L U K U 8

Component Object Model

Tässä luvussa

Kun ohjelmoit Microsoft Visual C++:lla, Component Object Modelin (COM) hyvä tunteminen tekee sovelluksen rakentamisesta yksinkertaisempaa uudelleen käytettävien ohjelmaelementtien avulla. Ohjelman osien uudelleenkäyttö nopeuttaa kehitystyötä ja tuottaa luotettavampaa koodia.

Tässä luvussa esitellään COM:n käsitteistöä ja kerrotaan ActiveX-kontrolleista, jotka ovat itserekisteröityviä COM-komponentteja, jotka on suunniteltu sijoitettaviksi ActiveX-kontrollisäilöihin kuten dialogeihin tai Web-sivuille. Tämä luku sisältää luvuissa 9-11 välttämättömät taustatiedot. Niissä käsitellään COM-komponenttien käytännön toteutusta.

Ennen kuin aloitat

Ennen kuin aloitat tämän luvun läpikäymisen sinun tulisi lukea luvut 2-7 ja suorittaa niihin liittyvät tehtävät.

Oppitunti 1: COM-arkkitehtuuri

COM on binääristandardi, jossa määritellään tapa, jolla eri kielillä eri ympäristöissä tehdyt ohjelman osat voivat kommunikoida keskenään. COM-komponentti on uudelleen käytettävä ohjelman osa, joka noudattaa COM-määrittelyä.

Tällä oppitunnilla tutustut Win32-käyttöjärjestelmien perusarkkitehtuuriin, joka mahdollistaa COM-komponenttien luomisen ja käyttämisen sovelluksissa. Tutustut COM-määrittelyn tärkeimpiin ominaisuuksiin ja siihen, kuinka COM-komponentit rekisteröidään niin, että asiakassovellukset voivat tunnistaa ne ja käyttää niiden palveluja.

Tämän oppitunnin jälkeen:

- Tunnet arkkitehtuurin, joka antaa asiakkaalle mahdollisuuden COM-objektien käyttämiseen.
- Tunnet rajapintojen merkityksen COM:ssa.
- Tunnet **IUnknown**-rajapinnan merkityksen.
- Tiedät, kuinka globally unique identifiers (GUIDs) yksilöivät komponentit ja rajapinnat yksikäsitteisesti.
- Tunnet COM-objektin rekisteröimiseen tarvittavat rekisterimerkinnät.
- Tiedät, kuinka **CoCreateInstance()**-funktiota käytetään COM-objektien luomiseen.

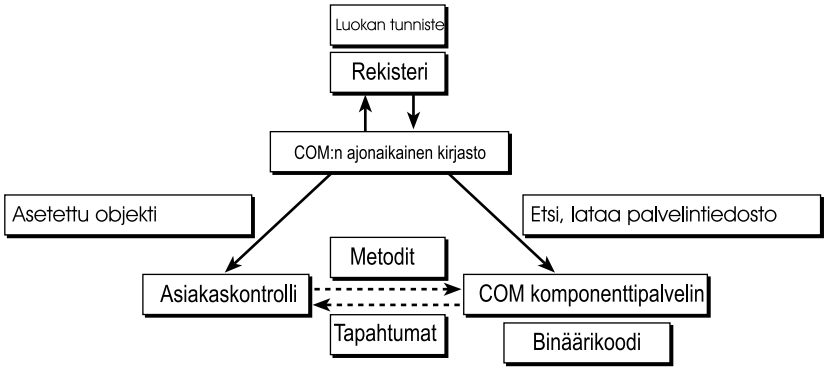
Oppitunnin arvioitu kesto: 50 minuuttia

COM-objektien käyttäminen

COM-objektien avulla voit rakentaa sovelluksesi yhdistelemällä erillisiä yhteistyössä toimivia komponentteja toisiinsa. Yksi COM:in käytöstä saatavia etuja on se, että ne mahdollistavat välillisen suorittamisen riippumatta siitä, missä binääriobjekti fyysisesti sijaitsee. Tärkeä COM-objektien piirre on myös mahdollisuus objektien suoritusaikana tapahtuvaan dynaamiseen käyttöönottoon.

Exe-tiedostoon sijoitettu sovellus voi esimerkiksi linkittää itseensä dynaamisesti DLL-tiedostossa sijaitsevan COM-komponentin. Koska komponentti ja suoritettava tiedosto linkitetään dynaamisesti, niiden ei tarvitse sijaita fyysisesti samassa paikassa. Binääriobjektin suorituksenaikaisen linkityksen aikaansaamiseksi ohjelma ja sovellus paikantavat binääriobjektin yhteistyössä. COM määrittelee kaikki elementit, joita sinä ja käyttöjärjestelmä käytätte yhteistyössä toimivan linkityksen aikaansaamiseksi.

Kuvassa 8.1 näet elementit, jotka osallistuvat dynaamiseen linkitykseen.



Kuva 8.1 COM-elementit, joita käytetään dynaamisessa linkityksessä

COM-objektit toteutetaan .dll tai .exe -tiedostoina. Jokainen järestelmässäsi sijaitseva COM-objekti täytyy rekisteröidä Windows-käyttöjärjestelmässä. Rekisteritietoja säilytetään Windowsin rekisterissä. Rekisterissä on luokan tunniste (ClassID), joka yksilöi objektin yksilöllisesti kaikissa tietokoneissa. ClassID on tietyllä tavalla muodostettu maailmanlaajuisesti yksilöllinen tunniste tai *GUID*. Opit lisää GUID:sta myöhemmin tällä oppitunnilla.

Kun asiakas haluaa muodostaa dynaamisen linkin COM-objektiin, se käyttää ClassID:tä binääriobjektin yksilöimiseen. COM:n suorituksenaikainen kirjasto, Windows-käyttöjärjestelmän sisäinen osa, sisältää keinot, joiden avulla asiakas paikantaa ja ottaa COM-objektin käyttöön.

Kun COM-kirjasto vastaanottaa ClassID:n, se lähtee tutkimaan rekisterin **HKEY_CLASSES_ROOT**-osaa selvittääkseen ClassID:n omistavan COM-objektin sijainnin. Jos sijainti selviää, COM-kirjasto luo objektin ilmentymän ja palauttaa sen rajapinnan osoittimen. Asiakas käyttää tämän osoittimen avulla objektin tarjoamia palveluja. Muussa tapauksessa asiakasta informoidaan virheestä. Sovellus hyödyntää dynaamisesti linkitettyä objektia käyttämällä sen rajapinnoissa lueteltuja palveluja. Jos objektin täytyy käynnistää vuorovaikutus asiakasohjelman kanssa, objektin sanomakäsittelijät rekisteröidään objektin kanssa. Objekti laukaisee tapahtuman sopivissa ja tarkkaan määrätyissä tilanteissa. Nämä tapahtumakäsittelijät huolehtivat tapahtumien prosessoinnista.

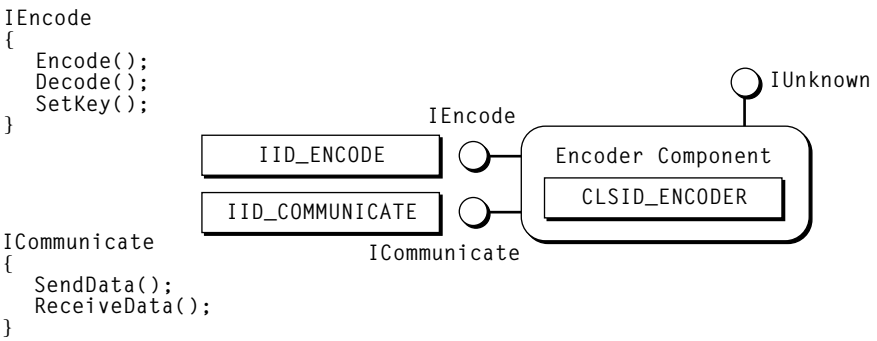
Yksi COM:n ominaisuuksista on *sijainnin läpinäkyvyys*. Asiakasohjelmaa tehtäessä ei tarvitse pohtia, suoritetaanko käytettävä objekti samassa prosessissa, toisessa prosessissa vai erillisellä tietokoneella olevassa prosessissa.

COM:n rajapinnat

COM-objektin palveluja käytetään yhden tai useamman rajapinnan kautta. COM-rajapinta on yhteen kuuluvien metodien looginen ryhmä, jolla on oma GUID, josta käytetään nimitystä Interface Identifier (IID). COM-komponentteja kuvataan usein käyttämällä asiakas/palvelin-mallin termistöä, jolloin COM-palvelin on komponentti, joka tarjoaa asiakasohjelmille palveluja julkistamiensa rajapintojen kautta. Tässä mallissa rajapinta voidaan käsittää komponentin tarjoamien palvelujen kuvaukseksi.

Rajapinnan tarjoamia palveluja päästään käyttämään hankkimalla rajapinnan osoitin, joka osoittaa funktioiden osoittimista muodostettuun taulukkoon, josta käytetään nimeä *vtable*. Jokainen vtablessa oleva funktion osoitin mahdollistaa yhden rajapinnan metodin käyttämisen.

Kuvassa 8.2 nähdään COM palvelinobjekti **Encoder**, joka julkistaa kolme rajapintaa. **IUnknown** (käsitellään seuraavassa jaksossa) on rajapinta, joka tulee toteuttaa jokaisessa COM-komponentissa. **IEncoder** ja **ICommunicate** -rajapinnat ovat komponentin suunnittelijan määrittämiä rajapintoja, jotka julkistavat **Encoder**-objektin palveluja. Kuvassa 8.2 havainnollistetaan tavallista tapaa, jolla COM-komponentteja kuvataan. Tarjolla olevat metodit nähdään kaavion vasemmassa laidassa.



Kuva 8.2 Komponentit ja rajapinnat

Kuvasta 8.2 nähdään, että komponenteilla ja rajapinnoilla on erilliset GUID:t. Kuvassa näkyvät GUID:n nimet ovat #defined-määritysten avulla nimettyjä tunnisteita, joilla viitataan eri tyyppisiin GUID:hin. Myöhemmin tässä luvussa selviää, että GUID:t ovat suuria heksadesimaalilukuja.

Rajapinta (interface) on komponentista loogisesti erillään oleva yksikkö, jonka komponentti toteuttaa. Toisin sanoen useat komponentit voivat toteuttaa saman rajapinnan. Rajapinta vastaa C++:n abstraktin luokan määrittelyä. Rajapinnan

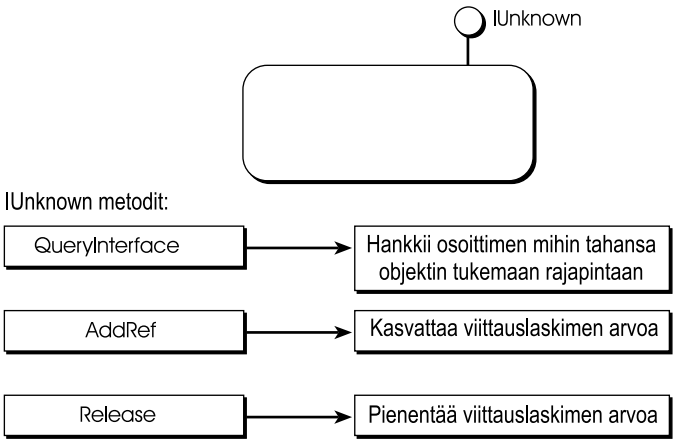
identiteetti määritellään sen yksilöllisellä IID:llä. Kun rajapinnan määrittelyt jukistetaan, taataan samalla kaikille rajapinnan käyttäjille ja toteuttajille, että *se ei muutu*. Metodien määrän, järjestyksen ja argumenttien sekä paluuarvojen tietotyyppien taataan pysyvän samoina. Jos metodeja halutaan lisätä tai muuttaa, täyttyy määritellä uusi rajapinta, jolla on toinen IID.

Se, että rajapinta saadaan käyttöön osoittimena funktiotaulukkoon (vtableen), näyttäisi viittaavan siihen, että COM-objekteja voivat käyttää vain asiakassovellukset. Monet COMia tukevat kielet (Microsoft Visual Basic, esimerkiksi) tukevat *ominaisuuksia* (properties) — komponenttien julkisia tietojäseniä, jotka vastaavat C++-luokan jäsenmuuttujia. C++:lla tehdyissä komponenteissa ominaisuudet toteutetaan funktioparina, jota käytetään kapseloi-dun luokan tietojen hakemiseen ja asettamiseen. Tätä tekniikkaa käsitellään luvussa 9.

