytät tästä luokasta, asiakas voi hankkia rajapinnan osoittimet mihin tahansa COM-objektisi tukemaan rajapintaan käyttämällä **QueryInterface()**-metodia. Metodit **AddRef()** ja **Release()** liittyvät viittauslaskimeen, jonka avulla varmistetaan, että käytössä olevaa COM-objektia ei poisteta muistista.

CComCoClass-kantaluokasta periyttämällä saadaan käyttöön luokkatehtaan perustoteutus. Luokkatehdas luo COM-palvelimen ilmentymiä. Tälle kantaluokalle toimitetaan argumentteina palvelinluokan nimi (**CEncoder**) ja viittaus sen GUID:hen (**CLSID_Encoder**). ATL:n velho käyttää UUIDGEN.EXE-apuohjelmaa projektin GUID:n muodostamisessa.

Toinen tärkeä osa luokan määrittelyssä on COM-kartta. Tämä kartta sisältää luettelon COM-objektin tukemista rajapinnoista. COM-karttaan tehdään lisäyksiä käyttämällä makroa **COM_INTERFACE_ENTRY**. Kulissien takana sovelluskehys ylläpitää vastaavaa **ATL_INTMAP_ENTRY** rakenteiden taulukkoa, joka yhdistää rajapinnan GUID:t fuktioihin, jotka vastaavat rajapintojen osoittimia. COM -karttaa käyttää usein **QueryInterface()**-metodi, joka periytettiin malliluokasta **CComObjectRootEx**. Aina kun asiakas suorittaa **QueryInterface()**-metodin, sen perustoteutuksessa käydään tämä kartan läpi ja etsitään sieltä vastaavaa GUID:ta. Jos vastaavuus löytyy, yhdistetty funktio palauttaa vastaavan rajapinnan osoittimen.

Komponentin metodien toteutus

Encoder.cpp-tiedosto sisältää edellisellä oppitunnilla tekemäsi **EncodeString()**-metodin rungon.

```
#include “stdafx.h”
#include “EncodeServer.h”
#include “Encoder.h”

STDMETHODIMP CEncoder::EncodeString(const BSTR instring, BSTR *outstring)
{
    BSTR tempstring = ::SysAllocString(instring);
    wcscpy(tempstring, instring);

    for(UINT i = 0; i < ::SysStringLen(tempstring); i++)
```

```
        tempstring[i] += m_Key;

    *outstring = ::SysAllocString(tempstring);

    ::SysFreeString(tempstring);

    return S_OK;
}
```

Tässä koodissa on muutamia COM:lle ominaisia piirteitä, jotka Add Method to Interface Wizard lisäsi puolestasi automaattisesti.

Metodissa käytetään paluuarvon tietotyyppinä makroa **STDMETHODIMP**. Tämä makro määritellään otsikkotiedostossa BASETYPS.H seuraavasti:

```
HRESULT export stdcall
```

Koska paluuarvoksi on määritelty HRESULT, Add Method to Interface Wizard lisää automaattisesti koodirivin, joka palauttaa vakion S_OK. Tämä reaalivakio noudattaa HRESULT:n tietorakennetta ja sisältää rakenteeseen liittyvän onnistumista ilmaisevan koodin. Lisää tietoja HRESULT-tyypistä saat luvun 13 oppitunnilta 2.

