netgenium

Konzolové aplikace

Framework NET Genium

Obsah

1 Základní informace

- Konzolové aplikace slouží k volání vlastního programového kódu v jazyce C# buď na straně serveru, nebo na straně klienta.
- Konzolové aplikace se používají v případech, kdy
  - je potřeba spouštět jednorázové servisní činnosti nad databází nebo
  - je potřeba tyto činnosti provádět v pravidelných intervalech.
- Pravidelné spouštění konzolových aplikací je možné nastavit v plánovači úloh systému Windows, nebo je možné konzolové aplikace přizpůsobit ke spouštění jako službu Windows.
- Konzolové aplikace se používají také pro přípravu a ladění externích funkcí. Detailní popis externích funkcí je uveden v samostatné příručce „Externí funkce“.
- Konzolové aplikace se připojují do databáze NET Genia
  - napřímo z aplikačního serveru, který je umístěný ve stejné síti jako databázový server (aplikační server a databázový server je často jeden a ten samý počítač) nebo
  - pomocí webových služeb, které jsou součástí každého NET Genia.
- Přímé připojení do databáze NET Genia je doporučovaný způsob, protože je řádově rychlejší než připojení přes webové služby, a podporuje databázové transakce, které v případě webových služeb není možné používat vůbec.
- Ladění nebo úpravy zdrojových kódů, které používají přímé připojení do databáze NET Genia, vyžadují databázi umístěnou ideálně na lokálním počítači, nebo ve stejné síti. To často není možné zajistit buď z bezpečnostních důvodů, nebo kvůli příliš velké databázi, kterou není reálné zazálohovat na databázovém serveru a přenést na lokální počítač. V takovém případě je doporučováno ladit zdrojové kódy pomocí API ve spojení s webovými službami.
- Konzolové aplikace využívají objekty a metody z knihovny „NETGeniumConnection.dll“, která je uložena v adresáři „NETGenium\bin“. „NETGeniumConnection.dll“ je knihovna se základními funkcemi pro práci s databází a se souborovými přílohami.
- Konzolové aplikace se vyvíjí prostřednictvím samostatného projektu v aplikaci „Visual Studio 2015“ a vyšší, resp. programováním zdrojových kódů tohoto projektu, a následnou kompilací projektu do souboru formátu „exe“.

2 Vytvoření konzolové aplikace

- V prvním kroku je nutné vytvořit nový projekt konzolové aplikace ve Visual Studiu pomocí následujících kroků, a přidat referenci na knihovnu „NETGeniumConnection.dll“:
  - Spustit Visual Studio
  - Z menu na hlavní liště zvolit „File / New / Project…“ (Ctrl+Shift+N)
    - Framework: .NET Framework 4.5.2
    - Project Type: Console Application
    - Name: ConsoleApplication1
    - Location: volitelné umístění projektu
    - Solution: Create new solution
    - Create directory for solution: Ne
    - Add to Source Control: Ne
  - Kliknout pravým tlačítkem na „References“, zvolit „Add Reference…“, a vybrat cestu k souboru „NETGenium\bin\NETGeniumConnection.dll“ na disku počítače
  - Zvolit režim kompilace „Debug“
    - Režim „Debug“ ve výchozím nastavení generuje soubory formátu „exe“ a „pdb“
    - Díky souboru formátu „pdb“ se snadno odhalují chyby a přerušení v konzolových aplikacích, protože součástí „Stack Trace“ každé chyby je i název souboru a číslo řádky, na které došlo k přerušení
    - Režim „Release“ se doporučuje až pro finální verzi vyladěných zdrojových kódů v konzolové aplikaci. Ve výchozím nastavení režim „Release“ generuje pouze soubor formátu „exe“.
  - Spustit projekt – z menu na hlavní liště zvolit „Debug“ / „Start Debugging“ (F5)

