




Modulul 7

Servicii și Domenii de aplicație



Overview

- ▶ Crearea și instalarea serviciilor
- ▶ Domenii de aplicații



11/20/2014

2

În cadrul unui proces, codul este executat de către un thread.

Puteți folosi clasele **Microsoft .NET Framework** pentru a crea obiecte ce reprezintă fire de execuție, pentru a executa cod și pentru a sincroniza firele de execuție. Thread-urile rulează în cadrul unui domeniu de aplicație, care este o unitate de izolare în cadrul unui proces .NET Framework.

În acest modul veți învăța cum să folosiți clasele .NET Framework pentru a construi aplicații multithread, cum să realizați servicii Windows și clase pentru instalarea serviciilor, și cum să accesați și să configurați domenii de aplicație.

Obiective

După completarea acestui modul veți fi capabili să:

- ❖ Descrieți și să implementați aplicații multithread prin folosirea namespace-ului **System.Threading**.
- ❖ Creați, instalați și controlați un serviciu Windows prin folosirea namespace-ului **System.ServiceProcess**.
- ❖ Descrieți și creați domenii de aplicație.



Crearea și instalarea serviciilor **Windows**

- ▶ Introducere în serviciile **Windows**
- ▶ Arhitectura serviciilor
- ▶ Clasa **ServiceBase**
- ▶ Accesarea serviciilor
- ▶ Creare **Service Application**
- ▶ Instalarea serviciilor



11/20/2014 3

În acest capitol veți învăța să folosiți clasele .NET Framework pentru a crea servicii și scrieți cod pentru instalarea acestora.

Obiective

❑ După completarea acestui modul veți fi capabili să:

- ❖ Descrieți facilitățile oferite de un serviciu Windows.
- ❖ Descrieți arhitectura lucrului cu thread-uri pentru un serviciu.
- ❖ Implementați un serviciu prin folosirea clasei ServiceBase.
- ❖ Accesați un serviciu folosind clasa **ServiceControl**.
- ❖ Creați un **Service Project**.
- ❖ Creați o clasă **ServiceInstaller** pentru a instala un serviciu.



Introducere în serviciile **Windows**

- ▶ Instalare și dezinstalare folosind **SCM**:
 - ▶ Serviciile pot depinde de alte servicii
 - ▶ Un proces poate conține mai multe servicii
- ▶ Pornite, suspendate sau oprite de **SCM**:
 - ▶ Un serviciu poate fi configurat să pornească automat la pornirea computerului
- ▶ Rulează folosind un cont de securitate diferit de utilizatorul interactiv
 - ▶ Nu are interfață
 - ▶ Suportă comunicarea între procese

11/20/2014 4

Un serviciu **Windows** este un proces fără interfață cu utilizatorul proiectat să ruleze sub un cont de securitate diferit de utilizatorul interactiv. Serviciul poate fi configurat să ruleze atunci când computerul pornește și să fie restartat dacă acesta se închide în mod neașteptat.



Servicii și Securitate

Puteți configura un serviciu să ruleze sub un cont de securitate **.NET Framework**, astfel puteți crea un cont doar cu privilegiile cerute de serviciu. Aceste privilegii pot fi mai multe decât privilegiile unui utilizator interactiv, sau mai puține.

Pornirea unui serviciu

Utilizatorul interactiv nu pornește niciodată un serviciu direct. Serviciul este activat de **Software Configuration Manager (SCM)** folosind informații de configurare din registru. Windows oferă două instrumente în linie de comandă (**net.exe** și **sc.exe**) și o consolă **MMC** (Microsoft Management Console – disponibilă prin intermediul Control Panel), care vă permit să comunicați cu **SCM**. Dacă rulați utilitarul **net** folosind doar parametrul de start, utilitarul listează serviciile care rulează în mod curent. Puteți folosi parametrul de start pentru a furniza numele serviciului de start și parametru de stop pentru a furniza numele serviciului de oprire. Utilitarul **sc.exe** oferă comenzi suplimentare care vă permit să obțineți informații detaliate despre un serviciu.

Cel mai probabil veți folosi **Control Panel** pentru a controla serviciile. Această aplicație listează serviciile instalate pe computerul dumneavoastră; dacă aveți suficiente privilegii vă permite să configurați un anumit serviciu.



Introducere în serviciile **Windows**

- ▶ **SCM** poate trimite comenzi prin care instruește un serviciu să realizeze anumite acțiuni precum pornirea sau oprirea.
- ▶ **SCM** poate trimite și comenzi personalizate.
 - ▶ O comandă personalizată este un număr de 32 de biți pe ambele versiuni de Windows (32 biți sau 64 biți).
- ▶ Comunicare aplicație utilizator – serviciu:
 - ▶ comenzi SCM
 - ▶ comunicare între procese


11/20/2014 5

Există câteva puncte importante la care trebuie să fiți atenți:

- ❖ Puteți porni, opri, suspenda sau continua un serviciu; puteți determina dacă un serviciu va porni atunci când computerul pornește.
- ❖ Puteți da parametri de tip string metodei **ServiceMain** prin intermediul căsuței **Start Parameters**.
- ❖ Puteți specifica contul de securitate care va fi folosit; contul poate fi fie un cont **Local System** (cel mai puternic) sau orice alt cont din computer.
- ❖ Puteți selecta serviciul **Allow** pentru a interacționa cu check box-ul desktop-ului, dacă alegeți contul **Local System**.

