18

Koosteet

Tässä luvussa kerron koosteiden käytön suurimmista eduista eli .NET-komponenttiesi paketoinnista ja versioinnista. Näet myös, miten luot yhden tiedoston ja monen tiedoston koosteen käyttämällä al.exe-apuohjelmaa (Assembly Generation), miten teet jaettuja koosteita käyttämällä sn.exe-apuohjelmaa (Strong Name), miten selailet paikallista koostevarastoa (assembly cache) käyttämällä laajennusohjelmaa shfusion.dll (Assembly Cache Viewer) ja miten muokkaat koostevarastoa gacutil.exe-apuohjelmalla (Global Assembly Cache). Lopuksi tutkimme muutamaa esimerkkiohjelmaa ja katsomme, mitä kaikki hössötys versioinnista tarkoittaa ja miten .NETin versiointikäytäntö auttaa sinua välttämään "DLL-helliä".

Johdanto koosteisiin

Luvussa 16, "Metadatan kyseleminen reflection-metodien avulla," kuvattiin koostetta fyysiseksi tiedostoksi, joka koostuu yhdestä tai useammasta .NET-kääntäjän generoimasta portable executable (PE)-tiedostosta. Kuvaus kelpasi siinä yhteydessä. Koosteet ovat kuitenkin monimutkaisempia. Tässä täydellisempi määritelmä: kooste on luettelon (manifest), yhden tai useamman modulin ja mahdollisesti yhden tai useamman resurssin paketti. Koosteiden avulla voit ryhmitellä yhteensopivat toiminnalliset yksiköt yhteen tiedostoon jakelua, versiointi ja ylläpitoa ajatellen.

Kaikki PE-tiedostot, jotka käyttävät .NETin ajonaikaista ympäristöä, koostuvat koosteesta tai joukosta koosteita. Kun käännät sovelluksen käyttäen C#-kääntäjää, luot itse asiassa koosteen. Et välttämättä huomaa sitä, paitsi jos yrität sijoittaa useita moduleja yhteen koosteeseen tai yrität hyödyntää joitakin koosteperusteisia ominaisuuksia, kuten versiointia. On kuitenkin tärkeää huomata, että joka kerta, kuin teet EXEn tai DLLn (käyttäen */t:library*-valitsinta), luot koosteen, joka sisältää luettelon, joka puolestaa kuvaan

koosteen .NETin ajonaikaiselle ympäristölle. Lisäksi voit luoda modulin (käyttäen */t:module*-valitsinta), joka on itse asiassa DLL (jonka tarkenne on .netmodule) ilman luetteloa. Toisin sanoen, vaikka loogisesti se on yhä DLL, se ei kuulu koosteeseen ja se pitää lisätä koosteeseen joko */addmodule*-valitsimella, kun sovellus käännetään tai käyttämällä al.exe-apuohjelmaa. Näet miten tämä tehdään myöhemmin tämän luvun kappaleessa "Koosteiden luominen"

Luettelon tiedot

Koosteen luettelo voidaan tallentaa eri tavoin. Jos käännät yksinään toimivaa sovellusta tai DLL:ää, luettelo yhdistetään lopputuloksena syntyneeseen PE:hen. Tämä on nimeltään yhden tiedoston kooste (single-file assembly). Voit generoida myös monen tiedoston koosteen (multifile assembly), jolloin luettelo on joko erillisenä osasena koosteen sisällä tai sitten liitettynä johonkin koosteen moduliin.

Koosteen määrittely riippuu myös suuresti sen käytöstä. Asiakasohjelman kannalta kooste on nimetty ja versioitu kokoelma moduleita, käytettäviä tyyppejä ja mahdollisesti resursseja. Koosteen tekijän kannalta kooste on tapa pakata toisiinsa liittyviä moduleja, tyyppejä ja resursseja ja näyttää niistä asiakkaalle vain tarpeelliset. Tässä mielessä luettelo on se, joka toimii erottimena koosteen yksityiskohtaisen toteutuksen ja sen, mitä asiakkaan on mahdollista käyttää, välillä. Seuraavassa luettelo tiedoista, jotka tallennetaan koosteen luetteloon:

- **Koosteen nimi** Koosteen selväkielinen nimi.
- **Versiointitiedot** Tämä merkkijono sisältää neljä erillistä osaa, jotka muodostavat versionumeron. Osat ovat: pääversionumero (major), aliversionumero (minor), julkaisu (revision) ja käännös (build).
- **Jaettu nimi (valinnainen) ja allekirjoitustunnus** Tämä tieto liittyy koosteen jakeluun ja sitä käsitellään tarkemmin tämän luvun kappaleessa "Koosteen jakelu."
- **Tiedostot** Tämä luettelo sisältää kaikki koosteen sisältämät tiedostot.
- **Viitatut koosteet** Tämä on luettelo kaikista ulkoisista koosteista, joihin viitataan suoraan koosteesta.
- **Tyypit** Tämä luettelo sisältää kaikki koosteen tyypit ja tiedon siitä, mistä modulista se löytyy. Tämä on se tieto, joka auttaa luvun 16 reflection-metodien esimerkkiä (joka kävi läpi kaikki koosteen tyypit) toimimaan niin nopeasti.