2.1 Přímé připojení do databáze NET Genia

- Přímé připojení do databáze používá třídu „DbConnection“ umístěnou v namespace „NETGenium“. Tato třída se chová analogicky jako obvyklá třída „DbConnection“ z namespace „System.Data“, nebo „SqlConnection“ z namespace „System.Data.SqlClient“.
- Třída „DbConnection“ je universální, a používá se jak pro databázový server Firebird, tak MSSQL.

```
using NETGenium;
using System;

namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      bool readline = true;

      using (DbConnection conn = new
DbConnection(@"driver=firebird;datasource=localhost;user=SYSDBA;password=masterkey;database=C:
\Firebird\netgenium.fdb;charset=WIN1250;collation=WIN_CZ"))
      // using (DbConnection conn = new
DbConnection("server=(local);Trusted_Connection=true;database=netgenium"))
      using (DbCommand cmd = new DbCommand(conn))
      {
        conn.Open();

        conn.RootPath = "C:\\inetpub\\wwwroot\\netgenium";
        DateTime now = DateTime.Now;

        Console.WriteLine("Connected to database");
        Console.WriteLine();
        Console.WriteLine(conn.User.FormatTimeSpan(DateTime.Now - now));
      }

      Console.WriteLine("OK");
      if (readline)
      {
        Console.ReadLine();
      }
    }
  }
}
```

2.2 Připojení do databáze NET Genia pomocí webových služeb

- Alternativou pro přímé připojení do databáze pomocí objektu „DbConnection“ je použití objektu „NETGeniumConnection“. Jde o třídu umístěnou v namespace „System“ knihovny „NETGeniumConnection.dll“, která umožňuje připojení do databáze NET Genia pomocí webových služeb.
- Webové služby stejně jako knihovna „NETGeniumConnection.dll“ tvoří jednotné API, a jsou součástí každého NET Genia.
- Většina metod používaných pro práci s daty v knihovně „NETGeniumConnection.dll“ má implementovanou jak variantu metody určenou pro objekt „DbConnection“, tak pro „NETGeniumConnection“.
- Po vytvoření instance objektu „NETGeniumConnection“ je nutné se přihlásit pomocí jména a hesla uživatele s administrátorským oprávněním. Na toto přihlášení se vztahují omezení na povolené IP adresy, definované v nastavení NET Genia. Každé takové přihlášení čerpá licence pro přihlášení do NET Genia, proto je důležité se na konci práce odhlásit.
- Třída „NETGeniumConnection“ je universální, a používá se jak pro databázový server Firebird, tak MSSQL.
- Komunikace se vzdáleným NET Geniem by měla být vždy zabezpečená SSL certifikátem, a proto je nutné nastavit správný a serverem podporovaný způsob komunikace pomocí „ServicePointManager.SecurityProtocol“.
    - Konzolové aplikace ve výchozím nastavení používají zastaralý HTTPS protokol „SecurityProtocolType.Tls“, který je na správně zabezpečených serverech zakázaný.
        - Povolování/zakazování HTTPS protokolů probíhá na serveru zápisem do registrů.
        - Každé NET Genium má v adresáři „Config/Tools“ uložený program
            - „SSL.reg“ pro nastavení doporučované bezpečnostní konfigurace HTTPS protokolů a
            - „SSL-ie6.reg“ pro nastavení konfigurace, která umožňuje přístup do NET Genia i zastaralým zařízením jako je Internet Explorer verze 6, staré tablety, mobilní telefony apod.
            - Detailní popis programů „SSL.reg“ a „SSL-ie6.reg“ je uveden v samostatné příručce „Utility“.
    - Současný doporučovaný způsob komunikace v roce 2020 je přes HTTPS protokol „SecurityProtocolType.Tls12“.
    - Jedinou výjimkou, kdy můžeme používat nezabezpečenou komunikaci, je připojení do lokálního NET Genia přes adresu „http://localhost/...“.

```
using NETGenium;
using System;

namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
      bool readline = true;

      NETGeniumConnection conn = new NETGeniumConnection("http://localhost/netgenium");
      conn.Login("NETGeniumConnection", "heslo");

      using (DbCommand cmd = new DbCommand(conn))
      {
        DateTime now = DateTime.Now;

        Console.WriteLine("Connected to database");
        Console.WriteLine();
        Console.WriteLine(conn.User.FormatTimeSpan(DateTime.Now - now));
      }

      conn.Logout();

      Console.WriteLine("OK");
      if (readline)
      {
        Console.ReadLine();
      }
    }
  }
}
```