Huomio Tässä kirjassa käytetään termejä *ominaisuus* (property) ja *metodi* (method) erityisesti viitattaessa COM rajapinnan ominaisuuksiin. Tavallisten C++-objektien yhteydessä käytetään termejä *jäsenmuuttuja* (member variable) ja *jäsenfunktio* (member function).

IUnknown

Jokaisen COM-komponentin täytyy toteuttaa **IUnknown**-rajapinta, joka on esitetty kuvassa 8.3.



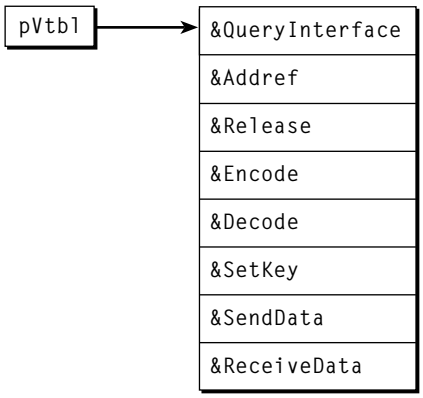
Kuva 8.3 **IUnknown**-rajapinta

IUnknown koostuu kolmesta metodista, jotka toteutetaan COM-objektissa. Toteuttamalla **QueryInterface()-**metodin annat asiakasohjelmalle keinon, jolla se voi käsitellä mitä tahansa COM-objektin rajapintaa.

Koska yhdellä komponentin ilmentymällä voi olla useita eri asiakkaita, täytyy COM-objektiin tehdä viittauslaskuri. Tämä suojattu tietojäsen sisältää tiedon komponenttiin yhteydessä olevien asiakkaiden määrästä. **AddRef()** ja **Release()** metodit käsittelevät tätä laskuria. Kun viittauslaskurin arvo putoaa nollaan, komponentti tuhoaa itsensä.

COM-komponentit täytyy toteuttaa niin, että **IUnknown**-metodit ovat aina vtablen kolme ensimmäistä metodia. Muiden rajapintojen julkistamat metodit luetellaan ryhmittäin näiden metodien jälkeen.

Kuva 8.4 esittää kuvan 8.2 **Encoder-**objektin mahdollista vtablea.



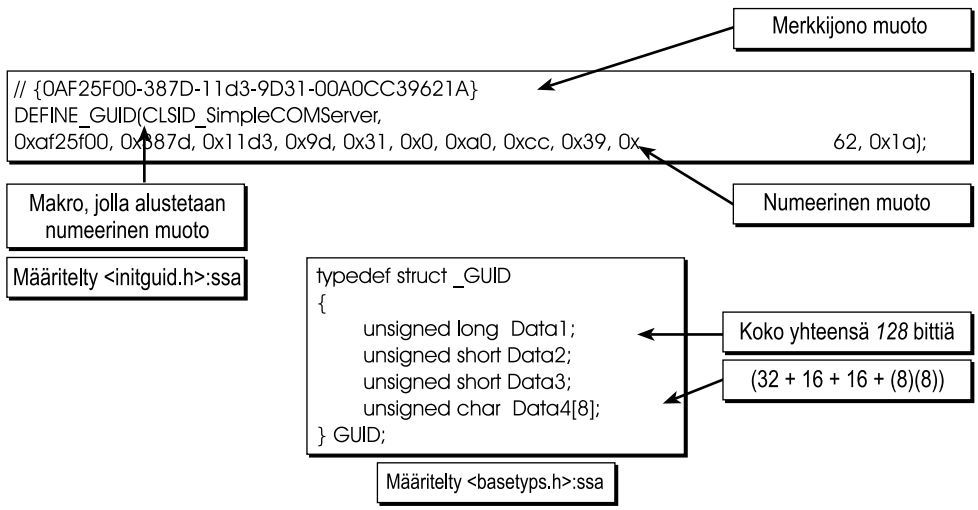
Kuva 8.4 Vtablen rakenne

Vtablen alussa ovat **IUnknown**-metodit, joita seuraavat ensin **IEncoder**-rajapinnan metodit ja sitten **ICommunicate**-metodit. Kun COM:n suorituksenaikainen kirjasto luo **Encoder**-objektin, se saa osoittimen vtablen alkuun (**pVtbl** kuvassa 8.4). Tätä osoitinta voidaan käyttää kutsuttaessa taulukon ensimmäistä funktiota — **QueryInterface()**-funktiota. **QueryInterface()**-funktion täytyy palauttaa osoitin pyydettyyn rajapintaan.

GUID

GUID:t ovat 128-bittisiä numerotunnisteita, jotka yksikäsitteisesti erottavat jokaisen COM-objektin ja COM objektin tukemat rajapinnat toisistaan. GUID:eille on annettu maailman laajuinen takuu siitä, että ne pysyvät yksilöllisinä hyvin pitkän ajan. GUID:n muodostamiseen käytetään komentoriviltä toimivaa UUIDGEN.EXE-apuohjelmaa (tai graafista versiota GUIDGEN.EXE).

GUID:n yksityiskohtainen kuvaus on kuvassa 8.5.



Kuva 8.5 GUID:n rakenne

GUID esiintyy itseasiassa kahdessa muodossa: merkkijonona ja numeerisena. Merkkijonomuotoa käytetään rekisterin useissa eri kohdissa. Numeerista esitystä GUID:sta tarvitaan käytettäessä GUID:ta asiakassovelluksessa ja COM-objektin varsinaisessa toteutuksessa.

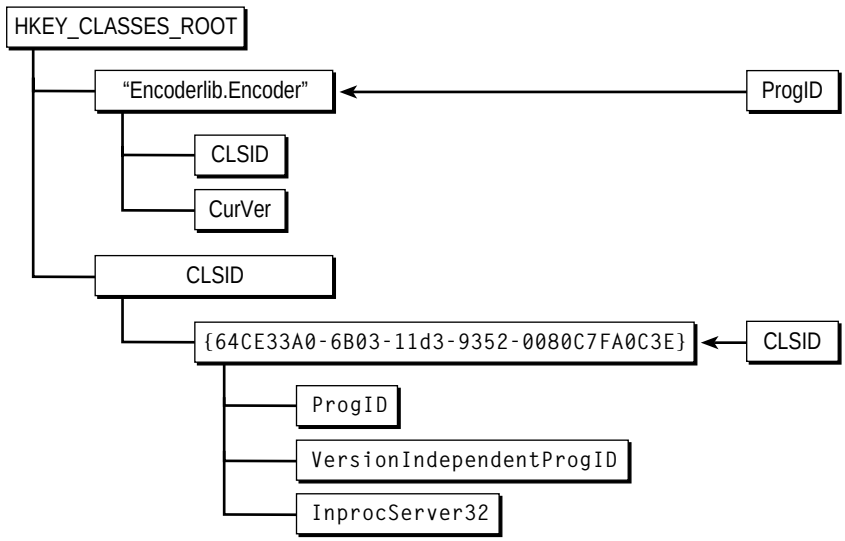
Kuten voi nähdä kuvasta 8.5, GUID:n numeerinen esitys on pituudeltaan 128-bittiä. _GUID rakenteessa unsigned long -tyyppinen kenttä **Data1** on 32-bittiä pitkä. Kentät **Data2** ja **Data3** ovat unsigned short -arvoja, joista kumpikin tarvitsee 16-bittiä. Kahdeksan unsigned **char** -arvoa vaativat jokainen 8-bittiä. Jos nämä kaikki lasketaan yhteen, nähdään, että GUID:n numeerinen esitys koostuu 128-bittisestä esityksestä.

Kun numeerista esitystä käytetään COM-objektissa tai C++-asiakkaan koodissa, määritellään muuttuja ja sijoitetaan käytettävä arvo muuttujaan erityistä makroa käyttämällä. Käytettävä makro on nimeltään **DEFINE_GUID** ja se on otsikkotiedostossa initguid.h. Käytettävän muuttujan nimi alkaa tavallisesti etuliitteellä *CLSID* tai *IID*. Nämä etuliitteet osoittavat sen, viittaako GUID COM-objektiin vai sen tukemaan rajapintaan.

COM:n reksiterimerkinnät

Kun asennat COM-objektin tietokoneeseen, täytyy objekti rekisteröidä tekemällä merkintä tietokoneen rekisteriin.

Seuraavalla sivulla olevassa kuvassa 8.6 on esitetty tyypillisen COM-objektin vaatima rekisterimerkintä.



Kuva 8.6 COM-objektin rekisteröinti

COM-objekti rekisteröidään tekemällä merkintä **HKEY_CLASSES_ ROOT**-aliavaimen esimääritellyn **CLSID**-avaimen alle. Näiden merkintöjen avulla COM-kirjastot voivat paikallistaa COM-objektin ja ladata sen muistiin. **CLSID**-avaimen alle sijoitetaan aliavain, joka on COM-objektin merkkijonomuodossa oleva **CLSID**. Tämän aliavaimen alle sijoitetaan aliavain, joka sisältää komponenttipalvelimen polun, seuraavaan tapaan:

```
HKEY_CLASSES_ROOT\CLSID\{64CE33A0-6B03-11d3-9352-0080C7FA0C3E}\
    InprocServer32 = c:\Encoder\debug\Encoder.dll
```

Aliavaimen nimi — **InprocServer32** — osoittaa, että komponenttipalvelin on paikallisella koneella oleva DLL.

Heti **HKEY_CLASSES_ROOT**:n alla on avain, jossa määritellään COM-objektin merkkijononimi. Tämä nimi on versioriippumaton ohjelmallinen tunniste (ProgID). Tähän nimeen liittyen tehdään aliavain, joka on eksplisiittisesti nimetty **CLSID**, jonka arvo osoittaa COM-komponentin GUID:n merkkijonomuodossa. Tämän merkinnän polkuesitys näyttää seuraavalta:

```
HKEY_CLASSES_ROOT\Encoder\CLSID  =  {64CE33A0-6B03-11d3-9352-0080C7FA0C3E}
```

Käyttämällä **CLSIDFromProgID()**-funktiota asiakassovellus voi hakea CLSID:n ProgID:stä. ProgID:n avulla asiakassovelluksen ohjelma voi luoda COM-objektin ilmentymän ilman virhealtista CLSID-arvojen kirjoittamista lähdekoodiin. Visual Basic -asikkaiden koodissa COM-objektit esitetään aina ProgID:llä, joten jos objektia käyttää muu kuin C++-asiakas, täytyy ProgID rekisteröidä.

CoCreateInstance()-funktion käyttäminen objektien luomiseen

Kun CLSID on saatu käyttämällä **CLSIDFromProgID()**-funktiota, sovellus toimittaa CLSID:n COM suorituksenaikaiselle kirjastolle ladatakseen COM-objektin ja saadakseen rajapinnan osoittimen. Tämän prosessin suorittamiseen käytetään funktiota **CoCreateInstance()**. Käyttämällä CLSID:tä ja rekisteriä se paikantaa määrätyn objektin ja palauttaa objektin rajapinnan osoittimen.

CoCreateInstance()-funktion kutsurajapinta on seuraava:

```
STDAPI CoCreateInstance (REFCLSID rclsid,
    LPUNKNOWN pUnkOuter,
    DWORD dwClsContext,
    REFIID riid,
    LPVOID * ppv) ;
```

Funktion ensimmäinen argumentti on objektin CLSID. Toista argumenttia käytetään objektin koostamiseen toisen objektin osaksi. Koostamista käsitellään tarkemmin luvun 10 oppitunnilla 2.