- **Turvaoikeudet** Tämä on luettelo turvaoikeuksista, jotka kooste nimenomaisesti kieltää.
- **Omat attribuutit** Luku 8, "Attribuutit," kuvasi omien attribuuttien luomisen. Tyyppien tapaan omat attribuutit tallennetaan koosteen luetteloon nopean käsittelyn takia.
- **Tuotetiedot** Näihin tietoihin kuuluvat yhtiö, tuotemerkki, tuote ja tekijänoikeus-arvot.

Koosteiden edut

Koosteet tarjoavat ohjelmoijalle useita helpottavia ominaisuuksia, esimerkiksi pakkaukset, jakelun ja versioinnin.

Koosteen pakkaaminen

Mahdollisuus pakata useita moduleja yhteen fyysiseen tiedostoon parantaa suorituskykyä. Kun luot sovelluksen ja jakelet sen monen käyttäjän koosteena, .NETin ajonaikaisen ympäristön pitää ladata vain kulloinkin tarvittavat modulit. Tämä pienentää sovelluksen tarvitsemaa muistia.

Koosteen jakelu

Ohjelmien pienin jakeluyksikkö .NETissä on kooste. Kuten aiemmin mainitsin, voit luoda .NET-modulin kääntäjän */t:module*-valitsimella, mutta sinun pitää sisällyttää moduli johonkin koosteeseen, jos haluat jakaa sen. Lisäksi, vaikka on houkuttelevaa sanoa, että koosteet ovat tapa sovelluksen jakeluun, se ei teknisesti ole totta. On täsmällisempää kuvata koosteita .NETissä luokan jakeluksi (kuten DLL:t Win32:ssa), jolloin yksi sovellus voi muodostua useasta koosteesta.

Koska koosteet ovat itsensä kuvaavia, helpoin jakelutapa on kopioida kooste haluttuun kansioon. Kun sitten yrität käynnistää koosteen sisältämän sovelluksen, luettelo kertoo .NETin ajonaikaiselle ympäristölle koosteen sisältämät modulit. Lisäksi kooste sisältää viittaukset ulkoisiin koosteisiin, joita sovellus tarvitsee.

Yleisin jakelutapa on käyttää *yksityisiä koosteita* (private assembly), jossa koosteet kopioidaan kansioon, joka ei ole jaettu. Miten määrittelet yksityisen koosteen? Se on oletus ja kooste on automaattisesti yksityinen, jos et erikseen tee siitä *jaettua koostetta* (shared

assembly). Koosteiden jakaminen vaatii hieman enemmän työtä ja käsittelen sen myöhemmin tässä luvussa kappaleessa "Jaetun koosteen tekeminen."

Koosteen versiointi

Toinen merkittävä etu koosteista on sen sisäänrakennettu versiointi (eli loppu DLL-hellille). DLL-hell tarkoittaa tilannetta, jossa sovellus korvaa toisen sovelluksen tarvitseman DLL:n, yleensä saman DLL:n aiemman version, jolloin vanhaa versiota käyttänyt sovellus menee rikki. Vaikka Win32 resurssitiedoston muoto mahdollistaa resurssin versioinnin, käyttöjärjestelmä ei tarkista mitään versiointisääntöjä. Vastuu jää yksinomaan sovelluksen ohjelmoijille.

Tämän ongelman ratkaisuksi koosteen luettelo sisältää koosteen versiointitietojen lisäksi luettelon kaikista sen tarvitsemsta koosteista sekä niiden versiotiedot. Tämän rakenteen ansiosta .NETin ajonaikainen ympäristö voi varmistaa, että versiointikäytännöstä pidetään kiinni ja sovellukset pysyvät toimintakunnossa, vaikka järjestelmään asennetaan uudempi, yhteensopimaton versio jaetusta DLL:stä. Koska versiointi on koosteiden paras ominaisuus, se käsitellään yksityiskohtaisesti useiden esimerkkien kanssa kappaleessa "Koosteiden versiointi."