3 Čtení dat z databáze

3.1 Načtení záznamů z SQL dotazu do objektu DataTable

```
// using NETGenium;
// using System;
// using System.Data;

DataTable data = Data.Get("SELECT * FROM sholiday WHERE date_ > " + conn.Format(new
DateTime(DateTime.Today.Year, 1, 1)), conn);
Console.WriteLine(conn.User.FormatDataTableText(data));

foreach (DataRow row in data.Rows)
{
  Console.WriteLine(row["id"]);
}
```

3.2 Uložení objektu DataTable na disk a jeho opětovné načtení z disku

```
// using NETGenium;
// using System;
// using System.Data;

DataTable data = Data.Get("SELECT * FROM sholiday WHERE date_ > " + conn.Format(new
DateTime(DateTime.Today.Year, 1, 1)), conn);

// HTML
Files.Write(Config.RootPath + "sholiday.html", conn.User.FormatDataTable(data));

// TXT
Files.Write(Config.RootPath + "sholiday.txt", conn.User.FormatDataTableText(data));

// XML
DataSet ds = new DataSet();
ds.Tables.Add(data);
ds.WriteXml(Config.RootPath + "sholiday.xml", XmlWriteMode.WriteSchema);

ds = new DataSet();
ds.ReadXml(Config.RootPath + "sholiday.xml", XmlReadMode.ReadSchema);
data = ds.Tables[0];

// JSON
Files.Write(Config.RootPath + "sholiday.json", ParserJSon.ToString(data));
data = ParserJSon.ToDataTable(Files.Read(Config.RootPath + "sholiday.json"));
```

3.3 Načtení záznamů z SQL dotazu do objektu DbRow

```
// using NETGenium;
// using System;

DbRow row = new DbRow("SELECT * FROM sholiday WHERE date_ > " + conn.Format(new
DateTime(DateTime.Today.Year, 1, 1)), conn);
if (row.Read())
{
  Console.WriteLine(row["id"]);
  Console.WriteLine(row.Report());
}
```

3.4 Parsování hodnot

```
// using NETGenium;
// using System;

DbRow row = new DbRow("SELECT * FROM sholiday WHERE date_ > " + conn.Format(new
DateTime(DateTime.Today.Year, 1, 1)), conn);
if (row.Read())
{
  int id = (int)row["id"];
  Console.WriteLine("id: " + id);

  int pid = Parser.ToInt32(row["pid"]);
  Console.WriteLine("pid: " + pid);

  double _pid = Parser.ToDouble(row["pid"]);
  Console.WriteLine("pid: " + _pid);

  string name = row["name"].ToString();
  Console.WriteLine("name: " + name);

  DateTime date = Parser.ToDateTime(row["date_"]);
  Console.WriteLine("date: " + conn.User.FormatDateTime(date));
}
```

3.5 Indexace načtených záznamů podle primárního klíče

```
// using NETGenium;
// using System;
// using System.Collections.Generic;
// using System.Data;

DataTable data = Data.Get("SELECT * FROM sholiday", conn);
Dictionary<int, DataRow> dictionary = Data.DictionaryInt32(data);

int id = 1;
if (dictionary.ContainsKey(id))
{
  DataRow row = dictionary[id];
  Console.WriteLine(row["id"]);
}
```

3.6 Indexace načtených vnořených záznamů podle cizího klíče „pid“

```
// using NETGenium;
// using System;
// using System.Collections.Generic;
// using System.Data;

DataTable data = Data.Get("SELECT * FROM sholiday", conn);
Dictionary<int, List<DataRow>> dictionary = Data.DictionaryInt32(data, "pid", true);

int pid = 0;
if (dictionary.ContainsKey(pid))
{
  List<DataRow> rows = dictionary[pid];
  Console.WriteLine(rows.Count + " rows");
}
```