Globaalit tulopistefunktiot

Workspace-näkymän välilehdeltä **ClassView** löydät Globals-nimisen kansion. Jos avaat tämän kansion, näet useita globaaleja funktioita (etuliitteellä *Dll*) ja yhden globaalin objektin **_Module**. Nämä elementit esiintyvät tiedostossa EncodeServer.cpp, ja ne on projektiin lisännyt ATL COM AppWizard. Niiden tar-koituksena on toimia tulopistefunktioina komponentin DLL:lle ja niitä kutsuvat COM ja muut järjestelmän apuohjelmat. Seuraava koodi sisältyy EncodeServer.cpp-tiedostoon:

```
#include “stdafx.h”
#include “resource.h”
#include <initguid.h>
#include “EncodeServer.h”

#include “EncodeServer_i.c”
#include “Encoder.h”

CComModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_Encoder,   CEncoder)
END_OBJECT_MAP()

////////////////////////////////////////////////////////////////////////////
// DLL Entry Point
```

```
extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID /*lpReserved*/)
{
    if (dwReason == DLL_PROCESS_ATTACH)
    {
        _Module.Init(ObjectMap, hInstance, &LIBID_ENCODESERVERLib);
        DisableThreadLibraryCalls(hInstance);
    }
    else if (dwReason == DLL_PROCESS_DETACH)
        _Module.Term();
    return TRUE;    // ok
}

////////////////////////////////////////////////////////////////////////////
// Used to determine whether the DLL can be unloaded by OLE

STDAPI DllCanUnloadNow(void)
{
    return (_Module.GetLockCount()==0) ? S_OK : S_FALSE;
}

////////////////////////////////////////////////////////////////////////////
// Returns a class factory to create an object of the requested type

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, LPVOID* ppv)
{
    return _Module.GetClassObject(rclsid, riid, ppv);
}

////////////////////////////////////////////////////////////////////////////
// DllRegisterServer - Adds entries to the system registry

STDAPI DllRegisterServer(void)
{
    // registers object, typelib and all interfaces in typelib
    return _Module.RegisterServer(TRUE);
}

////////////////////////////////////////////////////////////////////////////
// DllUnregisterServer - Removes entries from the system registry

STDAPI DllUnregisterServer(void)
{
    return _Module.UnregisterServer(TRUE);
}
```

Lähellä tiedoston alkua näet yhden **CComModule**-objektin **_Module**:n määrittelyn. **CComModule**-luokka toteuttaa COM-palvelinmoduulin, jonka avulla asiakas voi käsitellä moduulin komponentteja. **CComModule** tukee sekä prosessin sisäisiä että ulkopuolisia moduuleja. Out-of-process (EXE) -palvelimet käyttävät sovelluskohtaisia objekteja, jotka on *periytetty* **CComModule**:sta.

CComModule:n ilmentymät käyttävät taulukkoa, josta käytetään nimitystä *objektikartta* (object map). Taulukossa säilytetään luokan objektien määrittelyjä. Kulissien takana objektikartta luo ja ylläpitää **_ATL_OBJMAP_ENTRY**-struktuurien taulukkoa, joka sisältää kehyksen seuraaviin tehtäviin tarvitsemat tiedot:

- Objektien luominen luokkatehtaan avulla.
- Asiakkaan ja juuriobjektin välisen kommunikaation muodostaminen.
- Luokan objektin elinajan hallinta.
- Objektien määrittelyjen rekisteröinti ja rekisteristä poistaminen.

OBJECT_ENTRY-makro on sijoitettu jokaisen DLL:n COM-objektin COM-karttaan. **OBJECT_ENTRY**-makro ottaa kaksi parametriä. Ensimmäinen parametri on objektin yksilöivä GUID. Toinen parametri on COM-objektin toteuttavan luokan nimi. Käyttämällä tätä informaatiota, makron laajennus luo vastaavan **_ATL_OBJMAP_ENTRY**-merkinnän määrätylle objektille.

Kun DLL ladataan asiakkaan pyynnön seurauksena, COM suorittaa funktion **DllGetClassObject()**. Tämä funktio kutsuu globaalin _Module-objektin **CComModule ::GetClassObject()**-kantaluokkaa päästen näin käsittelemään määrätyn objektin luokkatehdasta. **CComModule::GetClassObject()** käsittelee objektikartan ylläpitämää taulukkoa, josta se noutaa luokkatehtaan **CreateInstance()**-metodin osoittimen. **GetClassObject()**-funktio käyttää osoitinta COM-objektin luomiseen, ajaa COM-objektin **QueryInterface()**-funktion ja palauttaa rajapinnan osoittimen COM:lle.

EXE-palvelimet eivät toteuta *Dll*...-funktioita; ne kutsuvat COM:n suorituksenaikaista funktiota **CoRegisterClassObject()** jokaisen funktion sisältämän luokkatehtaan käynnistyksen yhteydessä. Luokkatehtaiden osoittimet sijoitetaan sisäisesti ylläpidettyyn taulukkoon.

DllRegisterServer() ja **DllUnregisterServer()** funktiot lisäävät prosessinsisäisiin COM-objekteihin itserekisteröintiominaisuuden. Näitä funktioita kutsumalla ohjelmat kuten komentoriviltä ajettava RegSvr32.exe voivat lisätä tai poistaa COM-objektia koskevat tiedot rekisteristä. EXE-palvelimet selvittävät komentorivivalitsimien — RegServer tai UnregServer — avulla, pitääkö palvelin rekisteröidä vai poistaa rekisteristä. Sekä EXE että DLL kutsuvat lopulta **CComModule ::RegisterServer()** ja **CComModule:: UnregisterServer()** -funktioita. Nämä funktiot rekisteröivät tai poistavat rekisteristä kaikki

objektikartassa esitellyt objektit käyttämällä rekisteriskriptiresurssiin varastoituja tietoja, jotka on kuvattu tarkemmin seuraavassa osassa.