Koosteiden tekeminen

Jos teet DLL:n kääntäjän */t:library*-valitsimella, et pysty lisäämään sitä toiseen koosteeseen. Tämä johtuu siitä, että kääntäjä generoi automaattisesti DLL:lle luettelon ja siksi DLL itsessään on kooste. Katsotaan tästä seuraavaa esimerkkiä. Meillä on DLL (Module1Server.cs), joka sisältää tyhjän tyypin nimeltä *Module1Server*.

```
// Module1Server.cs
// tehty seuraavalla kääntäjän valitsimella
//     csc /t:library Module1Server.cs
public class Module1Server
{
}
```

Tähän DLL:ään viittaa asiakasohjelma (Module1Client.cs):

```
// Module1ClientApp.cs
// tehty seuraavalla kääntäjän valitsimella
//     csc Module1ClientApp.cs /r:Module1Server.dll
using System;
using System.Diagnostics;
using System.Reflection;
```

```
class Module1ClientApp
{
    public static void Main()
    {
        Assembly DLLAssembly = Assembly.GetAssembly(typeof(Module1Server));
        Console.WriteLine("Module1Server.dll Assembly Information");
        Console.WriteLine("\t" + DLLAssembly);

        Process p = Process.GetCurrentProcess();
        string AssemblyName = p.ProcessName + ".exe";
        Assembly ThisAssembly = Assembly.LoadFrom(AssemblyName);
        Console.WriteLine("Module1Client.exe Assembly Information");
        Console.WriteLine("\t" + ThisAssembly);
    }
}
```

Käännetään nyt nämä kaksi modulia seuraavilla valitsimilla:

```
csc /t:library Module1Server.cs
csc Module1ClientApp.cs /r:Module1Server.dll
```

Koodin suorittaminen tässä vaiheessa aiheuttaa seuraavat tulokset ja todistaa, että sekä EXE että DLL ovat olemassa omissa erillisissä koosteissaan:

```
Module1Server.dll Assembly Information
    Module1Server, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
Module1Client.dll Assembly Information
    Module1Client, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
```

Itse asiassa, jos muuttaisit *Module1Server*-luokan käsittelymääreen arvosta *public* arvoon *internal*, asiakaskoodi ei kääntyisi, koska määrityksen mukaan käsittelymääre *internal* tarkoittaa, että kyseinen tyyppi on vain muiden samassa koosteessa olevien koodien käytettävissä.

Useita moduleja sisältävän koosteen tekeminen

Voit sijoittaa molemmat esimerkkimme modulit samaan koosteeseen kahdella eri tavalla. Ensimmäinen tapa on muuttaa kääntäjän valitsimia. Tässä esimerkki:

```
// Module2Server.cs
// tehty seuraavalla kääntäjän valitsimella
//     csc /t:module Module2Server.cs
internal class Module2Server
{
}
```

Huomaa, että nyt voimme käyttää käsittelymäärettä *internal*, jolloin koodia voidaan käsitellä vain koosteen sisältä.

```
// Module2ClientApp.cs
// tehty seuraavalla kääntäjän valitsimella
//     csc /addmodule:Module2Server.netmodule Module2ClientApp.cs
using System;
using System.Diagnostics;
using System.Reflection;

class Module2ClientApp
{
    public static void Main()
    {
        Assembly DLLAssembly =
            Assembly.GetAssembly(typeof(Module2Server));
        Console.WriteLine("Module1Server.dll Assembly Information");
        Console.WriteLine("\t" + DLLAssembly);

        Process p = Process.GetCurrentProcess();
        string AssemblyName = p.ProcessName + ".exe";
        Assembly ThisAssembly = Assembly.LoadFrom(AssemblyName);
        Console.WriteLine("Module1Client.dll Assembly Information");
        Console.WriteLine("\t" + ThisAssembly);
    }
}
```

Module2Server.cs ja Module2Client.exe on käännetty seuraavasti:

```
csc /t:module Module2Server.cs
csc /addmodule:Module2Server.netmodule Module2Client.cs
```

Sinun pitää ensimmäiseksi poistaa /r-valitsin, koska sitä käytetään vain viittauksissa koosteisiin ja nyt molemmat modulit sijaitsevat samassa koosteessa. Sitten sinun pitää lisätä */addmodule*-valitsin, joka kertoo kääntäjälle, mitkä modulit lisätään luotavaan koosteeseen.