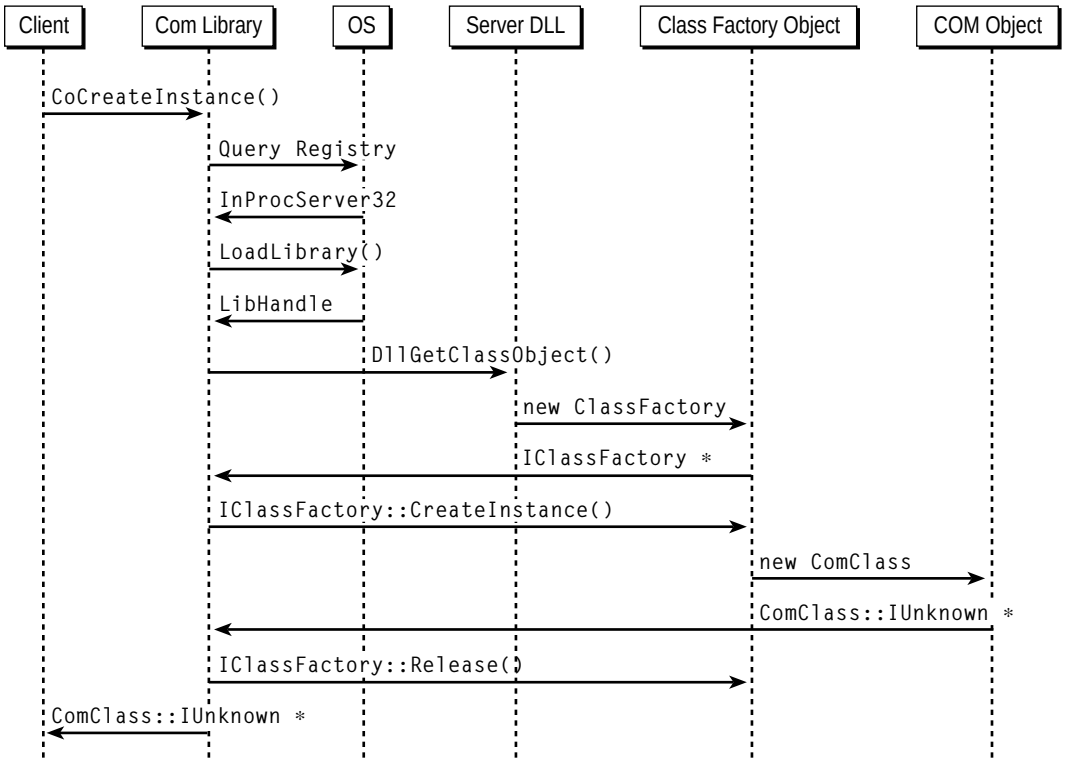
Kolmas argumentti määrittelee objektin suoritusympäristön. Tämän argumentin mahdolliset arvot ovat:

```
CLSCTX_INPROC_SERVER
CLSCTX_INPROC_HANDLER
CLSCTX_LOCAL_SERVER
CLSCTX_REMOTE_SERVER
```

Suoritusympäristöjä käsitellään seuraavalla oppitunnilla.

riid argumentti on pyydetyn rajapinnan IID. Jos COM-objektin luominen onnistuu ja se tukee asiakkaan pyytämää rajapintaa, rajapinnan osoitin palautetaan *ppv*-argumentin avulla.

CoCreateInstance()-funktion avulla käynnistettävän COM-objektin luomisprosessin yksityskohtainen analyysi on esitetty kuvassa 8.7 seuraavalla sivulla.



Kuva 8.7 COM objektin luomisvaiheet

Tämä vaihediagrammi esittää COM-objektin luomisessa käytettävät osat. COM-objektin luomiseen osallistuvat yksiköt on kuvattu diagrammin yläosaan. Prosessiin osallistuvien yksiköiden välisen viestinnän vaiheet etenevät ylhäältä alas.

Asiakassovellus pyytää pääsyä tiettyyn COM objektin tukemaan palveluun käyttämällä **CoCreateInstance()-**funktiota. Tämä funktio ottaa yhteyttä COM-kirjastoon pyytäen, että COM-objekti ladataan. COM-kirjasto tutkii rekisteriä osoitetun polun mukaisesti selvittäen palvelinkomponentin (tässä tapauksessa DLL) sijainnin ja nimen.

Kun komponenttipalvelin alustetaan, luodaan luokkatehdas. Luokkatehdas on objekti, joka luo tietyn luokan objekteja, joita käytetään COM-komponentin toteuttamiseen. Luokka tehdas toteuttaa **IClassFactory**-rajapinnan. Tämän jälkeen luokkatehtaan osoitin palautetaan aina COM-kirjastolle saakka. Käyttämällä osoitinta COM-kirjasto käynnistää luokkatehtaan tukeman **CreateInstance()-**metodin. Tämä metodi luo sen COM-luokan ilmentymän, joka todellisuudessa sisältää kaikki asiakkaan käytettävissä olevat metodit. Tämän luokan luominen muodostaa vtablen niin, että asiakas voi käyttää tuettuja metodeja. **IUnknown**-rajapinnan osoitin palautetaan COM-kirjastolle. Kun

COM-kirjasto vastaanottaa rajapinnan osoittimen, se vapauttaa luokkatehtaan ja palauttaa **IUnknown**-osoittimen asiakassovellukselle. Tämä osoitin viittaa nyt komponentin vtableen, antaen näin asiakkaalle mahdollisuuden COM-objektin tarjoamien metodien käyttämiseen.

Kaikki kuvatut vaiheet käydään läpi vain, jos asiakas luo COM-objektin ensimmäisen kerran. Jos toinen asiakas haluaa käyttää COM-objektin tukemaa metodia, sille toimitetaan jo suoritettavana olevan COM-objektin osoitin ja palvelimen viittauslaskurin arvoa kasvatetaan kutsumalla **IUnknown::AddRef()**-funktiota. Kun asiakas ei enää käytä koponentin rajapintaa, sen *täytyy* kutsua **IUnknown ::Release()**-metodia, joka pienentää viittauslaskurin arvoa. Jos yksikin asiakas jättää tämän tekemättä, viittauslaskurin arvo ei koskaan putoa nollaan, eikä komponenttia näin tuhota, mikä johtaa pahaan muistin vuotamiseen.

Oppitunnin yhteenveto

COM antaa mahdollisuuden dynaamisesti linkittää ohjelmakomponentteja, jotka voivat sijaita paikallisella tietokoneella tai verkossa olevalla tietokoneella. COM:n suorituksenaikainen kirjasto käyttää rekisterimerkintöjä COM-objektin etsimiseen ja lataamiseen. Jokaisen COM-objektin täytyy tukea omien rajapintojensa lisäksi **IUnknown**-rajapintaa. Jokainen rajapinta koostuu yhdestä tai useammasta metodista. Erilliset GUID:t yksilöivät COM-objektin ja jokaisen objektin julkistaman rajapinnan. Luokan toteutus sisältää luokkatehtaana tunnetun objektin, joka luo COM-objektin ilmentymiä.

Oppitunti 2: Rajapintojen etähallinta

COM palvelinta ei tarvitse ladata samaan prosessiin sen paveluja käyttävän asiakkaan kanssa. Asiakkaan ja pavelimen ei tarvitse edes sijaita samalla tietokoneella. Tällä on vaikutuksia tapaan, jolla tietoa siirretään asiakkaan ja palvelimen välillä. Kun COM-asiakas kutsuu COM-palvelinta, täytyy varmistaa, että palvelin saa parametrit ymmärtämässään muodossa, riippumatta siitä, missä tai millä alustalla palvelin toimii.

Tietojen siirtämistä prosessien rajojen yli kutsutaan *etähallinnaksi* (marshaling). Tällä oppitunnilla opit erilaisia tapoja, joilla COM-komponenttien etähallinta voidaan toteuttaa.

Tämän oppitunnin jälkeen:

- Tunnet eri suoritusympäristöt, joissa COM-komponentteja voidaan ajaa.
- Tunnet eri etähallintatekniikat.

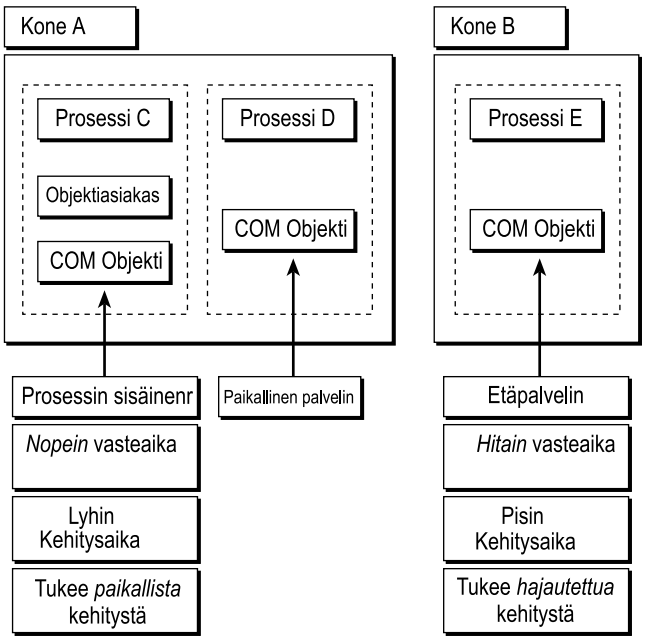
Oppitunnin arvioitu kesto: 30 minuuttia

Komponentin suoritusympäristö

COM-kompnenttia voidaan ajaa kolmella tavalla — prosessinsisäisenä palvelimena, prosessin ulkopuolisena palvelimena ja etäpalvelimena. Kuvassa 8.8 seuraavalla sivulla on yhteenveto näiden ympäristöjen toteutuksista ja ominaispiirteistä.

Prosessin sisäiset COM-palvelimet toteutetaan DLL:nä, ja ne ajetaan objektin asiakassovelluksen kanssa samassa prosessissa ja osoitetilassa. Koska COM-objekti sijaitsee samassa osoitetilassa, COM-palvelin vastaa nopeammin kuin mihin prosessin ulkopuolinen tai etäpalvelin pystyvät. Prosessin sisäisen palvelimen ohjelmoiminen on muihin verrattuna nopeampaa, koska kirjoitettavaa koodia on vähemmän. Tällainen palvelin täytyy kuitenkin asentaa ja rekisteröidä jokaiselle tietokoneelle, jossa asiakassovellusta aiotaan käyttää.

Prosessin ulkopuolinen COM-palvelin toteutetaan .exe-tiedostona, joka sijaitsee asiakassovelluksen kanssa samassa tietokoneessa, mutta suoritetaan eri osoitetilassa. Tästä syystä täytyy varmistaa, että metodien argumentit siirtyvät oikein prosessirajojen yli — täytyy siis toteuttaa etähallinta. Koska rajapintojen etähallinnan toteuttaminen vaatii hieman lisäkoodin kirjoittamista, COM-objektilta kuluu hieman pidempi aika kutsuihin vastaamiseen. Tietoja prosessirajojen yli välittävän lisäkoodin kirjoittamisesta seuraa hieman lisäkoodausta. Koska COM-objekti suoritetaan samalla tietokoneella kuin asiakasohjelmakin, täytyy COM-objekti kopioida jokaiseen asiakaskoneeseen.



Kuva 8.8 COM objektien suoritusympäristöt

Prosessin ulkopuolinen COM-objekti voidaan toteuttaa myös niin, että se suoritetaan etätietokoneella. Tällaista objektia kutsutaan *etäpalvelimeksi* (remote server). Tässä tapauksessa tiedonsiirto asiakastietokoneelta palvelinkoneelle tapahtuu tietoverkon välityksellä. Palvelimen vasteajat kasvavat huomattavasti. Metodien suorittaminen verkon kautta ei ole määrätyn kestoinen tapahtuma. Aina kun etämetodi käynnistetään, vaikka kyseessä olisi sama metodi, aiheutuva viive on eri pituinen. Jos tietoverkko on ruuhkan tukkima, vastauksen saaminen kestää todennäköisesti kauemmin. Sama koodi, jota käytetään paikallisten palvelimien etähallintaan, tukee myös etäpalvelimien etähallintaa. Jos käytät etäpalvelinta, siitä tarvitaan vain yksi etätietokoneelle asennettu ja rekisteröity kopio. Useammat etäasiakkaat voivat käyttää tätä yhtä COM-palvelinta. Koska palvelimesta on vain yksi kopio, COM-komponentin toteutuksen muuttaminen on suhteellisen tuskaton ja vähän vaivaa vaativa tehtävä.

Etähallintatekniikat

Asiakkaan ja palvelimen väline etähallinta on tärkeä asia toteutettaessa COM-objektia tai -palvelinta. Riippuen asiakkaan ja palvelimen välisestä suhteesta käytetään metodikutsujen etähallinnassa erilaisia ohjelmointitekniikoita.

Taulukossa 8.1 on esitetty ohjelmointitekniikat, joita voidaan käyttää COM-objektien metodien etähallinassa, etähallinnan tyypit ja rajat, joiden yli etähallinta tehdään, sekä etähallinnan lähestymistapa.