Rekisteriskriptiresurssi

Kun asiakas yrittää ladata COM-objektia, COM:n suorituksenaikainen osa etsii rekisteristä tiettyä informaatiota. Nämä tiedot ovat rekisterissä **HKEY_CLASSES_ROOT (HKCR)**-avaimen alla. ATL-velho luo kaikki tarvittavat tiedot sisältävän rekisteriskriptin COM-komponenteille.

Rekisteriskripti lisätään projektin resursseihin. Skriptitiedoston tarkennin on .rgs, ja se on projektikansiossa. Seuraavassa Encoder.rgs-tiedosto:

```
HKCR
{
    EncodeServer.Encoder.1 = s 'Encoder Class'
    {
        CLSID = s '{69F4B917-6641-11D3-934B-0080C7FA0C3E}'
    }
    EncodeServer.Encoder = s 'Encoder Class'
    {
        CLSID = s '{69F4B917-6641-11D3-934B-0080C7FA0C3E}'
        CurVer = s 'EncodeServer.Encoder.1'
}
NoRemove CLSID
{
    ForceRemove    {69F4B917-6641-11D3-934B-0080C7FA0C3E}
        = s 'Encoder Class'
    {
        ProgID = s 'EncodeServer.Encoder.1'
        VersionIndependentProgID = s 'EncodeServer.Encoder'
        InprocServer32 = s '%MODULE%'
        {
        }
        'TypeLib' = s '{69F4B91A-6641-11D3-934B-0080C7FA0C3E}'
}
}
}
```

Juuri nähty rekisteriskripti sijoittaa tiedot rekisterin alihaaraan **HKEY_CLASSES_ROOT**. Merkinnät tehdään CLSID:lle, ProgID:lle (sekä versiosta riippuva että riippumaton muoto) ja tyyppikirjastolle. Koska Encoder-objekti on sijoitettu DLL-serveriin, luodaan InprocServer32-merkintä. Tämä merkintä sisältää muuttujan %MODULE%. Kun Visual Studio prosessoi rekisteriskriptiä, projektin todellinen nimi sijoitetaan tämän muuttujan paikalle.

IDL-tiedosto

IDL-tiedosto käännetään Microsoft IDL -kääntäjällä (MIDL). MIDL luo seuraavat tiedostot projektikansioon:

- Edustaja/sovitin-lähdekoodi perusetähallinnan toteuttamiseksi.
- Tyyppikirjastotiedosto.
- C-tiedosto, joka määrittelee komponentin rajapinnan GUID:t.
- C/C++-headertiedosto, joka määrittelee komponentin julkistamat rajapintame-todit.

Encoder-objektille generoitu edustaja/sovitin-lähdekoodi on sijoitettu tiedostoihin DllData.c ja EncodeServer_p.c. Tämä koodi käännetään *edustaja/sovitin-DLL:ksi*, jota COM-käyttää toteuttaessaan rajojen yli tapahtuvaa etähallintaa. Tyyppikirjastotiedosto EncodeServer.tlb on linkitetty DLL:ään, kun komponentti käännetään.

EncodeServer-projekti käyttää sekä GUID-määrittelytiedostoa (EncodeServer_i.c) että rajapinnanmäärittely otsikkotiedostoa (EncodeServer.h). On kuitenkin tärkeää tuntea nämä tiedostot, koska ne voidaan sisällyttää (#include-komennolla) C/C++-asiakkaan koodiin COM-komponenttien luomista ja käyttämistä varten. CLSID **CLSID_Encoder** ja IID **IID_IEncoder** määritellään EncodeServer_i.c-tiedostossa ja niitä voidaan käyttää parametreinä kutsuttaessa **CoCreateInstance()** -funktiota. EncodeServer.h-tiedosto määrittelee luokan, joka edustaa **IEncoder**-rajapintaa, joka sisältää kääntäjän tarvitsemat rajapinnan metodien kutsurajapinnat. Käytät tämän luokan ilmentymää kutsuessasi **IEncoder-**metodia, kun luodaan **Encoder-**objektia.

Näiden tiedostojen käytöstä asiakasohjelmissa kerrotaan luvun 10 oppitunnilla 1.