4 Zápis dat do databáze

4.1 INSERT INTO – vytvoření nového záznamu v databázi

4.1.1 DataSaver – vytvoření jednoho záznamu

```
// using NETGenium;
// using System;

DataSaver ds = new DataSaver("sholiday", 0, cmd);
ds.Add("name", "Nový rok");
ds.Add("date_", new DateTime(DateTime.Today.Year, 1, 1));
ds.Execute();

Console.WriteLine(ds.Report());
```

4.1.2 DataSaver – vytvoření dvou záznamů v samostatné transakci

```
// using NETGenium;
// using System;

cmd.Transaction = conn.BeginTransaction();

try
{
  DataSaver ds = new DataSaver("sholiday", 0, cmd);
  ds.Add("name", "Nový rok");
  ds.Add("date_", new DateTime(DateTime.Today.Year, 1, 1));
  ds.Execute();

  Console.WriteLine(ds.Report());

  ds = new DataSaver("sholiday", 0, cmd);
  ds.Add("name", "Nový rok");
  ds.Add("date_", new DateTime(DateTime.Today.Year + 1, 1, 1));
  ds.Execute();

  Console.WriteLine(ds.Report());

  cmd.Transaction.Commit();
}
catch (Exception ex)
{
  cmd.Transaction.Rollback();
  throw ex;
}
```

4.1.3 DataSaver – vytvoření jednoho záznamu zkopírováním jiného záznamu

```
// using NETGenium;
// using System;

int id = 1;

DbRow row = new DbRow("SELECT * FROM sholiday WHERE id = " + id, conn);
if (row.Read())
{
  // row["ng_zadanokym"] = conn.User.LoginName;
  // row["ng_zadanokdy"] = DateTime.Now;
  // row["ng_zmenenokym"] = DBNull.Value;
  // row["ng_zmenenokdy"] = DBNull.Value;

  DataSaver ds = new DataSaver("sholiday", 0, cmd);
  ds.Add(row);
  ds.Execute();

  Console.WriteLine(ds.Report());
}
```

4.1.4 DataSaverSynchro – vytvoření jednoho záznamu včetně vytvoření záznamu historie a zajištění synchronizace záznamu

```
// using NETGenium;
// using System;

DataSaverSynchro ds = new DataSaverSynchro("sholiday", 0, conn);
ds.Add("name", "Nový rok");
ds.Add("date_", new DateTime(DateTime.Today.Year, 1, 1));
ds.Save(cmd);

Console.WriteLine(ds.Report());
```

4.2 UPDATE – editace existujícího záznamu v databázi

4.2.1 DataSaver – editace záznamu, jehož ID je načteno z databáze

```
// using NETGenium;
// using System;

int id = Data.ExecuteScalar2("SELECT id FROM sholiday WHERE name = " + conn.Format("Nový rok")
+ " AND date_ = " + conn.Format(new DateTime(DateTime.Today.Year, 1, 1)), conn);
if (id != 0)
{
  DataSaver ds = new DataSaver("sholiday", id, cmd);
  ds.Add("name", "Nový rok - TEST");
  ds.Execute();

  Console.WriteLine(ds.Report());
}
```

4.2.2 DataSaver – editace záznamu, jehož ID je uloženo v proměnné; pokud záznam neexistuje, bude vytvořen

```
// using NETGenium;
// using System;

int id = 1;

DataSaver ds = new DataSaver("sholiday", id, true, cmd);
ds.Add("name", "Nový rok - TEST");
ds.Execute();

Console.WriteLine(ds.Report());
```

4.2.3 DataSaverSynchro – editace záznamu, jehož ID je načteno z databáze, včetně vytvoření záznamu historie a zajištění synchronizace záznamu

```
// using NETGenium;
// using System;

int id = Data.ExecuteScalar2("SELECT id FROM sholiday WHERE name = " + conn.Format("Nový rok")
+ " AND date_ = " + conn.Format(new DateTime(DateTime.Today.Year, 1, 1)), conn);
if (id != 0)
{
  DataSaverSynchro ds = new DataSaverSynchro("sholiday", id, conn);
  ds.Add("name", "Nový rok - TEST");
  ds.Save(cmd);

  Console.WriteLine(ds.Report());
}
```