Taulukko 8.1 COM-objektien etähallinnassa käytettävät tekniikat

Etähallinnan tyyppi	Rajat	Lähestymistapa
ei etähallintaa	DLL	Globaalit osoitteet
Standardietähallinta	Prosessi	Interface Definition Language
Automaation etähallinta	Ohjelmointikieli	Automaation etähallinta
Mukautettu etähallinta	Prosessi	Erityinen ohjelmisto, protokolla

Kun COM-objekti toimii prosessin sisäisenä palvelimena, se pakataan DLL:ään, mikä tarkoittaa sitä, että COM-objekti ladataan asiakkaan prosessitilaan. COM-objektin metodit ovat suoraan asiakkaan kutsuttavissa ja tieto voidaan välittää vapaasti objektille ja objektilta.

Standardi etähallinta

Kun asiakas kutsuu paikalliselle tai etäpalvelimelle sijoitetun COM-komponentin rajapinnan metodeja, tieto kulkee prosessirajojen tai verkon solmujen yli. Tällaisen siirron toteuttamiseksi täytyy toteuttaa etähallinta, jotta voidaan olla varmoja, että asiakas ja palvelin osaavat kommunikoida keskenään. Visual C++ sisältää apuohjelman — *MIDL kääntäjä* — joka mahdollistaa prosessin ulkoisen palvelimen ja sen asiakkaan välisen etähallinnan toteuttavan DLL:n tuottamisen. Standardi etähallinta toteutetaan määrittämällä rajapinnat käyttämällä Interface Definition Language (IDL) -kieltä. IDL on vahvasti tyypitetty syntaksiltaan Visual C++:n kaltainen kieli, jonka avulla rajapinnat voidaan määritellä täsmällisesti.

MIDL-kääntäjä kääntää IDL-koodin ja muodostaa C-lähdekoodin kahta komponenttia, edustajaa ja sovitinta, varten. Edustaja liittyy asiakassovellukseen ja palvelin käyttää sovitinta. Kun tämä koodi käännetään, saadaan aikaan *edustaja/sovitin-DLL* (proxy/stub DLL), jota COM käyttää tiedonsiirron hoitamiseen prosessi- tai tietokonerajojen yli asiakkaan ja COM-palvelimen välillä.

Automaation etähallinnan käyttäminen

Automaation etähallinta toteutetaan käyttämällä COM-palvelinta (oleaut32.dll), joka sisältää COM-pohjaisen Automaatio-tekniikan (aiemmin OLE-automaatio) etähallintapalvelut. Automaation avulla muulla kuin C++-kielellä kir-joitetut sovellukset voivat käyttää COM-komponentteja. Automaation toteuttaa COM-rajapinta **IDispatch**, jota käsitellään tarkemmin seuraavalla oppitunnilla. Automation etähallintaa käytettäessä ei sanomarajapintaa tarvitse toteuttaa.

Komponenttien rajapinnat voidaan määrittää käyttämään Automaation etähallintaa määrittämällä ne IDL:n attribuutilla **oleautomation**, kuten seuraavassa koodissa nähdään:

```
[
      oleautomation,
      object,
      uuid(A84DA762-6486-11D3-9347-0080C7FA0C3E),

      helpstring("IHello Interface"),
      pointer_default(unique)
]
interface IHello : IUnknown
{
      [propget, helpstring("property String")] HRESULT
      String([out, retval] BSTR *pVal);
};
```

Jotta kommunikaatio eri kielien välillä olisi mahdollista, Automaatiossa määritellään joukko standardeja tietotyyppejä, jotka voidaan pakata yhdistelmätietorakenteeseen **VARIANT**. Automaation etähallintaa käytettäessä täytyy käyttää Automaatioyhteensopivia tietotyyppejä. **BSTR** tietotyyppi edellisessä koodinpätkässä on Automaatiossa käytetty merkkijonotyyppi.

Automaation etäkäsittely ei ole yhtä tehokas kuin MIDL:n tuottama standardi etäkäsittelykoodi.

Mukautettu etäkäsittely

Joissain tapauksissa standardi etäkäsittely ei sovellu käytettäväksi sovelluksessa. Sellaisissa tapauksissa palvelinobjekti voi tarjota tuen mukautetulle (custom) etäkäsittelylle.

Mukautetun etäkäsittelyn tukemiseksi palvelimen täytyy toteuttaa **IMarshal**-rajapinta. Kun palvelin tukee mukautettua etäkäsittelyä, se ei käytä standardeja etäkäsittelyn rakenteita. Sen sijaan COM vaatii, että palvelinobjekti muodostaa siirrettävät tiedot sisältävän paketin ja siirtää sen asiakkaan ympäristöön. COM muodostaa objektikohtaisen käsittelijän, joka ottaa paketin vastaan ja toimii asiakkaan puolella älykkäänä edustajana. Älykäs edustaja purkaa paketin asiakkaalle.

Tiedon siirto prosessirajojen yli onnistuu nyt samoin kuin standardissa etähallinnassakin. Asiakas ei ole tietoinen mukautetusta etähallinnasta (tai ylipäätään etähallinnasta).

Yleensä etähallinnan toteuttaminen ei ole COM-objektien tekijöille ongelma. MIDL muodostaa standardin etähallinnan vaatiman koodin ja Automaation etähallinta huolehtii sanomarajapinnan toteuttavien komponenttien

etähallinnasta. Joissain tapauksissa on kuitenkin erityisiä syitä, jolloin objektien täytyy tukea mukautettua etähallintaa. Näitä tilanteita ovat:

- **Jaettu muisti** Standardi etähallinta kopioi aina asiakkaan ja palvelimen välillä siirrettävät tiedot. Mukautettua etähallintaa käyttämällä voidaan luoda jaettu muisti, jota sekä asiakas että palvelin voivat käyttää ja näin vältytään toistuvalta kopioinnilta.
- **Arvoon perustuva etähallinta** Käyttämällä etähallinnassa viittausten sijasta arvoja voidaan objektista tehdä paikallinen kopio, jolloin kaikki seuraavat kutsut voidaan suorittaa paikallisesti.
- **Älykäs edustaja** Sisäisen välimuistin tilasta riippuen älykäs edustaja voi päättää, suoritetaanko etäkutsu vai ei. Joitakin metodeja voidaan toteuttaa suoraan tekemättä kutsua palvelimelle. Älykäs edustaja voi parantaa suorituskykyä myös mahdollistamalla asynkroniset kutsut. Asynkronisten kutsujen mahdollistamiseksi edustaja luo uuden säikeen, joka suorittaa kutsujan palaa palvelimen käsitellessä kutsua.
- **COM:n säiemallien ohittaminen** Aina ei välttämättä haluta olla sidoksissa perussäiemalleihin. Säiemalleja on käsitelty tämän luvun oppitunnilla 4.

Oppitunnin yhteenveto

Jokainen COM-objekti suoritetaan tietyssä ympäristössä. Prosessin sisäinen palvelin toimii asiakassovelluksen osoitetilassa. Kun asiakas kutsuu palvelinta, kutsu täytyy siirtää etähallinan kautta asiakkaan ympäristöstä palvelimen ympäristöön. Standardi etähallinta toteutetaan tekemällä IDL-kieltä ja MIDL-kääntäjää käyttämällä edustaja/sovitin DLL, joka huolehtii etähallinnasta asiakkaan ja toisessa prosessissa toimivan palvelimen välillä. Kielten välisessä etähallinnassa käytetään Automaation etähallintaan. Automaation etähallintaa käytettäessä täytyy käyttää Automaation kanssa yhteensopivia tietotyyppejä. Toteuttamalla **IMarshal**-rajapinnan, voit luoda erikoistapauksissa käytettävän mukautetun etähallintamekanismin.

Oppitunti 3: Sanomarajapinnat

Oppitunnilla 2 kerrottiin, että sanomarajapintoja käytetään Microsoftin Automaatiotekniikan toteuttamiseen. Automaatio mahdollistaa eri kielillä kirjoitettujen komponenttien välisen kommunikaation.

Tällä oppitunnilla kerrotaan tarkemmin, kuinka **IDispatch**-rajapintaa ja VARIANT-tietoyyppiä käyttetään apuna, kun halutaan antaa muille kielille mahdolisuus COM-komponenttien käyttämiseen. Opit myös, kuinka Microsoftin ohjelmointikielillä kuten Visual Basicilla tai Microsoft Visual J++:lla tehdyt asiakasohjelmat voivat käyttää komponentin rajapintoja suoraan käyttämättä sanomarajapintaa.

Tämän oppitunnin jälkeen:

- Tiedät, kuinka sanomarajapinta käynnistää metodeja asiakkaan puolesta.
- Tunnet VARIANT-tietotyypin rakenteen.
- Tiedät, kuinka Visual Basic -asiakkaat voivat tyyppikirjaston avulla käsitellä komponentin rajapintoja suoraan.
- Tiedät, kuinka komponentin metodit julkistetaan kaksoisrajapinnan avulla.

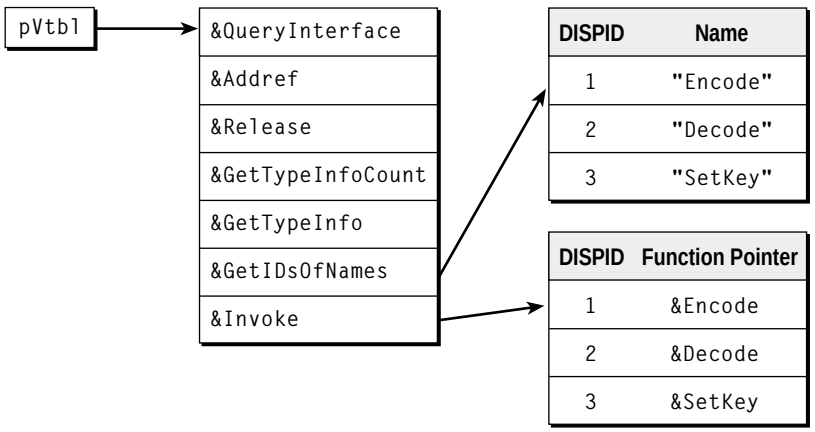
Oppitunnin arvioitu kesto: 30 minuuttia

IDispatch-rajapinta

Sanomarajapinnan avulla eri kielet voivat käyttää COM-objekteja. Ajatellaan esimerkiksi skriptikieltä kuten Microsoft Visual Basic Scripting Editionia (VBScript), jolla luodaan C++:lla kirjoitetun komponentin ilmentymä. VBScriptissä ei käytetä voimakasta tyypitystä ja se ei tunne kaikkia C++:n tietotyyppejä. Jotta VBScript-asiakas voisi kommunikoida onnistuneesti palvelinkomponentin kanssa, komponentilla täytyy olla *sanomarajapinta* (dispatch interface). Rajapinta tehdään toteuttamalla COM:n standardirajapinta **IDispatch**.

Tietyissä tilanteissa on tarkoituksenmukaista tehdä COM-objekteja, joita voidaan käyttää vain C++-asiakkaissa. Tällaisia voisivat olla esimerkiksi ohjelmistotalon sisäiseen käyttöön tarkoitetut komponentit. Sellaisessa tapauksessa ei kannata lisätä **IDispatch**-rajapinnan tuomaa lisäkuormaa. Jos COM-komponentin halutaan kuitenkin olevan käytössä useammilla kielillä, täytyy **IDispatch**-rajapinta toteuttaa. Muilla Microsoftin ohjelmointikielillä kuten Visual Basicillä tehdyt ohjelmat voivat käyttää C++:lla tehtyjä COM-komponentteja, joissa **IDispatch**-rajapintaa ei ole toteutettu. Visual Basic -asiakkaat voivat kuitenkin käyttää vain rajapintoja, joissa käytetään *automaatio-yhteensopivia* parametrejä ja paluuarvoja. Automaatioyhteensopivia ovat ne tietotyypit, joita voidaan pakata **VARIANT** standardiin tietomuotoon. VARIANT on COM:ssa määritelty yhdistelmätietotyyppi, jota käsitellään tarkemmin myöhemmin tällä oppitunnilla.

Kuvassa 8.9 on havainnollistettu yksinkertaisesti, kuinka **IDispatch**-rajapinta voitaisiin toteuttaa **Encoder**-luokaan.



Kuva 8.9 **IDispatch**-rajapinnan toteutus

Encoder-objektin vtable sisältää merkinnät, joka osoittavat toteutettuihin **IDispatch**-funktioihin. Asiakassovellus suorittaa **GetIDsOfNames()**-metodin, antamalla esimerkiksi metodin nimen "Encode" merkkijonona. **GetIDsOfNames()-**metodi pitää sisällään taulukon, joka yhdistää jokaisen metodin nimen numee-riseen tunnisteeseen, josta käytetään nimitystä Dispatch ID tai DISPID. DISPID on yksinkertaisesti numeerinen arvo — kuvan 8.9 esimerkissä **Encode**-funktion DISPID on 1.