Comunicarea cu un serviciu


Un serviciu nu are o interfață cu utilizatorul și sunt implementate ca o aplicație Windows fără fereastră. Rularea unei aplicații Windows este de obicei menținută prin intermediul cozii de mesaje a ferestrei principale. Acest lucru nu se întâmplă și cu un serviciu; serviciile sunt menținute de către **SCM**.



Arhitectura serviciilor

Flux:

- ▶ **SCM** pornește procesul
- ▶ Procesul retunează informațiile serviciului
- ▶ **SCM** creează thread-ul pentru serviciu
- ▶ Serviciul returnează informațiile pentru metoda handler



11/20/2014 6



Un proces serviciu conține unul sau mai multe servicii. Un serviciu este activat și controlat de un **SCM** prin folosirea informațiilor din registru. Aplicațiile de utilizator pot comunica cu serviciul prin folosirea unor mesaje de comandă sau prin folosirea unui canal de comunicare între procese.

Arhitectura

Fiecare serviciu are o metodă de început numită **ServiceMain**, în plus față de punctul de început al procesului.

La o cerere de pornire a unui serviciu, **SCM** citește informațiile de configurare din registru. Dacă informațiile de configurare indică faptul că serviciul depinde de alte servicii, **SCM** asigură faptul că toate acele servicii rulează înainte să fie pornit procesul serviciu. Acesta creează un thread pentru metoda **ServiceMain** a serviciilor pe care dorește să le pornească. Prima acțiune a metodei **ServiceMain** este de a informa **SCM-ul** care este metoda Handler. Această metodă preia comenzile de la **SCM**, inclusiv cele de pornire și de oprire a serviciului și comenzi personalizate de la aplicațiile utilizator.

În final, serviciul trebuie să informeze **SCM-ul** pe durata inițializării și să furnizeze informații despre cât de mult va dura această inițializare. Dacă **SCM** nu primește aceste informații de stare, se comportă ca și cum există o problemă cu serviciul și astfel va termina procesul.



Arhitectura serviciilor

- ▶ **Procesul serviciu** – thread principal
 - ▶ rulează punctul de început al procesului (Main)
 - ▶ determină durata procesului
 - ▶ la terminarea lui, se oprește procesul
- ▶ **SCM**
 - ▶ preia control asupra thread principal
 - ▶ creează un thread pentru apel metodă *ServiceMain* pe fiecare serviciu ce dorește să îl pornească
 - ▶ dă comenzi metodei Handler prin thread principal

11/20/2014 7



Servicii și thread-uri

Când Windows pornește procesul serviciu, creează un thread, thread-ul principal, să ruleze punctul de început al procesului (în .NET Framework acesta este metoda Main). Thread-ul principal determină durata procesului. Dacă thread-ul principal se termină atunci procesul se oprește.

În cazul serviciilor, codul punctului de început înregistrează serviciile care aparțin procesului folosind **SCM**. După înregistrare **SCM** preia controlul asupra thread-ului principal; deci are control asupra timpului de execuție al procesului serviciu. **SCM** creează apoi un thread pentru a apela metoda **ServiceMain** pentru fiecare serviciu pe care dorește să îl pornească, ceea ce înseamnă că orice cod din această metodă nu va rula pe același thread ca și punctul de început al procesului.

De asemenea serviciul creează de obicei un thread separat pentru mesajele IPC, astfel comenzile utilizator prin IPC sunt preluate de un alt thread.

În final, **SCM** folosește thread-ul principal pentru a da comenzi metodei Handler.



Clasa **ServiceBase**

- ▶ Metoda statică **Run** informează **SCM-ul** de rutinele metodei handler ale serviciului
- ▶ **SCM** apelează metodele handler pentru serviciu:
 - ▶ Când serviciul pornește, se oprește, este suspendat sau continuă
 - ▶ Când este ridicat un eveniment legat de sursa de energie, server terminal, sau închiderea sistemului
- ▶ **SCM** apelează serviciul pentru a trata comenzile personalizate

•OnStart	•OnContinue	•OnCustomCommand
•OnStop	•OnPause	•OnPowerEvent
•OnShutdown	•OnSessionChange	•Run



11/20/2014 8

.NET Framework furnizează clasa **ServiceBase** cu unitatea de asamblare **System.ServiceBase**, pe care ar trebui să o adăugați ca referință la proiectului de serviciu. Folosiți clasa **ServiceBase** ca o clasa de bază pentru serviciu.

Această clasa are o metodă statică numită **Run**, care conține tot codul necesar pentru a înregistra metoda **ServiceMain** și o metodă handler pentru comenzi furnizată de clasă. Metoda handler pentru comenzi apelează membri care pot fi suprascrise pentru anumite evenimente, astfel dacă doriți să tratați aceste evenimente puteți suprascrive metodele respective. Constructorul clasei **ServiceBase** este apelat la crearea obiectului și ar trebui folosit doar pentru inițializarea acestuia. Inițializarea serviciului ar trebui realizată în metoda **OnStart**.

Tratarea comenzilor

Clasa **ServiceBase** înregistrează o metodă privată ca metodă handler pentru comenzi . Această metodă apelează o metodă diferită (ce poate fi suprascrisă) pentru orice comandă specifică.