Sovelluksen kääntäminen ja suorittaminen tuottaa seuraavat tulokset:

```
Module1Server.dll Assembly Information
    Module2Client, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
Module1Client.dll Assembly Information
    Module2Client, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
```

Toinen tapa koosteen tekemiseen on käyttää al.exe-työkalua (Assembly Generator). Se ottaa syötteenään yhden tai useamman tiedoston, jotka ovat joko .NET-moduleita (sisältäen MSIL:n) tai resurssi- tai kuvatiedostoja. Tulostiedosto on luettelon sisältävä kooste. Sinun tulee käyttää al.exe-ohjelmaa esimerkiksi silloin, kun sinulla on useita DLL:iä ja haluat toimittaa ja versioida ne yhtenä yksikkönä. Oletetaan, että DLL:si ovat nimeltään

A.DLL, B.DLL ja C.DLL. Teet nämä kolme DLL:ää sisältävän koosteen al.exe-ohjelmalla seuraavasti:

```
al /out:COMPOSITE.DLL A.DLL B.DLL C.DLL
```

Jaetun koosteen tekeminen

Jaettu kooste tehdään, kun koostetta on tarkoitus käyttää useassa sovelluksessa ja versiointi on tärkeää. (Puhumme versioinnista seuraavassa kappaleessa.) Jotta voit jakaa koosteen, sinun pitää tehdä koosteelle *jaettu nimi* (shared name, joskus myös strong name) käyttämällä sn.exe-apuohjelmaa (Strong name), joka kuuluu .NET SDK:hon. Neljä etua, jotka jaetun nimen käyttämisestä seuraa, ovat:

- Se on .NETin menetelmä yksilöllisen nimen (globally unique name) generoimiseksi.
- Koska generoitu avainpari (selvitetään kohta) sisältää allekirjoituksen, pystyt selvittämään, onko koostetta peukaloitu sen alkuperäisen luonnin jälkeen.
- Jaettu nimi takaa, että kolmas osapuoli ei voi julkaista uudempaa versiota koosteestasi. Tämäkin johtuu allekirjoituksesta, sillä kolmannella osapuolella ei ole käytössään yksityistä avaintasi.
- Kun .NET lataa koosteen, ajonaikainen ympäristö voi tarkistaa, että kooste tulee tekijältä, jolta kutsuja olettaakin sen tulevan.

Ensimmäinen vaihe jaetun nimen luomisessa on käyttää sn.exe-apuohjelmaa ja luoda koosteelle avaintiedosto. Se tehdään *-k*-valitsimella ja antamalla sen tiedoston nimi, joka tulee sisältämään generoidun avaimen. Seuraavalla komentorivikäskyllä teemme avaintiedoston (InsideCSharp.key).

```
sn -k InsideCSharp.key
```

Tämän suorittaminen antaa seuraavanlaisen ilmoituksen:

```
Key pair written to InsideCSharp.key
```

Nyt lisätään *assembly:AssemblyKeyFile*-attribuutti lähdetiedostoon. Olen seuraavassa luonut yksinkertaisen joukon tiedostoja esitelläkseni tätä vaihetta:

```
// Module3Server.cs
// tehty seuraavalla kääntäjän valitsimella
//     csc /t:module Module3Server.cs
internal class Module3Server
{
}
```

(jatkuu)

```
// Module3ClientApp.cs
// tehty seuraavalla kääntäjän valitsimella
//     csc /addmodule:Module3Server.netmodule Module3ClientApp.cs
using System;
using System.Diagnostics;
using System.Reflection;

[assembly:AssemblyKeyFile("InsideCSharp.key")]

class Module3ClientApp
{
    public static void Main()
    {
        Assembly DLLAssembly =
            Assembly.GetAssembly(typeof(Module3Server));
        Console.WriteLine("Module1Server.dll Assembly Information");
        Console.WriteLine("\t" + DLLAssembly);

        Process p = Process.GetCurrentProcess();
        string AssemblyName = p.ProcessName + ".exe";
        Assembly ThisAssembly = Assembly.LoadFrom(AssemblyName);
        Console.WriteLine("Module1Client.dll Assembly Information");
        Console.WriteLine("\t" + ThisAssembly);
    }
}
```

Kuten näet, *assembly:AssemblyKeyFile*-attribuutin muodostin ottaa parametrikseen avaintiedoston, joka muodostettiin sn.exe-apuohjelmalla ja tällä tavalla määrittelet avainparin, jota käytetään antamaan koosteellesi jaettu nimi. On tärkeää ymmärtää, että tämä attribuutti on koostetason attribuutti. Siksi se voidaan teknisesti sijoittaa mihin tahansa koosteen tiedostoon eikä sitä ole liitetty johonkin määrättyyn luokkaan. On kuitenkin tavallista sijoittaa tämä attribuutti heti *using*-määreiden alapuolelle ennen luokkien määrittelyä.