Kun asiakassovellus on saanut haluamansa metodin DISPID:n, se voi kutsua metodia **Invoke()**-metodin avulla. **GetIDsOfNames()**-metodilta saatu DISPID välitetään argumenttina **Invoke()**-metodille. Myös kutsutulle metodille välitettävät parametrit annetaan **Invoke()**-metodille **VARIANT**-tyyppien muodostamana taulukkona. **Invoke()** ottaa myös kaksi **VARIANT**-osoitinta metodin palauttamien arvojen välittämistä varten.

Huomio Tässä prosessi on kuvattu yksinkertaistettuna. Käytettyjen metodien parametrit voidaan esimerkiksi nimetä ja niillä voi olla myös oma Dispatch ID.

Invoke()-metodin toteutus kutsuu pyydettyä metodia Automaation-asiakkaan puolesta. Toteutuksessa pitää huolehtia siitä, että siinä on jonkinlainen taulukko, joka yhdistää DISPID:t komponentin metodeihin. Sen täytyy myös purkaa argumentit **VARIANT**-taulukosta ja välittää ne pyydetylle komponentin metodille oikeassa muodossa. Kaikki paluuarvot täytyy pakata **VARIANT**-osoittimen määräämään objektiin asiakkaalle palauttamista varten.

VARIANT-tietotyyppi

VARIANT-tietotyyppi on määritelty tiedostossa OAIDL.IDL seuraavasti:

```
struct tagVARIANT {
    union {
        struct __tagVARIANT {
            VARTYPE  vt;
            WORD     wReserved1;
            WORD     wReserved2;
            WORD     wReserved3;
            union {
                LONG              lVal;      // VT_I4
                BYTE              bVal;      // VT_UI1
                SHORT             iVal;      // VT_I2
                FLOAT             fltVal;    // VT_R4
                DOUBLE            dblVal;    // VT_R8
                VARIANT_BOOL      boolVal;   // VT_BOOL
                _VARIANT_BOOL     bool;      // (obsolete)
                SCODE             scode;     // VT_ERROR
                CY                cyVal;     // VT_CY
                DATE              date;      // VT_DATE
                BSTR              bstrVal;   // VT_BSTR
                IUnknown *        punkVal;   // VT_UNKNOWN
                IDispatch *       pdispVal;  // VT_DISPATCH
                SAFEARRAY *       parray;    // VT_ARRAY
                BYTE *            pbVal;     // VT_BYREF|VT_UI1
                SHORT *           piVal;     // VT_BYREF|VT_I2
                LONG *            plVal;     // VT_BYREF|VT_I4
                FLOAT *           pfltVal;   // VT_BYREF|VT_R4
                DOUBLE *          pdblVal;   // VT_BYREF|VT_R8
                VARIANT_BOOL *    pboolVal;  // VT_BYREF|VT_BOOL
                _VARIANT_BOOl *   pbool;     // (obsolete)
                SCODE *           pscode;    // VT_BYREF|VT_ERROR
                CY *              pcyVal;    // VT_BYREF|VT_CY
                DATE *            pdate;     // VT_BYREF|VT_DATE
                BSTR *            pbstrVal;  // VT_BYREF|VT_BSTR
                IUnknown **       ppunkVal;  // VT_BYREF|VT_UNKNOWN
                IDispatch **      ppdispVal; // VT_BYREF|VT_DISPATCH
                SAFEARRAY **      pparray;   // VT_BYREF|VT_ARRAY
                VARIANT *         pvarVal;   // VT_BYREF|VT_VARIANT
                PVOID             byref;     // Generic ByRef
                CHAR              cVal;      // VT_I1
```

```
                    USHORT          uiVal;          // VT_UI2
                    ULONG           ulVal;          // VT_UI4
                    INT             intVal;         // VT_INT
                    UINT            uintVal;   // VT_UINT
                    DECIMAL *       pdecVal; //  VT_BYREF|VT_DECIMAL
                    CHAR *          pcVal;          //  VT_BYREF|VT_I1
                    USHORT *        puiVal;         //  VT_BYREF|VT_UI2
                    ULONG *         pulVal;         //  VT_BYREF|VT_UI4
                    INT *           pintVal;   //  VT_BYREF|VT_INT
                    UINT *          puintVal; //  VT_BYREF|VT_UINT
                    struct __tagBRECORD {
                        PVOID    pvRecord;
                        IRecordInfo *   pRecInfo;
                    } __VARIANT_NAME_4;       //  VT_RECORD
                } __VARIANT_NAME_3;
            } __VARIANT_NAME_2;
            DECIMAL decVal;
        } __VARIANT_NAME_1;
};
```

VARIANT-tietorakenne sisältää kaksi kenttää (jos varatut kentät jätetään laskematta). **vt**-kentässä kuvataan toisessa kentässä olevan tiedon tyyppi. Jotta toiseen kentään voitaisiin sijoittaa eri tyyppisiä tietoja, yhdistelmärakenne on nimetty. Tästä syystä toisen kentän nimi vaihtelee **vt**-kenttään syötetyn arvon mukaan. **vt**-kentän arvon määrittämisessä käytettävät vakiot on kuvattu rakenteen määrittelyssä vastaavalla rivillä kommenttina.

VARIANT ja **VARIANTARG** -tietorakenteiden käyttäminen vaatii kaksivaiheisen prosessin noudattamista. Otetaan esimerkiksi seuraava koodi:

```
long lValue = 555;
VARIANT vParam;

VParam.vt = VT_I4;
vParam.lVal = lValue ;
```

Ensimmäisellä rivillä määritetään tietotyypin tunniste. **VT_I4** vakio osoittaa, että toiseen elementtiin sijoitetaan pitkä kokonaisluku. **VARIANT**-tyypin määrittelystä nähdään, että toisen kentän nimeksi tulee **lVal**, kun **VARIANTARG** tietorakenteeseen tallennetaan pitkä kokonaisluku.

Käyttämällä **VARIANT**-tietoyyppiä parametrien välittämiseen heikommin tyypitetyt kielet, kuten VBScript, voivat käyttää vahvasti tyypitetyillä kielillä kuten C++:lla toteutettuja metodeja. **Invoke()**-metodissa voidaan tarkastaa, onko **VARIANT-**tyyppiin kapseloidun parametrin arvo oikean tyyppinen. Jos on, tieto voidaan purkaa ja välittää käynnistettävälle metodille. Jos ei, **Invoke()**-metodissa voidaan yrittää tietotyypin korjaamista käyttämällä **VariantChangeType()** API-funktiota.

Tyyppikirjastot

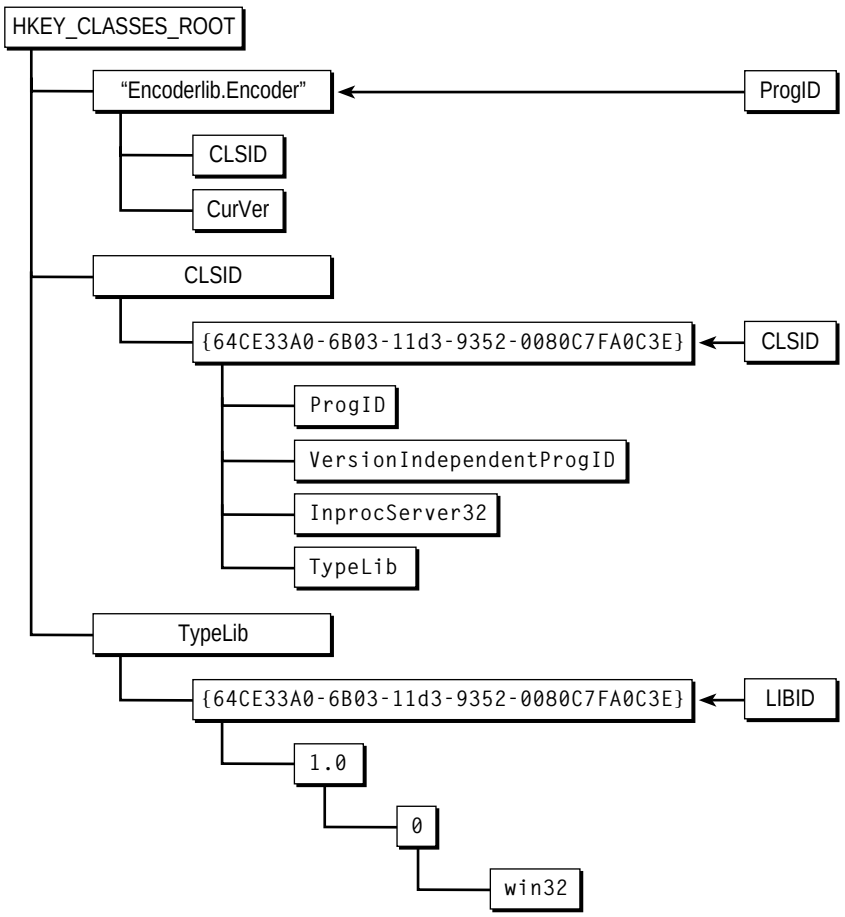
Voit ajatella, että edellä esitetty sanomarajapintamekanismi on varsin hidas tapa tietojen välittämiseen asiakkaan ja palvelimen välillä. Parametrien pakkaaminen ja purkamisen **VARIANT**-rakennetta käyttäen ei myöskään ole tyyppiturvallista. Korkean tason ohjelmointikielten, kuten Visual Basic 6.0:n, täytyy voida käyttää rajapinnan metodeja suoraan vtablen kautta.

Jotta tämä olisi mahdollista, Visual Basic asiakkaan täytyy tietää rajapinnan metodien tarvitsemien parametrien määrä ja tyyppi. Tämä tieto välitetään asiakkaille käyttämällä *tyyppikirjastoa* (type library) — joka on rajapinnan ominaisuuksien ja metodien binäärinen kuvaus — ja metodi argumenttejä. Tyyppikirjaston voi ajatella olevan käännetty, kielelestä riippumaton versio C++:n otsikkotiedostosta.

Visual Basic -ympäristö lukee tyyppikirjaston ja esittää kuvauksen ohjelmoijalle. Ohjelmoija luo komponentin ilmentymän käyttämällä Visual Basicin kielioppia, jonka kautta voidaan käsitellä komponentin vtablea suoraan. Tyyppikirjaston tiedot mahdollistavat argumenttien välittämisen rajapinnan metodien hyväksymässä muodossa. Näin Visual Basic -ohjelmoija voi ohittaa sanomarajapinnan ja parantaa suorituskykyä käyttäessään prosessin sisäistä palvelinta. Muista, että komponentin käyttäminen Visual Basic -asiakkaissa edellyttää, että käytetään vain Automaation tietotyyppejä. Tyyppikirjasto kuvataan käyttämällä IDL:ää. MIDL kääntäjä tuottaa tyyppikirjastotiedostoja, joiden tunniste on .tlb. Tämä tiedosto linkitetään usein siihen .dll tai .exe -palvelimeen, johon COM-komponentti on sijoitettu.

Jotta Visual Basicin kaltaiset asiakkaat voisivat käyttää tyyppikirjastoa, täytyy COM-objektia koskeviin rekisterimerkintöihin tehdä muutamia lisäyksiä. Nämä merkkinnät sisältävät COM-suorituksenaikaisen kirjaston tarvitseman tiedon tyyppikirjaston sijainnista.

Kuvassa 8.10 esitetään COM objektin rekisterimerkinnät, jotka tukevat tyyppikirjastoa.



Kuva 8.10 Tyyppikirjaston vaatimat rekisterimerkinnät

Itse asiassa tarvitaan vain kaksi lisämerkintää tyyppikirjaston sijainnin ilmoittamiseen. COM-objektin **CLSID**-aliavaimelle lisätään **TypeLib**-aliavain. Arvo, joka tähän aliavaimeen sijoitetaan, on tyyppikirjaston *LIBID* — tyyppikirjastolle IDL-tiedostossa määritelty GUID.