Clasa **ServiceBase**

- ▶ **Creare aplicație serviciu**
 - ▶ Creeare clasă derivată din clasa *ServiceBase*
 - ▶ Implementare metodă **OnStart**
 - ▶ Implementare metodă **OnCustomCommand**, pentru comenzi personalizate de la SCM
 - ▶ Suprascriere a celorlalte metode handler pentru comenzi, dacă este necesar.
- ▶ Proprietate *ServiceName* pentru setare nume
- ▶ Metoda statică *Run*

11/20/2014 9

Aceste metode pot fi suprascrise sau sunt virtuale, ceea ce înseamnă că puteți furniza cod pentru a le suprascrie. Sunt apelate asincron de metoda **Handler**, astfel dacă execuția codului durează o perioadă lungă de timp, serviciul poate trata alte comenzi și va folosi un fir de execuție thread pool.

Atenție! Nu există o definiție exactă a ceea ce înseamnă suspendarea unui serviciu. Dacă aplicațiile comunică cu serviciul folosind un canal **IPC**, cea mai simplă metodă de a suspenda serviciul este prin neacceptarea comenzilor **IPC**.

Dacă serviciul rulează sub un server de terminal și sesiunea este creată sau terminată, serviciul primește un obiect **SessionChangeDescription**, care are un ID de sesiune și o valoare din enumerația **SessionChangeReason** pentru a indica motivul conexiunii sau terminării.


Crearea unei aplicații serviciu

Creați o clasă derivată din clasa **ServiceBase**:

- ❖ **ServiceBase** face parte din namespace-ul **System.ServiceProcess** care va trebui adăugat ca referință în cadrul proiectului.
- ❖ Implementați metoda **OnStart**.
- ❖ Implementați metoda **OnCustomCommand**, pentru comenzi personalizate de la SCM.
- ❖ Suprascrieți celelalte metode handler pentru comenzi dacă este necesar.

Configurați proprietățile clasei. Folosiți proprietatea **ServiceName** pentru a da unui serviciu un nume unic.

În metoda Main apelați metoda statică **Run**. Dați acestei metode ca parametru un vector de obiecte **ServiceBase**, în care fiecare obiect reprezintă o instanță a fiecărui serviciu din proiect.




Clasa **ServiceBase**

Exemplu:

```
class SimpleService: ServiceBase
{
    public SimpleService()
    {
        ServiceName = "Simple Service";
    }
    protected override void OnStart(string[] args)
    {
        //Initializare IPC
    }
    protected override void OnStop()
    {
        //Opriti IPC
    }
}



class ServiceProgram
{
    // punct de început
    public static void Main()
    {
        ServiceBase.Run(new SimpleService())
    }
}
```



11/20/2014 10

Dacă doriți să generați un mesaj de notificare în event log, puteți folosi proprietatea **EventLog**. În plus, dacă setați proprietatea **AutoLog** la valoarea true, clasa va genera un mesaj de event log la primirea comenzilor de pornire, oprire, suspendare sau continuare.

Serviciul trebuie să fie instalat înainte de pornirea **SCM-ului**, astfel ar trebui să adăugați o clasă installer proiectului.



Accesarea serviciilor

- ▶ Clasa **ServiceController**
 - ▶ Oferă acces la servicii folosind **SCM-ul**
 - ▶ Folosește constructorul sau metoda **GetServices** pentru a accesa un serviciu după nume
 - ▶ Apelează metode pentru a porni, opri, suspenda sau continua un serviciu
 - ▶ Apelează comanda **ExecuteCommand**, pentru comenzi personalizate

•Start

•Stop

•Continue

•Pause

•ExecuteCommand

•GetServices

- ▶ Clasa **ServiceBase**
 - ▶ Metodele handler sunt apelate de **SCM**

•OnStart

•OnStop

•OnContinue

•OnPause

•OnCustomCommand

11/20/2014 11



Serviciile nu au o interfață cu utilizatorul. Există două modalități de a comunica cu un serviciu: **IPC** sau **SCM**. Folosiți clasa **ServiceController** pentru a porni un serviciu sau pentru a accesa un serviciu în execuție și pentru a informa **SCM-ul** să ofere comenzi serviciului.

Clasa **ServiceController**

Există două modalități de a inițializa o instanță a acestei clase:

- ❖ Dați numele serviciului constructorului clasei **ServiceController**.
- ❖ Apelați metoda **GetServices** pentru a obține un vector de obiecte **ServiceController**, unul pentru fiecare serviciu înregistrat pe computer.

Puteți folosi obiectul **ServiceController** pentru a oferi comenzi serviciului.



Accesarea serviciilor

Adăugarea unui serviciu din Service Explorer


- ▶ În **Designer view**, expandați nodul **Servers**, apoi expandați nodul pentru serverul pe care doriți să îl accesați.
- ▶ Deschideți nodul **Services**.
- ▶ Click dreapta pe serviciul pe care doriți să îl accesați, apoi pe meniul de context, faceți click pe **Add to designer**.

11/20/2014 12

Următorul cod explică modul de creare a unui obiect **ServiceController** și folosirea acestuia pentru a porni sau opri un serviciu.


```
public partial class MyForm: Form
{
    private Icontainer components;
    private ServiceController service;
    public MyForm()
    {
        InitializeComponent();
        //Creaza doua butoane si adauga metode handler
        components = new Container();
        service = new ServiceController("Simple Service");
        components.Add(service);
    }
    private void btnStart_Click(object sender, EventArgs e)
    {
        if(service.Status == ServiceControllerStatus.Stopped)
        {
            service.Start();
        }
    }
}
```

```
    }  
    private void btnStop_Click(object sender, EventArgs e)  
    {  
        if(service.CanStop)  
            service.Stop();  
    }  
}
```



Crearea **Service Application**

- ▶ Selectați **Windows Service** din *project types*
- ▶ Folosiți componenta **designer** pentru a adăuga elemente noi proiectului (ex.: conexiunile unei baze de date)
- ▶ Folosiți fereastra **Properties** pentru a configura serviciul




11/20/2014 13

Visual Studio furnizează un șablon de proiect pentru scrierea serviciilor Windows. Șablonul generează codul schelet pentru un obiect **ServiceBase**, precum și codul din metoda Main pentru a înregistra serviciul la **SCM**.