Kun nyt suoritat sovelluksen, huomaa koosteen *PublicKeyToken*-arvo. Se oli *null* kahdessa edeltävässä esimerkissä, koska nuo koosteet olivat yksityisiä. Nyt kun kooste on määritelty jaetuksi, koosteeseen on yhdistetty julkinen avain.

```
Module3Server.dll Assembly Information
    Module3Client, Version=0.0.0.0, Culture=neutral,
    PublicKeyToken=6ed7cef0c0065911
Module3Client.dll Assembly Information
    Module3Client, Version=0.0.0.0, Culture=neutral,
    PublicKeyToken=6ed7cef0c0065911
```

Tässä esimerkissä instantioimamme *Assembly*-objekti on jaettu. Mutta mistä tiedämme, mitkä .NET-järjestelmässä olevat koosteet ovat jaettuja? Vastaus on yleinen koostevarasto (global assembly cache). Seuraavassa kappaleessa kerron tästä .NETin osasta ja selvitän, millaista roolia se esittää jaetuissa koosteissa.

Yleisen koostevaraston käsittely

Jokaisessa .NET-järjestelmässä on *yleinen koostevarasto* (global assembly cache). Se palvelee kolmea päätarkoitusta:

- Sitä käytetään tallennuspaikkana Internetistä tai muilta palvelimilta (sekä http- että tiedostopalvelimilta) ladatuille koodeille. Määrättyä sovellusta varten ladatut koodit tallennetaan varaston yksityiseen osaan. Tämä estää muita käsittelemästä niitä.
- Se on tietovarasto useiden .NET-sovellusten käyttämille jaetuille komponenteille. Koosteet asennetaan varastoon käyttämällä gacutil.exe-apuohjelmaa ja ne sijoitetaan varaston yleiseen osaan, jossa niitä voivat käyttää kaikki koneen sovellukset.
- Yksi usein kuulemani kysymys kuuluu: "Minne täsmäkääntäjän kääntämä koodi tallennetaan, jotta sitä ei tarvitse kääntää uudelleen?" Nyt tiedät vastaukset: koosteiden konekieliset versiot, jotka on esikäännetty, tallennetaan koostevarastoon.

Koostevaraston tarkastelu

Katsotaan nyt varastosta sinne asennetut koosteet ja jaetut koosteet. Käytä Resurssienhallintaa ja avaa c:\winnt\assembly-kansio. Helpottaakseen koosteiden kiinnostavien tietojen selailua, .NET sisältää Assembly Cache Viewer (shfusion.dll)-nimisen laajennuksen. Sen avulla voit selata koosteen tietoja, kuten versionumeroa, kulttuuria, julkista avainta ja jopa sitä, onko kooste esikäännetty.

Toinen tapa varaston tarkasteluun on käyttää gacutil.exe-työkalua. Sen avulla voit tehdä muutamia perustehtäviä määrittelemällä jonkin seuraavista, toisensa poissulkevista, valitsimista.

- ***-i*** Tämä valitsin asentaa koosteen yleiseen koostevarastoon. Seuraavassa esimerkki:

```
gacutil –i HelloWorld.DLL
```

Näet kohta, miten lisätään *Module3Client*-kooste koostevarastoon tällä valitsimella.

- ***-u*** Tämä valitsin poistaa koosteen yleisestä koostevarastosta mukaan lukien versiotiedot. Jos et määrittele versiotietoa, kaikki määrätyn nimiset koosteet poistetaan. Siksi seuraavassa ensimmäinen esimerkki poistaa *HelloWorld*-koosteet riippumatta niiden versionumerosta ja toinen esimerkki poistaa vain määrätyn verion:

```
gacutil –u HelloWorld
gacutil –u HelloWorld, ver=1,0,0,0
```

- ***-l*** Tämä valitsin tuottaa luettelon yleisen koostevaraston sisällöstä. Luettelossa ovat kooosteen nimi, sen versionumero, sen sijainti ja sen jaettu nimi.

> Huomaa Joissakin aiemmissa .NETin beta-versioissa huomasin ongelman, kun yritin selata c:\winnt\assembly-kansiota eikä laajennusohjelma käynnistynyt. Syy oli se, että shfusion.dll ei ollut rekisteröitynyt kunnolla. Jos sinulla tapahtuu näin, avaa DOS-ikkuna ja siirry kansioon c:\winnt\Microsoft.net\framework\v*XXX,* jossa XXX tarkoittaa käyttämäsi .NET Frameworkin versionumeroa. Koska käytän beta-versiota, saattaa kansion nimi vielä muuttua ennen lopullisen tuotteen julkistusta. Joka tapauksessa etsi tiedosto shfusion.dll ja siirry sen kansioon. Rekisteröi sitten shfusion.dll. Omalla koneellani se tapahtui näin:
>
> c:\winnt\microsoft.net\framework\v1.0.2615>regsvr32 shfusion.dll