4.2.4 DataSaverSynchro – editace záznamu, jehož ID je uloženo v proměnné, včetně vytvoření záznamu historie a zajištění synchronizace záznamu; pokud záznam neexistuje, bude vytvořen

```
// using NETGenium;
// using System;

int id = 1;

DataSaverSynchro ds = new DataSaverSynchro("sholiday", id, true, conn);
ds.Add("name", "Nový rok - TEST");
ds.Save(cmd);

Console.WriteLine(ds.Report());
```

4.3 Synchronizace dat dvou databázových tabulek podle jednotného klíče

```
// using NETGenium;
// using System;
// using System.Collections.Generic;
// using System.Data;

string key = "ng_osobnicislo";

DataTable data1 = Data.Get("SELECT * FROM ng_data1", conn), data2 = Data.Get("SELECT * FROM
ng_data2", conn);
Dictionary<int, DataRow> dictionary = Data.DictionaryInt32(data1);

foreach (DataRow row2 in data2.Rows)
{
  DataRow row1 = Data.DataRow(dictionary, Parser.ToInt32(row2[key]));
  if (row1 == null)
  {
    DataSaver ds = new DataSaver("ng_data1", 0, cmd);

    for (int i = 1; i < data2.Columns.Count; i++)
    {
      ds.Add(data2.Columns[i].ColumnName, row2[data2.Columns[i].ColumnName]);
    }

    ds.Execute();
  }
  else
  {
    DataSaver ds = new DataSaver("ng_data1", (int)row1["id"], cmd);
    for (int i = 1; i < data2.Columns.Count; i++)
      if (row1[data2.Columns[i].ColumnName].ToString() !=
row2[data2.Columns[i].ColumnName].ToString())
      {
        ds.Add(data2.Columns[i].ColumnName, row2[data2.Columns[i].ColumnName]);
      }

    if (!ds.Empty)
    {
      ds.Execute();
    }
  }
}
```

5 Mazání dat z databáze

5.1 DELETE FROM – smazání záznamu z databáze

```
// using NETGenium;
// using System;

int id = Data.ExecuteScalar2("SELECT id FROM sholiday WHERE name = " + conn.Format("Nový rok -
TEST") + " AND date_ = " + conn.Format(new DateTime(DateTime.Today.Year, 1, 1)), conn);
if (id != 0)
{
  cmd.CommandText = "DELETE FROM sholiday WHERE id = " + id;
  cmd.ExecuteNonQuery();
}
```

5.2 DataSaverSynchro – smazání záznamu z databáze včetně vytvoření záznamu historie a zajištění synchronizace záznamu

```
// using NETGenium;
// using System;

int id = Data.ExecuteScalar2("SELECT id FROM sholiday WHERE name = " + conn.Format("Nový rok -
TEST") + " AND date_ = " + conn.Format(new DateTime(DateTime.Today.Year, 1, 1)), conn);
if (id != 0)
{
  DataSaverSynchro ds = new DataSaverSynchro("sholiday", id, conn);
  ds.Delete(cmd);
}
```

6 Souborové přílohy

6.1 Vytvoření souborové přílohy

```
// using NETGenium;
// using System;
// using System.IO;

string temp = Path.GetTempFileName();
Files.Write(temp, "abc");

int file = Attachment.Add("test.txt", temp, conn);
File.Delete(temp);
Console.WriteLine(file);
```

6.2 Načtení obsahu souborové přílohy pomocí objektu DbConnection

```
// using NETGenium;
// using System;
// using System.IO;

string temp = Path.GetTempFileName();
Files.Write(temp, "abc");

int file = Attachment.Add("test.txt", temp, conn);
File.Delete(temp);
Console.WriteLine(file);

string path = Attachment.FilePath(file, conn);
if (File.Exists(path))
{
  string s = Files.Read(path);
  Console.WriteLine(s);
}
```