Crearea unui proiect serviciu

- ❖ În Visual Studio, click pe **File** → **New** → **Projects**
- ❖ În **Project Types**, expandați nodul **Visual C#**.
- ❖ **Project Types**, click pe Windows.
- ❖ În **Visual Studio Installed Templates**, click pe **Windows Service**.
- ❖ Furnizați numele și locația proiectului.


Visual Studio afișează componenta designer unde puteți pune obiecte folosite de serviciu, din **Toolbox** sau **Server Explorer**.



Crearea **Service Application**

Probleme ce apar la debugging într-un proiect de serviciu:

- ▶ Serviciile sunt pornite de **SCM**, astfel **Visual Studio** nu poate porni procesul din **debugger**
- ▶ Serviciile nu vor rula folosind contul de utilizator interactiv



11/20/2014 14

Debugging pentru un serviciu

Visual Studio permite atașarea debugger-ului unui proces, nu și pornirea procesului din debugger. Pentru atașarea unui serviciu, Visual Studio trebuie să fie pornit din contul de **Administrator**.



Pornirea Visual Studio din contul de Administrator

- ❖ Click pe **Start**, apoi click dreapta pe **Microsoft Visual Studio 2008/2010**.
- ❖ Click pe **Run As**.
- ❖ În căsuța de dialog **Run As**, faceți click pe **The following user**.
- ❖ Asigurați-vă că **User name** este **Administrator**, introduceți parola, apăsați pe **OK**.

Crearea **Service Application**

Atașarea debugger-ului unui proces

- ▶ Faceți click pe **Debug**, apoi pe **Attach to Process**
- ▶ În căsuța de dialog **Attach to Process**, selectați **Show processes from all users**. Lista **Available Processes** afișează toate procesele, inclusiv procesele de serviciu
- ▶ Faceți click pe procesul de servicii la care doriți să faceți debug, apoi click pe **Attach**
- ▶ Debugger-ul este atașat serviciului.



Instalarea serviciilor

- ▶ Metodele claselor Installer sunt apelate folosind utilitarul de instalare
 - ▶ pentru acțiuni suplimentare puteți suprascrie metode
- ▶ Installer-ul trebuie să deriveze clasa **Installer** pentru a putea accesa colecția **Installers**
 - ▶ **ServiceProcessInstaller**
 - ▶ **ServiceInstaller**
- ▶ Fiecare clasă Installer trebuie să aibă atributul **RunInstallerAttribute**

11/20/2014 16



Informațiile folosite de **SMC** pentru a porni un serviciu sunt păstrate în registrele sistemului. Când instalați un serviciu pe computer, ar trebui să adăugați aceste informații în registre.

Trebuie să creați o instanță a clasei **ServiceInstaller** pentru fiecare serviciu din proces și să furnizați informații de instalare. De asemenea trebuie să creați o instanță a clasei **ServiceProcessInstaller**. .NET Framework furnizează un utilitar numit `instalutil.exe`, pe care trebuie să îl folosiți pentru a instala serviciul. Acest utilitar rulează fiecare clasă installer pentru a instala sau dezinstala servicii.

Clase Installer

Clasa Installer se găsește în namespace-ul `System.Configuration.Install`, care va trebui adăugat ca referință în cadrul proiectului.

Procesul de instalare ar trebui să aibă o clasă de instalare cu atributul **RunInstallerAttribute**. Folosiți constructorul acestei clase pentru a adăuga în colecția **Installers** o instanță a clasei **ServiceInstaller** pentru fiecare serviciu din proces.



Instalarea serviciilor

Exemplu:

```
[RunInstaller(true)]
public class MyInstaller: Installer
{
    public void MyInstaller()
    {
        ServiceProcessInstaller procInst = new ServiceProcessInstaller();
        procInst.Account = ServiceAccount.LocalSystem;
        ServiceInstaller servInst = new ServiceInstaller();

        servInst.ServiceName = "Simple Service";
        Installers.Add(procInst);
        Installers.Add(servInst);
    }
}
```



11/20/2014 17

Înregistrarea serviciilor

SCM folosește informații furnizate de clasele installer și stocate în registre pentru pornirea unui serviciu.

Un proces poate conține unul sau mai multe servicii.

.NET Framework furnizează clasa **ServiceProcessInstaller** pe care o folosim pentru a configura setările globale pentru proces, care afectează toate serviciile conținute de acesta. În tabelul următor sunt descrise principalele proprietăți ale clasei **ServiceProcessInstaller**.



Instalarea serviciilor

Adăugarea unei clase Installer folosind Visual Studio

Visual Studio furnizează un wizard pentru a adăuga instalare proiectului. Wizard-ul adaugă trei clase:

- ▶ O clasă derivată din clasa **ServiceProcessInstaller**, care deține informații de securitate pentru proces.
- ▶ O clasă derivată din clasa **ServiceInstaller**, care deține informații despre serviciu.
- ▶ O clasă derivată din clasa **Installer**, care are atributul **RunInstallerAttribute**. Această clasă are câmpuri instanțe ale celorlalte clase installer, și acestea sunt adăugate colecției **Installers**.