Nyt kun olemme luoneet avaintiedoston ja liittäneet sen koosteeseen, lisätään kooste yleiseen koostevarastoon. Teet sen seuraavalla komentorivikäskyllä:

```
gacutil -i Module3ClientApp.exe
```

Jos kaikki menee hyvin, saat seuraavan ilmoituksen:

```
Assembly successfully added to the cache
```

Tässä vaiheessa voit käyttää *gacutil -l* -käskyä tulostaaksesi koostevaraston koosteet ja tarkistaaksesi, onko *Module3Client* siellä tai voit käyttää Assembly Cache Viewer (shfusion.dll) -laajennusta. Käytetään jälkimmäistä. Jos avaat varaston Resurssienhallinnassa

(*C:\Winnt\Assembly* tai *C:\Windows\Assembly*), sinun pitäisi nähdä *Module3Client* yhdessä muiden koosteiden kanssa. Napauta sitä kakkospainikkeella ja valitse Properties-kohta, jolloin näet julkisen avaimen, versionumeron ja koosteen fyysisen sijainnin levylläsi. Julkinen avaimesi on tietenkin eri kuin minun, mutta oleellista tässä on se, että se on sama kuin näkyi Module3ClientApp-sovelluksen tulosteessa.

Koosteiden versiointi

Koosteen luettelo sisältää versionumeron sekä luettelon kaikista koosteen viittaamista koosteista versiotietoineen. Kuten kohta näet, versionumerot on jaettu neljään osaan seuraavalla tavalla (<pääversionumero><aliversionumero><julkaisu><käännös>):

```
<major><minor><build><revision>
```

Ohjelman suorituksen aikana .NETin ajonaikainen ympäristö käyttää tätä versionumeroa päättäessään, mitä koosteen versiota se käyttää sovelluksessa. Kuten kohta näet, oletustoiminto (nimeltään versiointikäytäntö) on se, että kun sovellus on asennettu, .NET käyttää automaattisesti uusinta versiota sovelluksen viittaamista koosteista, jos versionumero täsmää pääversionumeron ja aliversionumeron osalta. Voit muuttaa tämän oletuskäytännön asetustiedostoilla.

Versiointi kuuluu vain jaettuihin koosteisiin, sillä sitä ei sovelleta yksityisiin koosteisiin. Versiointi on ehkä tärkein tekijä, kun päätetään luoda ja jakaa koosteita. Katsotaan siksi muutamia esimerkkikoodeja, joista selviää miten tämä kaikki toimii ja miten toimitaan koosteiden versioinnin kanssa.

Esimerkki, jota aion käyttää, on yksinkertaistettu versio .NET SDK:n mukana tulevasta versiointiesimerkistä, sillä siihen ei sisälly Windows Forms -osuutta, koska haluan keskittyä versiointiin ja siihen, miten ajonaikainen ympäristö huolehtii siitä.

Minulla on kaksi suoritettavaa tiedostoa, jotka esittävät laskutuspaketteja nimeltään Personal ja Business. Molemmat näistä sovelluksista käyttävät jaettua koostetta nimeltä *Account*. Ainoa toiminnallisuus *Account*-luokassa on versiotietojen ilmoittaminen, jotta voimme varmistaa, että sovelluksemme käyttää aiottua luokan versiota. Sovellus sisältää useita versioita *Account*-luokasta, jotta voit nähdä itse, miten versiointi toimii oletuksena ja miten XML:ää käytetään luotaessa yhteys sovelluksen ja määrätyn koosteen version välille.

Luo aluksi kansio nimeltä Accounting. Luo tähän kansioon avaintiedosto, jota tulevat käyttämään kaikki *Account*-luokan versiot. Avaintiedosto tehdään seuraavalla Accounting-kansiossa annettavalla komentorivin käskyllä:

```
sn /k account.key
```

Kun avaintiedosto on tehty, luo Accounting-kansioon kansio nimeltä Personal. Personal-kansioon luo Personal.cs-niminen tiedosto, jonka sisältö näyttää seuraavalta:

```
// Accounting\Personal\Personal.cs
using System;

class PersonalAccounting
{
    public static void Main()
    {
        Console.WriteLine
            ("PersonalAccounting calling Account.PrintVersion");
        Account.PrintVersion();
    }
}
```