Tätä LIBID:tä käytetään toisena tyyppikirjaston reksiterimerkintänä, joka sijoitetaan **HKEY_CLASSES_ROOT \TypeLib**-avaimen alle. Kuten kuvasta 8.10 nähdään, LIBID avaimen alla on useita aliavaimia. **win32**-aliavain määrittelee tyyppikirjaston sisältävän tiedoston polun seuraavaan tapaan:

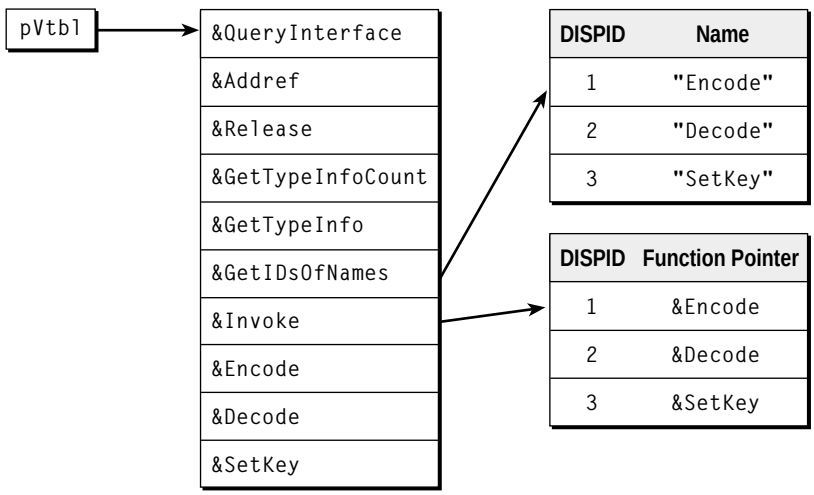
```
win32 = c:\Encoder\debug\Encoder.dll
```

Yhtäsuuruusmerkkiä seuraava arvo ilmaisee tyyppikirjastotiedoston fyysisen sijainnin polun. Tässä tapauksessa se on linkitetty COM palvelin DLL:ään. Sovellukset, joissa tyyppikirjastoja käytetään, osaavat lukea ne .dll ja .exe -tiedostoista.

Kun Visual Basicia käsketään viittaamaan COM-komponenttiin, se käyttää tätä informaatiota tyyppikirjastotiedoston polun selvittämiseen. Kun tiedoston polku on selvitetty, Visual Basic lukee ja tulkitsee tyyppikirjaston, muodostaa ohjelmoitavan esityksen ja antaa Visual Basic -ohjelmoijille mahdollisuuden kirjoittaa ja ajaa COM-palvelimen palveluja käyttäviä ohjelmia.

Kaksoisrajapinnat

Suositeltava tapa sanomarajapinnan toteuttamiseen on *kaksoisrajapinta* (dual interface). Kaksoisrajapinta mahdollistaa kaikkien **Invoke()** metodien kutsumisen suoraan komponentin vtablen kautta. Kuva 8.11 havainnol-listaa yhtä kaksoisrajapinnan toteutusta — siitä nähdään, kuinka **IEncoder**-metodien osoitteet on sisällytetty vtableen, samoin kuin **Invoke()**-funktion kautta käytettävään taulukkoonkin.



Kuva 8.11 Kaksoisrajapinnan toteutus

Jos toteutat kaksoisrajapinnan, komponentin julkistamat ominaisuudet ja rajapinnat ovat edelleen VBScriptin kaltaisten kielien käytettävissä sanomarajapinnan kautta; samalla kielet kuten Visual C++ voivat käyttää metodeja suoraan vtablen kautta. Visual Basicissa voidaan käyttää joko sanomarajapintaa tai vtablea. Kaksoisrajapinnan toteuttaminen antaa niille asiakkaille, jotka voivat käyttää vtablea, mahdollisuuden suorittaa nopeita metodikutsuja, sallien kui-tenkin sanomarajapinnan käyttämisen niille asiakkaille, jotka sitä tarvitsevat.

Oppitunnin yhteenveto

IDispatch-rajapinnan avulla sellaiset asiakasympäristöt kuin VBScript voivat käyttää COM-objekteja yhden aloituspisteen, **Invoke()**-metodin, kautta. **Invoke()**-metodille välitettävät argumentit pakataan **VARIANT**-tietorakenteisiin. Nämä tietorakenteet sisältävät kentän, joka ilmaisee tiedon tyypin ja toisen kentän, joka sisältää varsinaisen tiedon.

Jotta Visual Basicin tapaiset kehitysvälineet voisivat tarjota ohjelmoitavan rajapinan COM-objektille, täytyy sitä varten luoda tyyppikirjasto.Tyyppikirjasto on binääritiedosto, joka sisältää COM-objektin metodien, ominaisuuksien ja käytettyjen tietotyyppien kuvaukset. Visual Basic voi käyttää tyyppikirjaston sisältämää tietoa tehdessään suoria metodikutsuja COM-komponentin vtablesta saamiensa funktio-osoittimien avulla.

Kaksoisrajapinta mahdollistaa kutsujen tekemisen sekä vtablen että sanomarajapinnan kautta, asiakkaan kyvyistä riippuen.

Oppitunti 4: Säiemallit

Luvussa 5 opit, että voit parantaa sovelluksen suorituskykyä suorittamalla useita erillisiä tehtäviä samanaikaisissa säikeissä. Koska useat asiakassäikeet voivat olla samanaikaisesti yhteydessä samaan COM-objektin ilmentymään, täytyy varmistaa, että COM-objektin käsittely on asianmukaisesti synkronoitu.

Yleensä objektien synkronoimisen hoitaa käyttöjärjestelmä. Jotta voitaisiin tehdä monisäikeisessä ympäristössä tehokkaasti toimivia komponentteja, on tärkeää tuntea tapa, jolla käyttöjärjestelmä objektien synkronoinnin hoitaa. COM määrittelee neljä *säiemallia* (threading models), jotka yksinkertaistavat monisäikeisten komponenttein ohjelmointia.

Tämän oppitunnin jälkeen:

- Tunnet COM:n neljä säiemallia.
- Tunnet kunkin säiemallin käyttämisestä koituvat edut ja haitat.
- Tunnistat rekisteriavaimet, joita käytetään komponentin säiemallin määrittämiseen.

Oppitunnin arvioitu kesto: 30 minuuttia

COM:n säiemallit

Säiemalli kuvaa komponentissa toteutetun säieturvallisuuden tyypin ja tason. COM-komponentti voi tukea yhtä seuraavista neljästä säiemallista:

- single-threading-malli
- apartment-threading-malli
- free-threading-malli (monisäiemalli)
- mixed-threading-malli (tukee sekä apartment että free-threading mallia)

Ennen kuin asiakassäie voi käyttää COM-objektia, sen täytyy alustaa COM-kirjastot kutsumalla **CoInitializeEx()** API-funktiota. Näin tehdessään se voi määrittää mallin, jota se käyttää kutsuessaan luomiaan objekteja. Asiakassäie voi määrittää, että se käyttää apartment-threading mallia tai free-threading mallia.

Mikä tahansa asiakassovellus voi, riippumatta omasta säiemallistaan, turvallisesti käyttää missä tahansa säiemallissa toimivaa COM-objektia. Jos asiakkaan säiemalli on yhteensopimaton palvelimen mallin kanssa, COM:n suorituksenaikainen kirjasto huolehtii siitä, että asiakas ja palvelin voivat kommunikoida säieturvallisella tavalla. Tarvittavat toimenpiteet voivat kuitenkin hidastaa sovelluksen toimintaa merkittävästi.

Single-Threading-malli

COM-palvelin, joka tukee single-threading-mallia vaatii, että kaikki asiakkailta tulevat pyynnöt asetetaan jonoon yhteen säikeeseen. Single-threaded COM-palvelimet soveltuvat käytettäviksi vain yksisäikeisissä sovelluksissa. Single-threaded komponentin käsittely on hyvin tehokasta siinä säikeessä, joka on luonut komponentin. Säie, joka luo komponentin ensimmäisenä, voi saada suorat osoittimet komponentin julkistamiin rajapintoihin. Muilta säikeiltä tämän jälkeen tulevat kutsut eivät kuitenkaan voi käyttää suoria osoittimia, koska palvelimen prosessi tukee vain yhtä säiettä. Muiden säikeiden täytyy ohjata kutsunsa asiakkaan puolella olevan edustajan ja pavelimen pääsäikeessä olevan edustajan kautta. Ulkoisten säikeiden edustaja/palvelin-kutsut sijoitetaan palvelimen sanomajonoon. Ulkoisten säikeiden palvelutaso on heikko johtuen etähallinnan tuomasta lisäkuormasta ja siitä, että palvelimen luoneen säikeen suorien osoitteiden avulla tekemät palvelupyynnöt ohittavat jonossa olevat pyynnöt.

Apartment-Threading-malli

Apartment-threading-mallissa kaikki asiakkaat saavat käyttöönsä suoran osoittimen komponentin rajapintaan, ilman edustajien ja sovittimien käyttämistä. Apartment on käsitteellinen yksikkö, joka sisältää säikeiden samanaikaisen suorittamisen loogisen rakenteen. Apartment luodaan, kun säie kutsuu **CoInitializeEx()** API-funktiota alustaakseen COM-kirjastot. Apartment liitetään yhteen tai useampaan säikeeseen ja yhteen tai useampaan COM objektin ilmentymään. Edellyttäen, että apartmentin sisällä luodut objektit tukevat apartment-threading-mallia, kaikki apartmentin säikeet saavat käyttöönsä suorat osoittimet kaikkien apartmentin sisällä olevien objektien rajapintoihin.

Apartment-threading-mallin terminologia voi olla hämäävää, koska on olemassa kahden tyyppisiä apartmenteja: *single-threaded apartmenteja* (STA) ja *multithreaded apartmenteja* (MTA). Jako näiden välillä on seuraava:

- STA toteuttaa apartment-threading-mallin.
- MTA toteuttaa free-threading-mallin.

Prosessi voi sisältää useita STA:ita (joissa jokainen pitää sisällään yhden säikeen) mutta vain yhden MTA:n, johon voi kuulua useita säikeitä.

Single-Threaded-apartmentit

STA:ssa on vain yksi säie, joka voi luoda ja kutsua objekteja. Koska vain yksi säie voi käsitellä apartmentin objekteja, objektit ovat tehokkaasti synkronoituja.

STA:ta tukevat komponentit ovat suorituskykyisempiä kuin ne, jotka tukevat single-threaded mallia. Voit kirjoittaa STA-mallin avulla tehokkaampaa koodia. Samaan aikaan, kun säie yhdessä STA:ssa odottaa operaation päättymistä, toinen STA voi sallia säikeen välillisen suorittamisen kolmannessa STA:ssa.

Kun säie kutsuu **CoInitializeEx()**-funktiota käyttämällä *COINIT_APARTMENTTHREADED*-parametriä, säie luo STA:n. Säiettä, joka on alustettu STA:ksi, kutsutaan STA-tyyppiseksi säikeeksi. COM-objekti, joka on luotu STA-tyyppisessä säikeessä on vain kyseisen säikeen käytettävissä. Näin estetään useita säikeitä kutsumasta COM-objekteja samanaikaisesti.

Säikeiden synkronointi

Säie, joka käyttää toisessa apartmetissa sijaitsevaa COM-objektia, sen täytyy käyttää COM-objektin rajapintojen osoittimia etähallinnan kautta. Säikeet eivät voi käyttää rajapintojen suoria osoittimia, koska silloin useammat säikeet voisivat käsitellä objektia, mikä rikkoo STA-mallin sääntöjä.

Välittäessään rajapintojen osoittimet toisille säikeille, COM käyttää ikkunan sanomia useampien säikeiden synkronointiin. Jokaisella STA-säietyyppillä on sanomasilmukka, joka vastaanottaa muilta saman prosessin apartmenteilta tulevat kutsut. Kun asiakas kutsuu palvelinobjektia, etähallintakoodi sijoittaa vastaavan sanoman palvelimen sanomajonoon. Kaikki säikeelle tulevat kutsut asetetaan sanomajonoon siksi aikaa, kun objektin STA-säietyyppi käsittelee jokaisen sanoman yhden kerrallaan.

Rajapintojen osoittimien välittäminen

Kun rajapintojen osoittimia välitetään apartmentien välillä, se täytyy tehdä etähallinnan avulla. Välitettäessä osoitteita saman prosessin aparmentien välillä käytetään COM API -funktiota **CoMarshalInterThreadInterfaceInStream()**. Lisää tietoja **CoMarshalInterThreadInterfaceInStream()**-funktiosta saat Visual Studion ohjeesta.