11/20/2014 18

Atenție! Proprietățile **Password** și **Username** sunt stocate în format text în unitatea de asamblare, astfel aceste informații pot fi vizualizate. Totuși dacă setați proprietatea **Account** la valoarea **ServiceAccount.User** și nu completați proprietățile **Password** și **Username**, veți fi întrebați aceste informații la instalare. Acesta este cel mai sigur mod de a utiliza aceste proprietăți.

Adăugarea unei clase Installer folosind Visual Studio

În **Solution Explorer**, click-dreapta pe clasa serviciului, apoi click pe **View Designer** pentru a arăta serviciul în modul designer



- ❖ Click dreapta pe serviciul designer și apoi faceți click pe **Add Installer**
- ❖ Visual Studio crează un nou fișier numit **ProjectInstaller**, și apare un nou tab designer care prezintă două componente: un installer pentru proces și un installer pentru serviciu
- ❖ Folosiți fereastra **Properties** pentru a oferi valori fiecărui

installer

Clasa **Installer** este clasa de bază pentru clasa **ServiceInstaller** și **ServiceProcessInstaller**.

Instalarea Serviciilor

Doar administratorii pot instala un serviciu. Trebuie să porniți o fereastră **Command Prompt** folosind contul de **Administrator**. În această fereastră puteți rula utilitarul **instalutil** pentru a instala și dezinstala servicii și puteți folosi comanda **net** pentru a porni sau opri un serviciu.



Manipularea domeniilor de aplicație

- ▶ Introducere în domenii de aplicație
- ▶ Folosirea clasei **AppDomain**
- ▶ Configurarea domeniilor de aplicație
- ▶ Tipuri și unități de asamblare
- ▶ Crearea unui domeniu de aplicație
- ▶ Apelarea obiectelor dintr-un alt domeniu de aplicație



11/20/2014 19

Tot codul .NET Framework rulează în cadrul unui domeniu de aplicație. Un domeniu se comportă ca o unitate de izolare, iar codul dintr-un domeniu nu are acces direct la codul dintr-un alt domeniu. În acest capitol sunt descrise proprietățile domeniilor, modul de creare al acestora, precum și modul de executare al codului dintr-un domeniu.

Obiective

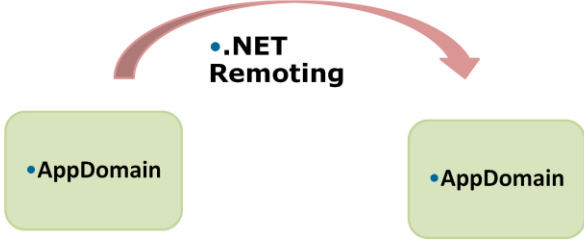
După completarea acestui capitol veți fi capabil să:

- ❖ Descrieți scopul și caracteristicile domeniilor de aplicații.
- ❖ Folosiți clasa **AppDomain** pentru a accesa și gestiona domeniile de aplicație.
- ❖ Configurați domenii de aplicație.
- ❖ Descrieți cum domeniile folosesc tipuri și unități de asamblare.
- ❖ Creați noi domenii și le configurați .
- ❖ Rulați cod într-un alt domeniu de aplicație.



Introducere în domenii de aplicație

- ▶ **AppDomain**
 - ▶ Fiecare proces are cel puțin un domeniu de aplicație
 - ▶ O unitate de asamblare este încărcată într-un anumit domeniu
- ▶ **.NET Remoting**
 - ▶ Codul dintr-un domeniu de aplicație poate accesa cod dintr-un alt domeniu de aplicație



```
graph LR; AD1[AppDomain] -- ".NET Remoting" --> AD2[AppDomain]
```



Un proces poate avea unul sau mai multe domenii de aplicație. Tot codul .NET Framework rulează în cadrul unui domeniu de aplicație și este restricționat la acel domeniu. Unitățile de asamblare sunt de asemenea restricționate la un domeniu; astfel, atunci când creați un obiect pornind de la o unitate de asamblare care nu a fost încă încărcată, la runtime unitatea de asamblare va fi încărcată în domeniul curent.

Un obiect .NET Framework nu poate fi mutat într-un alt domeniu de aplicație și nu are acces direct la obiecte dintr-un alt domeniu. Totuși .NET Framework furnizează **.NET Remoting** pentru activarea și apelarea obiectelor dintr-un alt domeniu.

Izolarea Domeniilor de aplicație

Un domeniu de aplicație este ca un proces de categorie ușoară rulând în cadrul unui alt proces. Precum procesele, domeniile de aplicație furnizează un anumit grad de securitate și izolare, și conțin resurse specifice acelui domeniu.

Un obiect este localizat în cadrul unui singur domeniu de aplicație, și poate face referință doar către obiecte din același domeniu de aplicație. Dacă un obiect aruncă o excepție care nu este tratată de codul obiectului, aceasta va fi tratată de domeniul aplicației. De asemenea .NET Framework vă permite să accesați obiecte din alte domenii folosind **.NET Remoting**.

Folosirea clasei **AppDomain**

- ▶ Încarcă și descarcă domenii
- ▶ Accesează componente și cod din alte domenii
- ▶ Oferă acces unităților de asamblare dintr-un domeniu
- ▶ Specifică politica unui domeniu
- ▶ Tratează excepțiile care nu au fost tratate

•CreateDomain	•Unload	•CurrentDomain
•CreateInstance	•DoCallback	•ExecuteAssembly
•Load	•GetAssemblies	•SetThreadPrincipal
•SetAppDomainPolicy	•SetPrincipalPolicy	•UnhandledException ⚡
•AssemblyResolve ⚡	•TypeResolve ⚡	

Academia Microsoft

Clasa **AppDomain** are metode și proprietăți pentru crearea și configurarea domeniilor de aplicație, precum și pentru apelarea obiectelor din alte domenii.