EncodeServer.idl-tiedosto on seuraava:

```
import “oaidl.idl”;
import “ocidl.idl”;
    [
        object,
        uuid(69F4B926-6641-11D3-934B-0080C7FA0C3E),
        helpstring(“IEncoder Interface”),
        pointer_default(unique)
    ]
    interface IEncoder : Iunknown
    {
        [helpstring(“method EncodeString”)] HRESULT
        EncodeString([in] const BSTR instring,
            [out, retval] BSTR * outstring);
        [propget, helpstring(“property Key”)]
            HRESULT Key([out, retval] short *pVal);
        [propput, helpstring(“property Key”)]
```

```
        HRESULT Key([in] short newVal);
    };

[
    uuid(69F4B91A-6641-11D3-934B-0080C7FA0C3E),
    version(1.0),
    helpstring("EncodeServer 1.0 Type Library")
]
library  ENCODESERVERLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");

    [
        uuid(69F4B917-6641-11D3-934B-0080C7FA0C3E),
        helpstring("Encoder Class")
    ]
    coclass Encoder
    {
        [default] interface IEncoder;
    };
};
```

Tämä IDL-tiedosto määrittelee **Encoder** COM-objektin, **IEncoder**-rajapinnan ja ENCODESERVERLib-tyyppikirjaston. Näet esimerkistä, että IDL:n syntaksi on hyvin samankaltainen kuin C++:ssa, huomattavimpanan erona atribuuttien sijoittaminen hakasulkuihin objektien määrittelyjen edessä.

MIDL-attribuutit määrittelevät rajapinnan piirteitä, apuluokkia ja kirjastoja. Esimerkiksi [**uuid**] on pakollinen attribuutti, joka määrittelee yksilöllisen tunnisteen (GUID) jokaiselle objektille. GUID, jotka esiintyvät tässä tiedostossa ovat ATL:n velhojen luomia. Huomaa [**propput**] ja [**propget**] attribuutit, jotka ilmaisevat sel-laisille kielille kuin Visual Basic, että metodia käsitellään ominaisuutena.

Huomaa, että apuluokkien määrittelyt on sijoitettu tyyppikirjaston määrittelyyn. Se tarkoittaa, että COM-objektin kuvaus sisällytetään tyyppikirjastoon. Sijoittamalla apuluokan määrittelyn kirjastolohkon *ulkopuolelle* voit estää COM-objektin tietojen lisäämisen tyyppikirjastoon. Näin voidaan menetellä, jos COM-objektia käytetään vain sisäisesti — muiden samassa DLL:ssä olevien objektien toimesta esimerkiksi.

Vaihtoehtoisia tapoja

ATL:n lisäksi on olemassa useita muitakin tapoja COM-objektien tekemiseen. Objekti voidaan luoda nollapisteestä käyttämällä C++:aa. Voit käyttää myös Microsoft Foundation Classes (MFC) -luokkia.

C++

COM-objektin ohjelmoiminen alusta alkaen C++:lla vaatii paljon ohjelmointia. Lopputulokseen pääseminen vaatii seuraavat vaiheet:

1. Luo DLL, joka isännöi prosessin sisäistä COM-palvelinta.
2. Periytä rajapintaluokka **IUnknown**-luokasta.
3. Muodosta GUID objektille (CLSID) ja rajapinnalle (IID).
4. Määritä rajapintaluokasta periytetty komponenttiluokka.
5. Toteuta **IUnknown**-metodit, jotka hallitsevat COM-objektin elinaikaa.
6. Toteuta COM-objektin varsinaisen tehtävän suorittavat komponenttikohtaiset metodit.
7. Tee luokkatehdas, joka huolehtii palvelimen luomisesta DLL:n muistialueelle.
8. Toteuta metodit **DllGetClassObject()** ja **DllCanUnloadNow()** prosessinsisäiseksi rajapinnaksi COM:lle.
9. Julkista **DllGetClassObject()** ja **DllCanUnloadNow()** -metodit käyttöjärjestelmälle.
10. Muodosta rekisteriskripti, joka sisältää objektiluokan ja luokan rajapinnan GUID:t.
11. Rekisteröi objekti käyttämällä rekisteriskriptiä.

Suurin osa näistä tehtävistä liittyy mekaanisiin vaiheisiin, joilla luodaan samanlainen COM:n peruskoodi, jonka ATL-muodostaa automaattiseti. Koska voit tehdä samat vaiheet muokkaamalla valmista koodia, tämä lähestymistapa on pitkällinen ja virhealtis. ATL:n käyttäminen säästää paljon aikaa ja tuottaa tehokkaita komponentteja, jotka toimivat todella hyvin.