Luo samaan Personal-kansioon uusi kansio nimeltä Account1000. Tämä kansio tulee sisältämään ensimmäisen version *Account*-luokasta. Kun olet tehnyt sen, tee kansioon seuraava Account.cs-niminen tiedosto:

```
// Accounting\Personal\Account1000\Account.cs
using System;
using System.Reflection;

[assembly:AssemblyKeyFile("..\\..\\Account.key")]
[assembly:AssemblyVersion("1.0.0.0")]
public class Account
{
    public static void PrintVersion()
    {
        Console.WriteLine
            ("This is version 1.0.0.0 of the Account class");
    }
}
```

Kuten näet, käytin *AssemblyeKeyFile* ja *AssemblyVersion* -attribuutteja osoittamaan C#-kääntäjälle aiemmin luodun avaintiedoston nimen ja *Account*-luokan version. Luo nyt Account DLL seuraavasti:

```
csc /t:library account.cs
```

Kun *Account*-luokka on luotu, se pitää lisätä yleiseen koostevarastoon:

```
gacutil –i Account.dll
```

Jos haluat, voit varmistaa, että *Account*-kooste on todella koostevarastossa. Siirry nyt Personal-kansioon ja luo sovellus näin:

```
csc Personal.cs /r:Account1000\Account.dll
```

Sovelluksen suorittaminen näyttää seuraavanlaiset tulokset:

```
PersonalAccounting calling Account.PrintVersion
This is version 1.0.0.0 of the Account class
```

Tähän mennessä emme ole tehneet mitään uutta. Katsotaan nyt, mitä tapahtuu, kun asennetaan toinen sovellus, joka käyttää *Account*-luokan uudempaa versiota.

Tee Accounting-kansioon uusi kansio nimeltä Business ja siihen uusi kansio nimeltä Account1001 esittämään *Account*-luokan uutta versiota. Luokka sijaitsee tiedostossa nimeltä Account.cs ja se on melkein samanlainen kuin edellinen versio.

```
// Accounting\Business\Account1001\Account.cs
using System;
using System.Reflection;

[assembly:AssemblyKeyFile("..\\..\\Account.key")]
[assembly:AssemblyVersion("1.0.0.1")]
public class Account
{
    public static void PrintVersion()
    {
        Console.WriteLine
            ("This is version 1.0.0.1 of the Account class");
    }
}
```

Kuten ennenkin, luo tämä *Account*-luokan versio seuraavilla komentorivikäskyillä:

```
csc /t:library Account.cs
gacutil –i Account.dll
```

Tässä vaiheessa sinulla pitäisi olla yleisessä koostevarastossa kaksi versiota *Account*-luokasta. Tee nyt Accounting\Business-kansioon seuraavanlainen Business.cs-tiedosto:

```
// Accounting\Business\Business.cs
using System;

class PersonalAccounting
{
    public static void Main()
    {
        Console.WriteLine
            ("BusinessAccounting calling Account.PrintVersion");
        Account.PrintVersion();
    }
}
```

Muodosta Business-sovellus seuraavalla komentorivikäskyllä:

```
csc business.cs /r:Account1001\Account.dll
```

Sovelluksen suorittaminen tuottaa seuraavan tuloksen, joka osoittaa, että Business-sovellus käyttää *Account*-koosteen versiota 1.0.0.1.

```
BusinessAccounting calling Account.PrintVersion
This is version 1.0.0.1 of the Account class
```

Suorita nyt Personal-sovellus uudelleen ja katso mitä tapahtuu!

```
PersonalAccounting calling Account.PrintVersion
This is version 1.0.0.1 of the Account class
```

Sekä Personal että Business -sovellukset käyttävät *Account*-koosteen uusinta versiota. Miksi? Syy löytyy .NETin ominaisuudesta Quick Fix Engineering (QFE) ja sen oletuksena noudattamasta versiointikäytännöstä.

QFE ja oletusversiointikäytäntö

Quick Fix Engineering -päivitykset (tai hot fixes) ovat ajoittamattomia korjauksia, jotka tehdään isojen ongelmien pikaiseksi korjaamiseksi. Koska ne eivät yleensä muuta koodin rajapintoja, on vain pieni mahdollisuus, että ne vaikuttavat epäsuotuisasti asiakaskoodiin. Siksi .NETin oletuskäytäntö on liittää kaikki asiakaskoodit automaattisesti uuteen korjattuun koodiversioon, jollei koosteen olemassa olevassa asetustustiedostossa nimenomaisesti liitetä sovellusta määrättyyn koosteen versioon. Koosteen uuden version oletetaan olevan QFE, jos ainoa osa, joka versionumerossa on muuttunut, on julkaisu (revision).