Multithreaded Apartmentit

MTA:ta kutsutaan myös free-threading-malleiksi. Tämä malli eroaa STA:sta siinä, että se sallii useamman säikeen sijoittamisen samaan apartmentiin. MTA:n avulla saavutetaan suurin suorituskyky.

Huomio MTA-malli otettiin käyttöön Windows NT:n versiossa 4.0 ja se on käytettävissä myös Windows 95:ssä, jos DCOM95 on asennettu.

Kaikki säikeet kutsuvat **CoInitalizeEx()**-funktiota käyttämällä *COINIT_MULTITHREADED*-parametriä yhdessä MTA:ssa ja niitä kutsutaan MTA-säietyypeiksi. Toisin kuin STA mallissa, prosessissa on vain yksi MTA. Kun uusia MTA-säikeitä lisätään MTA-säietyyppeinä, ne toimivat samassa apartmentissä. Myöskään etähallintaa ei säikeiden välillä tarvita.

MTA-tyyppisten COM-objektien täytyy olla säieturvallisia ja niiden täytyy sisältää synkronointikoodi. Koska etähallinta ei aiheuta pullonkaulaa, MTA:t tarjoavat palvelinosassa parhaan mahdollisen suorituskyvyn.

MTA-mallissa säikeet voivat kutsua COM-objektia samanaikaisesti, eikä COM synkronoi kutsuja. Koska synkronointia ei suoriteta, MTA:ta varten kirjoitettujen COM-objektien täytyy olla säieturvallisia. Tästä syystä synkroinointiobjekteja kuten tapahtumia (kuten luvun 5 oppitunnilla 3 kerrottiin) täytyy käyttää komponentin staattisten ja globaalien tietojen suojaamiseen.

Säiemallien rekisteröinti

Prosessinsisäisten palvelinten täytyy kirjata säiemallinsa rekisteriin, koska ne eivät yleensä kutsu funktiota **CoInitializeEx()**. Kun asiakas luo prosessinsisäisen COM-palvelinobjektin, COM käyttää rekisterissä olevaa merkintää selvittääkseen, tarvitaanko kyseisen rajapinnan osoittimen yhteydessä käyttää etähallintaa.

Säiemalli osoitetaan rekisterissä luomalla nimetty ThreadingModel-arvo seuraavaan avaimeen:

```
HKEY_CLASSES_ROOT\CLSID\[component CLSID]\InProcServer32
```

ThreadingModel voi sisältää yhden seuraavista arvoista:

- **None** Vain single-threading-malli
- **Apartment** Tukee STA-mallia
- **Free** Tukee vain MTA-mallia
- **Both** Sekä STA että MTA-malli

Oppitunnin yhteenveto

COM-määrittelyyn kuuluu neljä säiemallia, jotka yksinkertaistavat useiden säikeiden samanaikaisten yhteydenottojen synkronointia monisäikeisessä ympäristössä. Single-threading-malli vaatii, että kaikki objektille asiakkailta tulevat palvelupyynnöt asetetaan jonoon yhteen säikeeseen. Apartment-threading-malli käyttää STA:ta määritellessään vain yhdessä säikeessä käytettävissä olevien säikeiden loogisen ryhmän. Free-threading malli kokoaa säieturvalliset komponentit yhteen MTA:han. Jos komponentti tukee mixed-threading mallia, voivat sekä single että multi-threaded apartmentien säikeet turvallisesti käyttää sitä.

Apartment muodostetaan, kun säie alustaa COM-kirjastot kutsumalla **CoInitializeEx()** API-funktiota. **CoInitializeEx()**-funktion argumenttien avulla voidaan määritellä, käytetäänkö STA vai MTA-tyyppiä. Prosessin sisäiset palvelimet eivät tavallisesti kutsu **CoInitializeEx()**-funktiota — niiden säiemalli määritellään rekisterimerkinnän avulla.

COM varmistaa, että eri säiemalleja käyttävät asiakkaat ja palvelimet voivat toimia keskenään säieturvallisella tavalla, käyttämällä tarvittaessa etähallintaa. Kun rajapinnan osoitin siirretään apartmentien välillä, täytyy rajapinnan etähal-linnasta huolehtia kutsumalla **CoMarshalInterThreadInterfaceInStream()** COM API -funktiota.

Oppitunti 5: ActiveX-kontrollit

ActiveX-kontrolli on COM-objekti, joka on suunniteltu sijoitettavaksi ActiveX-kontrollisäilöön, kuten sovelluksen dialogiin tai Web-sivulle, suorittamaan jonkin erillisen toiminnon. ActiveX-kontrolleilla on yleensä graafinen käyttöliittymä.

Tällä oppitunnilla suoritetaan ActiveX-kontrollien lyhyt esittely ja kiinnitetään huomiota muutamiin kontrolleja tehtäessä tärkeisiin seikkoihin.

Tämän oppitunnin jälkeen:

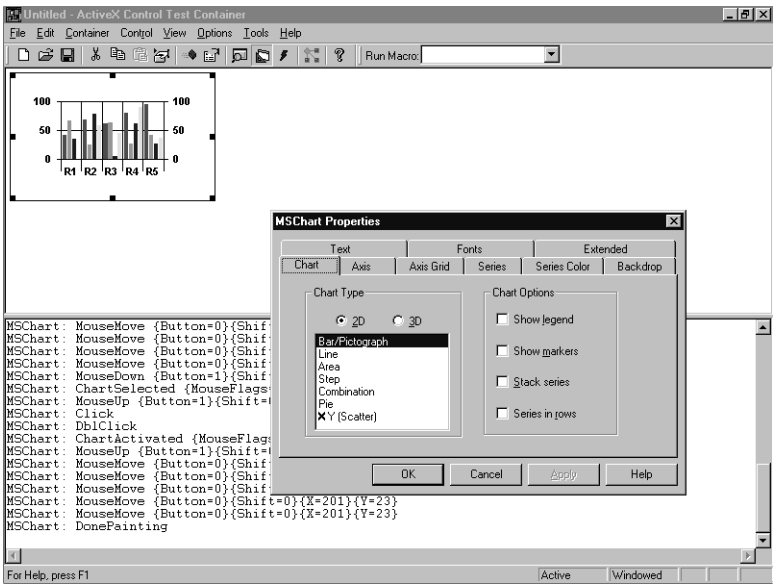
- Tunnet ActiveX-kontrollin ja ActiveX-säilön välisen vuorovaikutuksen.
- Tiedät, mitä rajapintoja ActiveX-kontrollin täytyy toteuttaa, jotta se vastaisi yleisimmin käytettyjen säilöjen vaatimuksia.
- Tunnet ActiveX-kontrollien tavallisimmat ominaisuudet.

Oppitunnin arvioitu kesto: 20 minuuttia

ActiveX-kontrollit ja säilöt

ActiveX-kontrolli on mikä tahansa COM-objekti, joka toteuttaa **IUnknown** rajapinnan ja jota isännöi itserekisteröityvä palvelin, joka voi olla DLL- tai .exe-tiedosto. ActiveX-kontrollin asiakasta kutsutaan *säilöksi* (container). ActiveX kontrollit sijoitetaan ActiveX-säilöön. Kuvassa 8.12 seuraavalla sivulla näet Microsoft Chart ActiveX-kontrollin, joka on sijoitettu ActiveX Control Test Containeriin.

ActiveX-kontrolleilla on tavallisesti näkyvä käyttöliittymä. Itse asiassa monet pitävät käyttöliittymän toteuttamista ActiveX-kontrollin määräävänä ominaisuutena. Kun tulevaisuudessa siirrytään käyttämään hajautettuja sovelluksia, on hyviä syitä sille, että tavalliset COM-komponentit eivät käytä näkyvää käyttöliittymää. Jos esimerkiksi COM-komponentti ilmoittaa virheestä käyttämällä dialogia ja kom-ponentti sattuu olemaan asennettuna toisessa rakennuksessa olevaan etäpalvelimeen, käyttäjä ei saa tietoa virheestä. Useimmat ActiveX-kontrollit on toisaalta tiukasti kytketty säilöön, jolla on käyttöliittymä. Säilö olettaa, että kontrolli toimii osana käyttöliittymää.



Kuva 8.12 ActiveX Control Test Container

Vaikka ActiveX-kontrollien määritelmä edellyttääkin tukea vain **IUnknown**-rajapinnalle, eri säilöillä on erilaisia vaatimuksia. Jotkin säilöt vaativat, että kontrollilla on toimiakseen oltava tietyt rajapinnat, kun taas toiset eivät. Jos kontrollin halutaan toimivan tietyn säilön kanssa, täytyy siihen sisällyttää säilön vaatimat rajapinnat.

Suurin osa normaalisti käytettävistä säilöistä olettaa, että kontrolli ylläpitää tietojoukkoa, laukaisee tapahtumia ja tukee rajapintoja, joita asiakassovellus tarvitsee ollakseen vuorovaikutuksessa sen kanssa. Taulukossa 8.2 seuraavalla sivulla on lueteltu rajapinnat, joita ActiveX Control Test Container (ja useimmat muutkin käyttöliittymäsäilöt) odottaa ActiveX-kontrollin tukevan.

Vaikka lueteltujen rajapintojen määrästä voisikin päätellä, että ActiveX-kontrollin tekeminen vaatii paljon raakaa työtä, sekä MFC ja ATL sisältävät ohjattuja toimintoja, joiden avulla voidaa luoda perusrunko ActiveX-kontrollille. Ohjattujen toimintojen ansiosta ohjelmoija voi keskittyä kontrollille ominaisten toimintojen toteuttamiseen.

Taulukko 8.2 Rajapinnat, joita ActiveX-kontrollin on vähintään tuettava

Rajapinta	Kuvaus
IOleObject	Tarvitaan kommunikointiin kontrollin asiakkaan kanssa paitsi käytettäessä tapahtumia. Tapahtumat hoidetaan **IConnectionPointContainer**-rajapinnan avulla.
IOleInPlaceObject	Toteutetaan kontrolleissa, jotka voidaan aktivoida paikoillaan ja sisältävät oman käyttöllittymänsä. Vaatii **IOleObject**-tuen.
IOleInPlaceActiveObject	Vaaditaan vain kontrolleissa, joilla on oma käyttöliittymä ja jotka tukevat rajapintaa **IOleInPlaceObject**.
IDataObject	Tarvitaan, jos kontrolli siirtää tietoa säilölle jollain tavalla, esimerkiksi yhteisen muistialueen tai tiedoston kautta. **IDataObject** sisältää COM:n Uniform Data Transfer protokollan, joka asettaa säännöt kaikentyyppisen tiedon vaihdolle.
IViewObject2	Toteutetaan näkyvissä kontrolleissa, jotka näyttävät ikkunan.
IDispatch	Tarvitaan kontrolleissa, joilla on mukautettuja ominaisuuksia tai metodeja, joita asiakas käyttää **IDispatch**::**Invoke()** metodin kautta.
IConnectionPointContainer	Vaaditaan, jos kontrolli laukaisee tapahtumia. Tämä rajapinta sisältää luettelon tapahtumista, jotka kontrolliobjekti voi laukaista.
IConnectionPoint	Vaaditaan, jos kontrolli tukee **IConnectionPointContainer**.
IProvideClassInfo	Toteutetaan kontrolleissa, jotka sisältävät tyyppikirjastotietoja, eli käytännössä useimmissa ActiveX-kontrolleissa. Rajapinnalta saadaan **GetClassInfo()**-metodia käyttämällä osoitin **ITypeInfo**-toteutukseen, josta asiakas voi hankkia kontrollin tyyppitiedot.
IPersistStorage	Vaaditaan kontrolleissa, jotka voivat käyttää säilön tarjoamaa **IStorage**:n ilmentymää.
IClassFactory	Alustaa pyydetyn luokan olion ja palauttaa osoittimen siihen. Olio identifioidaan CLSID-tunnisteella, joka on asetettu järjestelmän rekisterissä.
IClassFactory2	Sama kuin **IClassFactory**, mutta tukee lisensointia.

ActiveX-kontrollien ominaisuudet

ActiveX-kontrollit tukevat ominaisuuksia, metodeja ja tapahtumia, joiden avulla kontrollin toimintaa voidaan ohjata. ActiveX-kontrollit tukevat yleensä erillistä rajapintaa, jonka avulla suuri määrä ohjelmointi- ja skriptikieliä voi hyödyntää kontrollia ohjelmallisesti.