6.3 Načtení obsahu souborové přílohy pomocí objektu NETGeniumConnection

```
// using NETGenium;
// using System;
// using System.IO;

string temp = Path.GetTempFileName();
Files.Write(temp, "abc");

int file = Attachment.Add("test.txt", temp, conn);
File.Delete(temp);
Console.WriteLine(file);

// Attachment
Attachment = conn.GetAttachment(id);
if (attachment != null)
{
  Console.WriteLine(attachment.Name + ": " + attachment.ContentBytes.Length + " bytes");
}

// byte[]
byte[] buffer = conn.GetAttachmentAsBytes(id);
if (buffer != null)
{
  Console.WriteLine(buffer.Length + " bytes");
}

// string
string s = conn.GetAttachmentAsString(id, Encoding.UTF8);
if (s != null)
{
  Console.WriteLine(s);
}
```

7 E-maily

7.1 Odeslání e-mailové zprávy

```
// using NETGenium;
// using System;
// using System.Net.Mail;

string html = NETGenium.Email.Message.Container("<b>Hello</b>");

MailMessage message = new MailMessage();
message.From = new MailAddress("@");
message.To.Add(new MailAddress("@"));
message.Subject = "";
message.AlternateViews.Add(Config.CreateTextAlternateView(Html.ToText(html)));
message.AlternateViews.Add(Config.CreateHtmlAlternateView(html, conn));

Config.SendMessage(message, conn);
```

8 Logování

8.1 Logování na disk do adresáře „Logs“

```
// using NETGenium;
// using System;
// using System.IO;

try
{
  throw new NullReferenceException("args");
}
catch (Exception ex)
{
  L.E("MyFirstFunction", ex);
  // L.LogError("MyFirstFunction", ex);
  // L.Error("MyFirstFunction", ex);

  L.W("MyFirstFunction", ex);
  // L.LogWarning("MyFirstFunction", ex);
  // L.Warning("MyFirstFunction", ex);

  L.N("MyFirstFunction", ex);
  // L.LogNotice("MyFirstFunction", ex);
  // L.Notice("MyFirstFunction", ex);
}
```

8.2 Obecné logování na disk

```
// using NETGenium;
// using System;

Files.Write(Config.RootPath + "test.log", "TEST");

try
{
  Files.WriteLine(Config.RootPath + "test.log", DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss") +
Convert.ToChar(160) + "TEST");
}
catch { }

Files.WriteLineTryCatch(Config.RootPath + "test.log", DateTime.Now.ToString("yyyy-MM-dd
HH:mm:ss") + Convert.ToChar(160) + "TEST");
```

9 Služby

9.1 Základní informace

- Konzolové aplikace je možné nainstalovat jako službu systému Windows, takže následně běží na pozadí operačního systému, a spouští se sami v pravidelných intervalech.
- Konzolové aplikace v režimu služby (dále jen „Služby“) musí mít definovaný název služby a jedinečný GUID projektu konzolové aplikace.
    - Název služby se používá při instalaci služby, a pod tímto názvem je možné službu dohledat (zastavit, spustit, restartovat) v seznamu služeb operačního systému Windows nebo ve správci úloh (Task Manager).
    - GUID projektu se používá pro bezpečné spuštění vždy pouze jedné instance služby tak, aby nedošlo k paralelnímu spuštění více instancí služby (při zkopírování existujícího projektu konzolové aplikace je nutné nastavit nový GUID projektu, aby tyto dvě služby mezi sebou nekolidovaly právě díky společnému GUID projektu).
- Služby používají konfigurační soubory, ze kterých si načítají nastavení připojení do databáze, adresu poštovního serveru odchozí pošty, počáteční a koncový čas technologické pauzy, během které služba nepracuje, atd.
- Služby se nepřipojují sami do databáze, a používají jednotný systém pro volání hlavního zdrojového kódu, který se vyskytuje v běžné konzolové aplikaci uvnitř metody „static void Main(string[] args)“.
- Služby nemají přístup k připojeným diskovým jednotkám, pouze k pevným nebo síťovým jednotkám. K diskům připojeným pomocí příkazu „mount“ musí být přistupováno pomocí síťové cesty, například „\\192.168.0.1\D\NETGenium“.
- Služby vyžadují přidělení administrátorského oprávnění pomocí manifestu, jinak budou selhávat základní operace typu zálohování, čtení nebo zápis na disk apod.