MFC

On täysin mahdollista käyttää MFC:tä COM-objektin muodostamiseen. MFC on kuitenkin suunnattu sovellusten luomiseen ja sen COM-tuki liittyy voimakkaasti OLE:en. OLE (tunnetaan nykyisin yleensä ActiveX:nä) on COM-pohjainen tekniikka, joka antaa sovelluksille mahdollisuuden viedä ominaisuusryhmiä (niiden toimintojen ilmentymiä) muihin sovelluksiin. Esimerkiksi mahdollisuus upottaa Microsoft Excel laskentataulukko Microsoft Word -dokumenttiin hyödyntää OLE-tekniikkaa.

Koska OLE-ominaisuudet toteutetaan yleensä käyttämällä kaksoisrajapintaa, MFC COM -tuki perustuu **CCmdTarget**-luokan toteuttamalle sanoman-ohjaustekniikalle. Jos haluat luoda komponentin, joka julkistaa mukautetun rajapinnan, sinun on turvauduttava puhtaan C++-koodin käyttöön.

Toinen haitta MFC:stä COM:n yhteydessä on se, että MFC:n raskaat kirjastot täytyy toimittaa komponentin mukana. Jos komponentti toimitetaan sellaisen sovelluksen mukana, joka muutenkin sisältää kirjastot, tämä ei ole ongelma, mutta kirjastojen olemassaolon tarkastaminen ja asentaminen pienen komponentin takia on väsyttävää. Tietyissä tilanteissa MFC-kirjastojen staattinen linkittäminen objekteihin tekee niistä liian suuria. Yksi ATL:n kehittämiseen johtaneista syistä oli se, että MFC-komponentit ja kontrollit eivät soveltuneet Internetin kautta tapahtuvaan jakeluun ja käyttöön.

MFC on kuitenkin melko hyvä työkalu Automaatio-pohjaisten komponenttien tekemiseen ympäristöihin, joihin MFC-kirjastot voidaan toimittaa helposti. Voit käyttää MFC:tä luodessasi ActiveX-kontrolleja, joita voidaan helposti uudelleen käyttää MFC-sovelluksissasi useita kertoja. ActiveX-komponentteja käsitellään luvussa 11.

Oppitunnin yhteenveto

Kun teet COM-palvelimen käyttämällä ATL:n ohjattuja toimintoja, saat käyttöösi ATL:n malleihin ja makroihin perustuvan sovelluskehyksen. Komponentin luokat periytyvät kahdesta kantaluokasta: **CComObjectRootEx** ja **CComCoClass**. Nämä kaksi mallikantaluokkaa antavat käyttöösi **IUnknown**-rajapinnan metodit ja luokkatehtaan. ATL:n velhot tarjoavat käyttöösi joukon globaaleja tulopistefunktioita, jotka rekisteröivät komponentin ja antavat COM:lle mahdollisuuden luoda komponentin ilmentymiä asiakkaan pyyntöjen mukaan. Luotu IDL-tiedosto käännetään MIDL-kääntäjällä; se generoi edustaja/sovitinkoodin, tyyppikirjaston ja C/C++-headertiedostot, jotka määrittävät komponentin GUID:t ja määrittävät komponentin julkistamat rajapinnat. Asiakkaat voivat näitä header-tiedostoja käyttämällä luoda COM-objektien ilmentymiä ja kutsua rajapinnan metodeja.

Vaikka voit toteuttaa COM-objektit käyttämällä pelkkää C++:aa, helpoin ja turvallisin tapa on käyttää ATL:ää ja sen velhoja. MFC:n COM-tuki on suunnattu enemmän suurten automaatiopohjaisten komponenttien ja ActiveX-sovellusten kehittämiseen.

Laboratorio 9: STUpload tietolähde-komponentin luominen

Tässä laboratoriossa palataan STUpload-projektiin. Luomme prosessinsisäisen COM-palvelimen STLoadData.dll, joka isännöi **UploadStockData**-komponenttia. Tämä komponentti julkistaa **IUploadStockData**-rajapinnan, joka sisältää kolme metodia: **ConnectToDatabase()**, **Disconnect()** ja **UploadRecord()**.

Aloitetaan luomalla STLoadData-projekti STUpload-työtilaan.