Varasto-ominaisuudet

ActiveX-standardissa määritellään joukko varasto-ominaisuuksia (stock properties), jotka ovat yhteisiä useille kontrolleille. Tällaisia ominaisuuksia ovat esimerkiksi kontrollin näyttämien tekstien esittämiseen käytettävä fontti, sekä tekstin ja taustan väri. Varasto-ominaisuudet erotellaan *mukautetuista* (custom) ominaisuuksista: mukautetut ominaisuudet ovat kotrollin toiminnalle ominaisia.

Ympäristöominaisuus

Ympäristöominaisuuksien (ambient properties) avulla kontrolli saa tietoa säilön ulkoasusta. Kontrollin ominaisuudet tulisi asettaa niin, että ne sopivat säilön ympäristöön. Esimerkiksi ympäristöominaisuus **BackColor** edustaa säilön taustaväriä. Kontrolli voi muuttaa oman varasto-ominaisuutensa arvon vastaamaan **BackColor**-ominaisuuden ilmoittamaa väriä, jolloin se mukautuu ympäristöönsä. Kontrollien ohjelmoijia kehoitetaan tekemään ActiveX-kontrollit niin, että ne havaitsevat ja reagoivat säilön ympäristöominaisuuksiin.

Tapahtumat

Kontrollit voivat reagoida toimenpiteisiin myös luomalla *tapahtumia* (events). Tapahtumat ovat kontrollilta säilölle meneviä ilmoituksia. Tapahtumarajapinta määritellään kontrollissa, mutta toteutus sijaitsee säilössä, jonka kautta sanomarajapinta on käytettävissä.

Kuten ominaisuudetkin tapahtumat jaetaan varastotapahtumiin ja mukautettuihin tapahtumiin. Varastotapahtumat ovat tavallisia toimia kuten hiiren napautuksia tai näppäimen painalluksia. Mukautetut tapahtumat ovat kontrollikohtaisia.

Ominaisuusikkunat

Useimmilla ActiveX-kontrolleilla on ominaisuusikkuna, joka tarjoaa käyttäjille graafisen liittymän, jonka kautta kontrollin ominaisuuksia voidaan käsitellä. Kuvassa 8.12 nähdään Microsoft Chart ActiveX-kontrollin ominaisuussivu. Jokainen ominaisuussivu perustuu dialogimalliresurssiin ja on erillinen COM objekti, jolla on oma CLSID.

Ominaisuuksien tallentaminen

Yksi ActiveX-kontrollien ominaisuuksista on ominaisuuksien tallentaminen (tai serialisointi), jolloin ActiveX-kontrollin ominasuuksien arvot voidaan tallentaa tiedostoon ja lukea sieltä myöhemmin. Säilösovellus voi käyttää serialisointia kontrollin ominaisuuksien arvojen tallentamiseen vielä senkin jälkeen, kun se on jo tuhonnut kontrollin. ActiveX-kontrollin ominaisuuksien arvot voidaa sitten lukea tiedostosta muodostettaessa myöhemmin kontrollin uutta ilmentymää.

Oppitunnin yhteenveto

ActiveX-kontrollit ovat tavallisia COM-objekteja, jotka on sijoitettu ActiveX-kontrollisäilöihin. ActiveX-kontrollin on toteutettava säilön vaatimat rajapinnat. Yleensä tämä tarkoittaa myös käyttöliittymän toteuttamista niin, että kontrolli voi toimia osana säilön käyttöliittymää.

ActiveX-kontrollit tukevat tyypillisesti ominaisuuksia, metodeja ja tapahtumia, jotka tekevät kontrollista ohjelmoitavan.

Laboratorio 8: OLE/COM Object Viewer

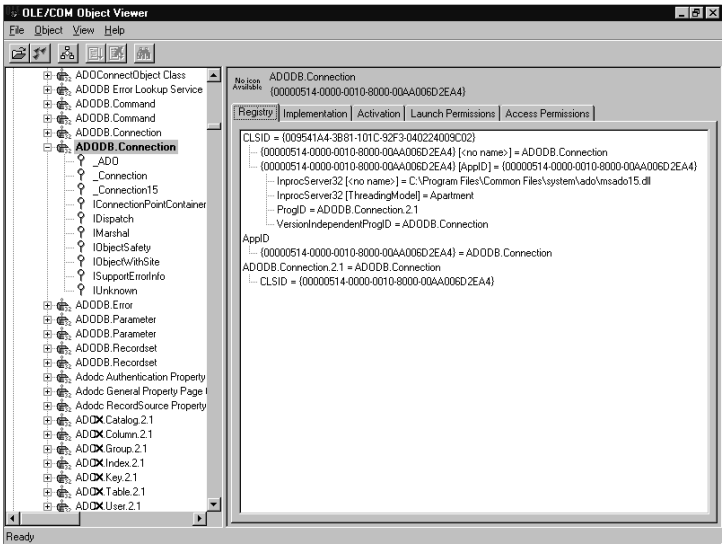
Tässä laboratoriossa tutustut COM-objektien ja ActiveX-kontrollien toteuttamiseen liittyviin aiheisiin. Käytät OLE/COM Object Viewerià rekisterin selaamiseen ja tutkit sekä COM-objektia että ActiveX-kontrollia.

► OLE/COM Object Viewerin avaaminen

1. Avaa Visual Studio.
2. Valitse **Tools**-valikosta **OLE/COM Object Viewer**.
3. Valitse **View**-valikosta **Expert Mode**. Selaa sitten Viewerin vasemmassa reunassa olevaa kansioluetteloa kunnes löydät kansion **All Objects**.
4. Napauta tämän kansion vasemmalla puolella olevaa plusmerkkiä, jolloin saat nähtäväksesi luettelon kaikista COM-objekteista ja ActiveX-kontrolleista, jot-ka on kirjattu rekisteriin. Osa merkinnöistä on GUID-muodossa. Toisissa käytetään merkkijonomuotoa.

► COM objektien selaaminen

1. Selaa luetteloa, kunnes löydät merkinnän **ADODB.Connection**. Tämä on sama **ADO Connection** -objekti, jota käytetään Laboratoriossa 10.
2. Valitse **ADO Connection** -objekti. Objektin rekisterimerkintöjä koskevat tiedot ilmestyvät ikkunan oikeanpuoleiseen osaan. Huomaa **InprocServer32**-merkintä, joka sisältää polun objektin toteutuksen sisältävään DLL-tiedostoon.
3. Napauta tämän kansion vasemmalla puolella olevaa plusmerkkiä, jolloin saat esille luettelon kaikista tämän COM-objektin tukemista rajapinnoista. Viewerin tulisi näyttää samalta kuin kuvassa 8.13.



Kuva 8.13 OLE/COM Object Viewer

Tässä luettelossa on useita rajapintoja. **IUnknown**-rajapinnan avulla voidaan käyttää suoria osoittimia. **IDispatch**-rajapinnan avulla, skriptikielet kuten VBScript, voivat hyödyntää COM-objektia. Rajapinta, jota asiakkaat käyttävät useimmin, on **_Connection**-rajapinta. Tämän rajapinnan metodien avulla voidaan muodostaa yhteys tietolähteeseen.

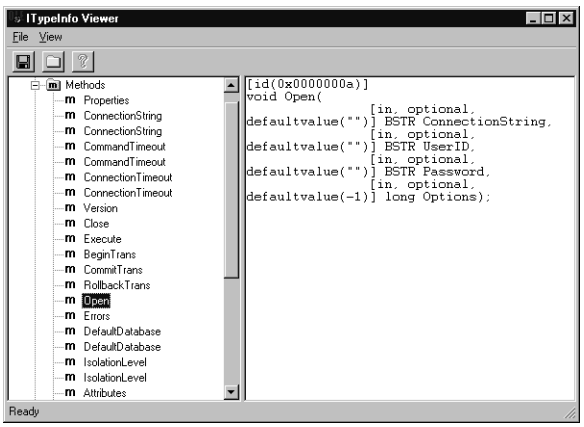
4. Napauta **_Connection**-rajapintaa. Rajapinnan tiedot ilmestyvät ikkunan oikeanpuoleiseen osaan.

   Jos katsot **CLSID**-merkintää ikkunan oikeanpuoleisessa osassa näet, että **InprocServer32** merkintä on *oleaut32.dll* — Windows Automaation etähallinta. Automaation etähallinan käyttäminen tämän rajapinnan etähallinnassa merkitsee sitä, että välitettäessä argumentteja tälle rajapinnalle ja sen paluuarvojen vastaanottamiseen on käytettävä Automaatio-yhteensopivia tyyppejä (tyyppejä, jotka voidaan sijoittaa **VARIANT**-rakenteeseen).

   TypeLib-merkinnän alta löydät **win32**-avaimen, jonka arvoksi on asetettu "C:\Program Files\ Common Files\ADO\MSADO15.DLL". Tästä nähdään, että tyyppikirjasto on linkitetty palvelin DLL:än.

   Kaksoisnapauta **_Connection**-rajapintaa. **Default Interface Viewer** -dialogi tulee näkyviin.

5. Valitse **Default Interface Viewer**-dialogista **View Type Info**.
6. Napauta **ITypeInfo Viewer** -dialogissa Methods-kansion vasemmalla puolella olevaa plusmerkkiä.
7. Selaa metodiluetteloa, kunnes löydät **Open**-metodin.
8. Napauta **Open**-metodia.
9. Ikkunan oikeanpuoleisessa osassa näet yksityiskohtaisia tietoja metodista, kuten kuvassa 8.14.



Kuva 8.14 ITypeInfo Viewer

Ensimmäinen merkintä viittaa dispatch tunnisteeseen. Sitä seuraavat asiakassovelluksen tälle metodille välittämät argumentit. Huomioi Automaatio-yhteensopivien tietotyyppien käyttö.

10. Sulje **ITypeInfo Viewer** -dialogi.
11. Napauta **Default Interface Viewer** -dialogissa **Close**.
12. Napauta **ADODB.Connection**-merkinnän vasemmalla puolella olevaa miinusmerkkiä OLE/COM Object Viewerissä.

► **ActiveX-kontrollin tutkiminen**

1. Selaa merkintöjä OLE/COM Object Viewerissä, kunnes löydät kohdan **Microsoft ADO Data Control**.
2. Napauta merkintää **Microsoft ADO Data Control**. Näet Viewerin oikeassa reunassa tätä kontrollia koskevat tiedot.
3. Näet CLSID alla kohdassa **InprocServer32** määrittelyn "[system directory]\MSADODC.OCX". ActiveX-kontrollit sijoitetaan tavallisesti tiedostoihin, joiden tarkennin on .ocx. (Nämä tiedostot ovat kuitenkin DLL:iä.)
4. Näet Viewerin oikeassa reunassa avaimen **Control**. Tämän avaimen olemassaolo on vahva viite siihen, että tutkit ActiveX-kontrollia.
5. Napauta Viewerin vasemmassa reunassa **Microsoft ADO Data Control** -merkinnän vasemmalla puolella olevaa miinusmerkkiä.
6. Näet useita tämän merkinnän tukemia rajapintoja. Listassa luetellaan kaikki rajapinnat, joita useimpien ActiveX-kontrollien oletetaan tukevan.
7. Kaksoisnapauta **IAdodc**-rajapintaa, joka on **Microsoft ADO Data Control** -merkinnän alla.
8. Valitse **Default Interface Viewer** -dialogista **View Type Info**.
9. Napauta Methods-kansion oikealla puolella olevaa plusmerkkiä **ITypeInfo Viewer** -dialogissa, jolloin Methods-kansio avautuu.
10. Valitse mikä tahansa Methods-kansion metodeista. Rajapinnan tunniste ja metodin kutsurajapinta tulevat näkyviin **ITypeInfo Viewer** -dialogin oikeaan reunaan.
11. Sulje **ITypeInfo Viewer** -dialogi.
12. Valitse **Default Interface Viewer** -dialogista **Close**.
13. Sulje OLE/COM Object Viewer.

Kertaus

1. Mikä on COM-rajapinta?

2. Mikä on GUID, ja mikä on sen tehtävä COM:ssa?

3. Miten prosessin sisäinen palvelin, paikallinen palvelin- ja etäpalvelin-COM-objektit poikkeavat toisistaan?

4. Kuinka standardi etähallinta toteutetaan?

5. Mikä on tyyppikirjasto ja kuinka sitä käytetään?

6. Mitkä ovat seuraukset COM-objektin määrittämisestä free-threaded-malliseksi?

7. Mitä asioita tulee ottaa huomioon ActiveX-kontrollia toteutettaessa?