Domenii și unități de asamblare

Multe din proprietățile unui domeniu de aplicație sunt read-only; astfel după crearea unui domeniu putem citi aceste proprietăți, dar nu le putem modifica. Totuși putem specifica valorile acestor proprietăți la creare. Domeniul curent este accesibil prin intermediul proprietății **CurrentDomain** și metodei **Thread.GetDomain**.

Crearea unei metode se poate realiza folosind metoda **CreateDomain**; această metodă va fi explicată într-un slide ulterior. Dacă aveți o referință către un obiect **AppDomain**, îl puteți trimite metodei **Unload**, pentru a descărca domeniul. Această metodă se referă la întregul domeniu. Clasa **AppDomain** mai are o metodă numită **Load**, folosită pentru a specifica o anumită unitate de asamblare pentru a fi încărcată în domeniu.

Nu puteți descărca unități de asamblare individual, deoarece metoda **Unload**

descarcă toate unitățile de asamblare din domeniu. Puteți crea un domeniu dinamic și folosi clasele **ReflectionEmit** pentru a crea tipuri.

Metoda **GetAssemblies** returnează un vector care conține toate unitățile de asamblare încărcate în domeniu, iar metoda **ExecuteAssembly** încarcă și execută o unitate de asamblare prin apelarea metodei de început al acesteia.



Folosirea clasei **AppDomain**

► **Evenimente**

- 8 evenimente
- *UnhandledException*
 - unul din evenimente
 - este ridicat atunci când o excepție este ridicată pe un thread și aceasta nu este tratată
 - adăugare metodă handler pentru eveniment

Academia Microsoft

Rularea codului într-un domeniu

După ce o unitate de asamblare a fost încărcată într-un domeniu, puteți folosi diverse metode **CreateInstance** pentru a crea o instanță de tip în unitatea de asamblare. Pentru a executa codul unui obiect dintr-un alt domeniu, există un mecanism care implică trimiterea unui obiect delegat **CrossAppDomainDelegate** metodei **DoCallback**.

Domenii și Securitate

Când creați un domeniu, puteți furniza informații adiționale (pentru securitatea accesului la cod) folosite pentru a determina setul de permisiuni care va fi acordat unei unități de asamblare. Setul de permisiuni este ales din grupurile de cod disponibile, iar politica de securitate al accesului la cod determină grupurile de cod disponibile. La crearea unui domeniu acesta primește politică pentru enterprise, user și mașină; puteți apela metoda **SetPrincipalPolicy** pentru a specifica tipul de principiu legat de un thread la crearea acestuia.

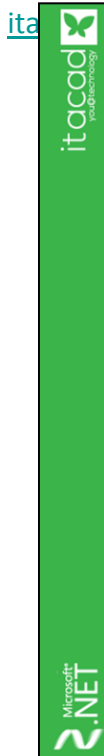
Evenimente AppDomain

Există opt evenimente, iar unul dintre acestea este **UnhandledException**. Aceasta este ridicat atunci când o excepție este ridicată pe un thread și aceasta nu este tratată.

Aceasta este ultima șansă de a trata excepția, deoarece după ce acest eveniment este ridicat la runtime de obicei se oprește domeniul de aplicație. Dacă dorim să oferim o ultimă șansă de a trata excepția (spre exemplu salvarea datelor), putem să adăugăm o metodă handler pentru acest eveniment.

```
static void Main()
{
    AppDomain.CurrentDomain.UnhandledException +=
        new UnhandledExceptionHandler(ExceptionHandler);
    ...
}

static void ExceptionHandler(object sender,
    UnhandledExceptionEventArgs e)
{
    ...
}
```



Configurarea domeniilor de aplicație

► AppDomainSetup

- Trimisă metodei **CreateDomain**, la crearea unui nou domeniu
- Este read-only după crearea domeniului
- Folosită pentru localizarea unităților de asamblare
- Folosită pentru a specifica fișierul de configurare și pentru a modifica informații de configurare

•ApplicationBase

•CachePath

•DynamicBase

•PrivateBinPath

•ShadowCopyDirectories

•ApplicationTrust

•ConfigurationFile

•SetConfigurationBytes

► Evidence

- Trimisă metodei **CreateDomain**, la crearea unui nou domeniu
- Folosită pentru determinarea permisiunilor

Academia Microsoft

❑ Majoritatea proprietăților de domeniu sunt read-only după crearea domeniului; astfel singura modalitate de a schimba modul în care un domeniu funcționează este prin specificarea acestui mod la crearea domeniului. Proprietățile domeniului de aplicație principal sunt accesibile prin intermediul proprietății **SetupInformation**. Această proprietate este un obiect **AppDomainSetup**. Pentru a seta aceste proprietăți trebuie să dați ca parametru un obiect inițializat **AppDomainSetup** metodei **CreateDomain**.

❑ Configurare și Domenii de aplicație

❑ Clasa **AppDomainSetup** este o clasă complexă cu multe proprietăți. Multe din proprietățile respective au legătură cu modul în care .NET Framework caută unitățile de asamblare cerute; spre exemplu, proprietatea **ApplicationBase** reprezintă numele directorului de bază al aplicației, iar proprietatea **PrivateBinPath** furnizează directorul în care sunt căutate unitățile de asamblare private. Există multe alte astfel de setări în fișierul de configurare. Implicit domeniul de aplicație folosește fișierul de configurare (pentru un proces, acesta are numele procesului cu extensia ".config", pentru o aplicație Web – web.config). Puteți schimba proprietatea **ConfigurationFile** pentru a indica faptul că un alt fișier este folosit.