Safe Mode -asetustiedoston tekeminen

Oletusversiointikäytäntö sopii varmaan useimpiin tilanteisiin mutta entäpä, jos sinun pitäisi määritellä, että Personal-sovellus käyttää vain sitä koosteen versiota, jonka kanssa se on toimitettu? Tällöin tarvitset avuksesi XML-muotoista asetustiedostoa. Sen nimi on sama kuin sovelluksen (tunnisteella .cfg) ja se sijaitsee samassa kansiossa. Kun sovellus käynnistetään, sen asetustiedosto luetaan ja .NET käyttää sieltä löytämiään määräyksiä valitessaan sovelluksen kanssa käytettävien koosteiden versioita.

Kun haluat määritellä, että sovelluksen tulee aina käyttää sitä koosteen versiota, jonka kanssa se on toimitettu, määrittelet sovelluksen sidontamuodoksi (binding mode) arvon "*safe*". Puhekielellä sanotaan joskus, että sovellus asetetaan turvattuun muotoon. Luo nyt Accounting/Personal-kansioon tiedosto nimeltä PersonalAccounting.cfg ja kirjoita siihen seuraava teksti. Huomaa erityisesti <*AppBindingMode*>-tagi.

```
<?xml version ="1.0”?>
<Configuration>
<BindingMode>
<AppBindingMode Mode="safe"/>
</BindingMode>
</Configuration>
```

Jos nyt suoritat Personal-sovelluksen uudelleen, saat seuraavat tulokset:

```
PersonalAccounting calling Account.PrintVersion
This is version 1.0.0.0 of the Account class
```

Koosteen määrätyn version käyttäminen

Katsotaan nyt toista yleistä versiointiin liittyvää tarvetta. Esimerkin loppuosassa aiheutamme virheen. Tee Account\Business-kansioon kansio nimeltä Account1002 ja siihen seuraava *Account*-luokka. Tällä kertaa *Account.PrintVersion*-metodin ainoa tarkoitus on aiheuttaa poikkeus ja siten esittää toimimatonta *Account*-luokan päivitystä.

```
// Accounting\Business\Account1002\Account.cs
using System;
using System.Reflection;

[assembly:AssemblyKeyFile(“..\\..\\Account.key”)]
[assembly:AssemblyVersion(“1.0.0.2”)]
public class Account
{
    public static void PrintVersion()
    {
        // Tämä esittää koodivirhettä.
        throw new Exception();

        Console.WriteLine
            (“This is version 1.0.0.2 of the Account class”);
    }
}
```

Luo nyt Account-kooste seuraavasti:

```
csc /t:library Account.cs
gacutil –i Account.dll
```

Sekä Personal että Business -sovelluksen suorittaminen aiheuttaa nyt käsittelemättömän poikkeuksen ja ohjelman kaatumisen. Tilanne kuvaa aivan liian yleistä ilmiötä ohjelmistoteollisuudessa: asennetaan jotain toimimatonta, joka vioittaa useita muitakin sovelluksia. Personal-sovelluksen kohdalla emme halua palata turvattuun

muotoon, koska haluamme käyttää viimeistä toimivaa Account-luokan versiota (1.0.0.1) ja turvattu muoto taas käyttäisi alkuperäistä vrsiota (1.0.0.0).

Ratkaisu löytyy jälleen sovelluksen asetustiedostosta. Muokkaa nyt Accounting\PersonalAccounting.cfg-tiedostoa niin, että sen XML-tagi näyttää seuraavalta:

```
<?xml version ="1.0”?>
<Configuration>
<BindingPolicy>
<BindingRedir Name="Account”
                  Originator="32ab35a4550339b1”
                  Version="*”
                  VersionNew="1.0.0.1”
                  UseLatestBuildRevision="no"/>
</BindingPolicy>
</Configuration>
```

Huomaa, että avain, joka määritellään *Originator*-attribuutille on oman järjestelmäni avain. Sinun pitää korvata se avaimella, joka generoitiin, kuin loit Account.key-tiedoston. Voit hakea tarvittavan arvon avaamalla koostevaraston ja hakemalla sieltä *Account*-luokan.

Suorita vielä lopuksi Personal-sovellus. Huomaat, että nyt se toimii ilman ongelmia ja käytettävä *Account*-luokan versio on todella haluamamme (1.0.0.1).

Yhteenveto

Tässä luvussa kävin läpi koosteiden käytön suurimmat edut, joita ovat paketointi, jakelu, versiointi ja turvallisuus. Puhuin myös yhden tiedoston ja monen tiedoston koosteista ja al.exe-apuohjelmasta, siitä, miten luodaan ja jaetaan koosteita sn.exe-apuohjelman avulla ja gacutil.exe-työkaluohjelmasta. Näytin myös, miten ohjataan .NETin oletusversiointikäytäntöä XML-muotoisten asetustiedostojen avulla.