► **STLoadData-projektin luominen**

1. Avaa STUpload-työtila. Valitse **File**-valikosta **New**.
2. Napauta **New**-dialogin **Projects**-välilehteä. Valitse projektiluokka **ATL COM AppWizard**.
3. Kirjoita **Project name** -ruutuun **STLoadData**. Valitse **Add to current workspace** ja napauta **OK**.
4. Napauta ATL COM AppWizardin vaiheessa 1 **Finish**, ja luo sitten uusi projekti napauttamalla **OK**.
5. STLoadData-projekti avautuu ClassViewiin. Projektin nimi on kirjoitettu liha-voituna, mikä tarkoittaa, että STLoadData on aktiivinen projekti. Varmista, että näytöllä on koko **Build**-työkalurivi eikä **Build**-minirivi. Työkalurivi antaa sinulle mahdollisuuden siirtyä projektista toiseen alasvetovalikkoa käyttämällä.
6. Napauta hiiren kakkospainikkeella **STLoadData**-projektia ClassViewissä. Valitse pikavalikosta **New ATL Object**.
7. Valitse **ATL Object Wizard** -dialogin **Category**-luettelosta **Objects**. Valitse **Objects**-ruudusta **Simple Object** -kuvake ja napauta sitten **Next**.
8. Valitse **ATL Object Wizard Properties** -dialogin **Names**-välilehti. Kirjoita **Short Name** -ruutuun **UploadStockData**.
9. Aseta **Attributes**-sivulla seuraavat attribuutit:
   - Valitse **Threading Model** kohdasta **Apartment**.
   - Valitse **Interface** kohdasta **Dual**.
   - Valitse **Aggregation**-kohdasta **No**.
10. Varmista, että yksikään valintaruutu ei ole valittuna, ja lisää sitten **UploadStockData**-objekti napauttamalla **OK**.

► **IUploadStockData-metodien lisääminen**

1. Avaa ClassViewissä kohta **STLoadData classes**. Napauta hiiren kakkospainikkeella **IUploadStockData**-rajapintaa.
2. Valitse pikavalikosta **Add Method**.
3. Kirjoita **Return Type** -ruutuun **HRESULT**.
4. Kirjoita **Method Name** -ruutuun **UploadRecord**.
5. Kirjoita **Parameters**-ruutuun seuraava koodi:

   ```
   [in] BSTR fund, [in] DATE date, [in] double price, [in] BSTR uplBy, [in] DATE uplDate
   ```

6. Toista samat vaiheet seuraavien metodien lisäämiseksi:

   ```
   HRESULT ConnectToDatabase()
   ```

   and

   ```
   HRESULT Disconnect()
   ```

Nämä metodit toteutetaan laboratoriossa 11, koska ne ovat itse toisten COM-komponenttien (ADO kirjasto) asiakkaita.

Tutki STLoadData.idl-tiedostoa. Huomaa erityisesti seuraava rajapinnan määrittely:

```
[
      object,
      uuid(241A7771-6888-11D3-934F-0080C7FA0C3E),
      dual,
      helpstring("IUploadStockData Interface"),
      pointer_default(unique)
]
interface IUploadStockData : Idispatch
{
      [id(1), helpstring("method UploadRecord")]
      HRESULT UploadRecord([in] BSTR fund, [in] DATE date,
            [in] double price, [in] BSTR uplBy,
            [in] DATE uplDate);
      [id(2), helpstring("method ConnectToDatabase")]
      HRESULT ConnectToDatabase();
      [id(3), helpstring("method Disconnect")]
      HRESULT Disconnect();
};
```

Huomaa, kuinka **IUploadStockData**-rajapinta periytyy **IDispatch**-rajapinnasta. Näin ATL-toteuttaa kaksoisrajapinnan. Skriptiasiakkaat voivat käyttää **UploadStockData**-komponenttia sanomarajapinnan kautta. Visual C++- ja Visual Basic -asiakkaat voivat käyttää suoraan komponentin vtablea.

Kertaus

1. Kuvaile mallikantaluokan **CComObjectRootEx** sisältämät ominaisuudet.

2. Kuvaile mallikantaluokan **CComCoClass** sisältämät ominaisuudet.

3. Mikä on COM-kartta ja kuinka sitä käytetään?

4. Mikä on objektikartta ja kuinka sitä käytetään?

5. Kuinka avainsanoja **interface**, **coclass** ja **library** käytetään IDL:ssä, ja kuinka ne liittyvät toisiinsa?