❑ Informațiile de configurare sunt specifice fiecărui domeniu de aplicație în parte. Atunci când codul unui domeniu de aplicație citește informații de configurare, la runtime se încarcă secțiunea ca un obiect care va fi stocat în

memorie. Următoarea dată când codul domeniului cere o setare din aceeași secțiune, este folosit obiectul din memorie. Astfel, după ce un domeniu de aplicație a pornit, ar trebui să considerați setările de configurare ca și read-once: sunt citite o singură dată din fișierul de configurare, cu alte cuvinte nu sunt reflectate schimbări în cadrul fișierului.

❑ Totuși, dacă creați un alt domeniu de aplicație, care va folosi implicit fișierul de configurare, setările vor fi citite din acesta și vor evidenția toate schimbările realizate fișierului.

Tipuri și unități de asamblare

- ▶ Tipurile sunt definite în cadrul unei unități de asamblare
- ▶ Resursele sunt încărcate într-un anumit domeniu:
 - ▶ Unități de asamblare, module și alte resurse
- ▶ Dacă două domenii folosesc același tip, fiecare are o copie separată a unității de asamblare
 - ▶ Câmpurile statice sunt specifice domeniului

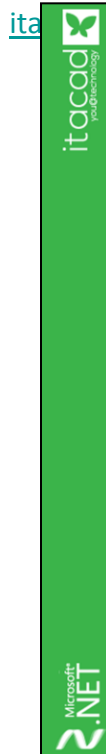
Academia Microsoft

❑ Obiectele sunt instanțe ale tipurilor de date, iar tipurile de date sunt definite în cadrul unităților de asamblare. La fel ca și obiectele, tipurile de date și unitățile de asamblare aparțin unui singur domeniu de aplicație. Acest lucru înseamnă că tipurile de date sunt specifice fiecărui domeniu de aplicație.

❑ Date statice și Domenii de aplicație

❑ Dacă aveți două domenii și fiecare domeniu încarcă câte un tip, unitatea de asamblare este încărcată în ambele domenii. Un tip poate avea instanță sau date statice.

❑ Fiecare obiect prezintă spațiu de stocare pentru datele instanței. Datele statice sunt partajate de către toate instanțele și aparțin tipului. Deoarece fiecare tip aparține unui anumit domeniu, dacă un tip este folosit în mai multe domenii atunci există o copie a datelor statice în fiecare domeniu ce folosește tipul respectiv.



Crearea unui domeniu de aplicație

- ▶ Dacă este necesar, creați un obiect **AppDomainSetup** folosind informațiile de setup
- ▶ Dacă este necesar, creați un obiect **Evidence** și un obiect **PermissionSet** pentru permisiunile care vor fi acordate tuturor unităților de asamblare din domeniu
- ▶ Apelați metoda **CreateDomain** pentru a crea un domeniu de aplicație

Academia Microsoft

Pentru crearea unui nou domeniu de aplicație trebuie să apelați metoda **CreateDomain**. Această metodă este supraîncărcată și opțional puteți specifica seturi de permisiuni și alte proprietăți. Runtime creează un nou domeniu în cadrul aceluiași proces și returnează un obiect **AppDomain**. Prin folosirea acestui obiect puteți încărca unități de asamblare crea tipuri.

Apelarea metodei CreateDomain

Următoarele exemple prezintă modul de creare a unui domeniu:

```
AppDomainSetup setup = new AppDomainSetup();
setup.ApplicationBase = @"E:\MyApp\DomainOne";
setup.ConfigurationFile = @"E:\MyApp\DomainOne\domainOne.config";
```

```
Evidence baseEvidence = AppDomain.CurrentDomain.Evidence;
Evidence evidence = new Evidence(baseEvidence);
```

```
// Restrictionarea permisiunilor - codul provine de la //www.example.com
evidence.AddHost(new Url("www.example.com"));
evidence.AddHost(new Zone(SecurityZone.Internet));
```



```
//Creare AppDomain
AppDomain newDomain = AppDomain.CreateDomain("domainOne", evidence, setup);
```

În exemplul de mai sus noul domeniu este creat folosind proprietatea **ApplicationBase** setată la folderul E:\MyApp\DomainOne. Când noul domeniu va crea tipuri folosind unități de asamblare

private, la runtime folderul este vizualizat.


Acest lucru nu afectează tipurile create folosind unități de asamblare partajate. Implicit, fișierul de configurare pentru o aplicație este încărcat din folderul de bază al aplicației, iar numele este numele prescurtat al aplicației cu extensia **".config"**. În exemplul de mai sus, domeniul are propriul fișier de configurare, care afectează tot codul ce folosește fișierul respectiv, inclusiv clasele .NET Framework.

De asemenea este specificată dovada și faptul că ar trebui codul să fie tratat la fel ca și codul descărcat de e Internet; în cazul de față dacă este un site numit www.example.com. Securitatea accesului la cod folosește această dovadă pentru a determina permisiunile unităților de asamblare ce rulează în cadrul domeniului.



Apelarea obiectelor dintr-un alt domeniu

- ▶ **AppDomain**
 - ▶ Apelați metoda **CreateInstance**
 - ▶ Apelați metoda **Unwrap** pentru a obține un proxy
 - ▶ Trimiteți tipul obiect și invocați
- ▶ **AppDomain**
 - ▶ Creați delegatul **CrossAppDomainDelegate**
 - ▶ Metoda **DoCallback**



Academia Microsoft

După ce ați creat un domeniu de aplicație, puteți rula cod în cadrul acesteia. Există două modalități de a realiza acest lucru: rularea unei metode folosind un delegat sau prin încărcarea și executarea de tipuri.

Accesarea tipurilor

Codul poate avea acces direct la tipuri doar în cadrul aceluiasi domeniu de aplicație. Dacă creați un alt domeniu și apoi la runtime se creează un proxy în domeniul inițial care folosește **.NET Remoting** pentru a comunica peste granița domeniului; această metodă de comunicare se numește **marshalling by reference**; deoarece o referință la un obiect dintr-un alt domeniu către domeniu de aplicație inițial. De asemenea puteți mobiliza prin valoare, dacă obiectul este serializat.

Datele serializate sunt trimise peste graniță, iar în domeniul destinație este creat un obiect, care este inițializat folosind datele serializate.

Ca apelant al codului nu puteți alege modul de mobilizare al obiectului; această decizie este luată de persoana care a realizat tipul. Pentru a mobiliza prin referință trebuie derivată clasa **MarshalByRefObject** la crearea tipului. Pentru a mobiliza prin valoare, proiectantul trebuie să adauge atributul

SerializableAttribute. Există multe modalități de a controla serializarea datelor, însă folosirea atributului **SerializableAttribute** este cea mai simplă modalitate.

Dacă o clasă nu are atributul **SerializableAttribute** și nu este derivată din **MarshalByRefObject**, nu poate fi accesată de peste granița unui domeniu.

Folosind DoCallback

Cea mai simplă modalitate de a rula cod dintr-un alt domeniu este folosind metoda **DoCallback**.

Atenție! Observați numele acesteia. **Back** este scris cu majusculă.

Metoda primește o instanță a unui delegat **CrossAppDomainDelegate**, care este un delegat fără parametri.

Metoda **CreateNewDomain** apelează metoda **DomainName** pentru a afișa numele friendly al domeniului. Apoi creează noul domeniu și apelează metoda **DoCallback**. La sfârșit descarcă domeniul și oferă următoarele rezultate:

TestApp.exe

domainOne

Primul domeniu de aplicație dintr-un proces este creat la runtime, și numele friendly al acestuia este numele procesului. Aceste rezultate arată faptul că același cod este apelat în ambele domenii de aplicație, prima oară în domeniul implicit, apoi în noul domeniu.

Crearea tipurilor

Clasa **AppDomain** vă permite să creați obiecte în cadrul unui domeniu. Tipul creat trebuie să fie mobilizat prin referință, dar metodele apelate din afara domeniului pot avea parametri mobilizați fie prin referință, fie prin valoare. Există patru metode pe care le puteți apela. Primele două sunt metodele **CreateInstance** și **CreateInstanceFrom**. Metoda **CreateInstance** identifică tipul folosind două string-uri: numele tipului și numele complet al unității de asamblare care conține tipul. Metoda **CreateInstanceFrom** identifică tipul folosind numele tipului și numele fișierului care conține unitatea de asamblare (care conține tipul).

Aceste două metode returnează un obiect **ObjectHandle** folosit de infrastructura .NET Remoting pentru a identifica obiectul. Nu puteți apela obiectul prin intermediul obiectului **ObjectHandle**; în schimb trebuie să despachetați acest obiect pentru a obține un proxy (metoda **Unwrap**). Clasa **AppDomain** are două metode **CreateInstanceAndUnwrap** și **CreateInstanceFromAndUnwrap**, care vor crea obiectul și vor apela automat **Unwrap**.

După despachetarea obiectului puteți să îl trimiteți tipului obiect și apoi să îi apelați metodele.

```
using System.Reflection;

class Test
{
    void CreateNewDomain()
    {
        Info info = new Info();
        info.DomainName();
        AppDomain domain = AppDomain.CreateDomain("domainOne");
        Info info2 = (Info)domain.CreateInstanceAndUnwrap(
            Assembly.GetExecutingAssembly().FullName, "Info");
        info2.DomainName();
        AppDomain.Unload(domain);
    }
}



class Info: MarshalByRefObject
{
    public void DomainName()
    {
        Console.WriteLine(AppDomain.CurrentDomain.FriendlyName);
    }
}
```

Codul important este apelarea metodei **CreateInstanceAndUnwrap**. Tipul este definit în unitatea de asamblare curentă, și reflecția este folosită pentru a returna numele complet al unității de asamblare curente în execuție. La runtime, numele este folosit pentru a încărca unitatea de asamblare în noul domeniu.

Al doilea parametru este numele tipului. În aceste exemple clasa **Info** este namespace-ul global și nu este o clasă internă. Dacă tipul este un namespace, trebuie să furnizați numele namespace-ului; dacă este o clasă internă ar trebui să furnizați numele clasei părinte.


```
namespace Utilities
{
    class Parent
    {
        class InnerClass: MarshalByRefObject
        {
        }
    }
}
```

Dacă doriți să creați o instanță a clasei **InnerClass** într-un alt domeniu de aplicație, numele folosit este **Utilities.Parent+InnerClass** (+ indică faptul că clasa este internă).



Sumar

- ▶ Crearea și instalarea serviciilor
- ▶ Domenii de aplicații



Academia Microsoft