9.2 Vytvoření obálky služby a úprava souboru „Program.cs“

- V prvním kroku je nutné přesunout logiku konzolové aplikace umístěnou v metodě „static void Main(string[] args)“ do nového souboru „Service.cs“.
- Z tohoto zdrojového kódu je nutné odstranit vytvoření instance objektu „DbConnection“, „DbCommand“, a nastavení adresáře NET Genia.
- Minimální zdrojový kód služby zobrazuje následující příklad, namespace i název služby je pro názornost nastaven na „CRMService“:

```
using NETGenium;
using System;

namespace CRMService
{
  public class Service : ServiceTemplate, IServiceTemplate
  {
    public override string ServiceName { get { return "CRMService"; } }

    public Service()
    {
    }

    protected override void HandleProcess(string[] args, Config config, DbCommand cmd)
    {
      DbConnection conn = cmd.Connection;
      DateTime now = DateTime.Now;
    }
  }
}
```

- Ve druhém kroku je nutné změnit obsah souboru „Program.cs“ a nastavit interval spouštění služby.
- Konzolovou aplikaci od této chvíle nebude možné spustit běžným způsobem pomocí příkazu „Debug“ / „Start Debugging“ (F5), proto je důležité tento krok naplánovat až na chvíli, kdy bude konzolová aplikace vyladěná a stabilní.

```
using NETGenium;

namespace CRMService
{
    class Program
    {
        static void Main(string[] args)
        {
            if (false)
            {
                new Service().Run(args);
                return;
            }

            new Service().Process(args,
            new ServiceTemplateInstaller.InstallOptions()
            {
                Interval = 5
            });
        }
    }
}
```

- Kdykoliv v budoucnu je možné konzolovou aplikaci upravit zpětně tak, aby se dala spustit běžným způsobem pomocí příkazu „Debug“ / „Start Debugging“ (F5), a bylo možné další ladění nebo zkoušení konzolové aplikace.

```
using NETGenium;

namespace CRMService
{
    class Program
    {
        static void Main(string[] args)
        {
            if (true)
            {
                new Service().Run(args);
                return;
            }

            new Service().Process(args,
            new ServiceTemplateInstaller.InstallOptions()
            {
                Interval = 5
            });
        }
    }
}
```

9.3 Manifest

- Ve třetím kroku je nutné přidělit konzolové aplikaci administrátorské oprávnění pomocí souboru manifestu:
  - Spustit Visual Studio a otevřít projekt konzolové aplikace
  - Kliknout pravým tlačítkem na název projektu v Solution Exploreru, zvolit „Add / New Item…“, do pole pro vyhledávání napsat „manifest“, najít položku „Application Manifest File“, a zvolit „Add“
  - V obsahu souboru manifestu najít „<requestedExecutionLevel level="asInvoker" uiAccess="false" />“ a změnit na „<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />“

9.4 Příklad služby CRMService

- Následující příklad demonstruje jednoduchou službu „CRMService“, která stahuje obsah technologické stránky, příchozí e-maily zálohuje do adresáře „POP3Downloads“, a do okna konzolové aplikace a na disk do adresáře „Logs“ vypisuje detaily e-mailu.
- Obsah souboru „Service.cs“:

```
using NETGenium;
using System;
using System.Collections.Generic;
using System.Data;
using System.IO;

namespace CRMService
{
  public class Service : ServiceTemplate, IServiceTemplate
  {
    public override string ServiceName { get { return "CRMService"; } }

    private string dir, backupdir;

    public Service()
    {
      AddRequiredConfigKey("pop3Email");
      AddRequiredConfigKey("pop3Url");
      AddRequiredConfigKey("pop3Name");
      AddRequiredConfigKey("pop3Password");

      dir = Config.RootPath + "Temp\\";
      if (!Directory.Exists(dir))
      {
          Directory.CreateDirectory(dir);
      }

      backupdir = Config.RootPath + "POP3Downloads\\";
      if (!Directory.Exists(backupdir))
      {
        Directory.CreateDirectory(backupdir);
      }
    }

    protected override void HandleProcess(string[] args, Config, DbCommand cmd)
    {
      Files.DeleteDirectoriesYYYYMMDD(backupdir, DateTime.Today.AddMonths(-6));
      Files.DeleteLogs(Config.RootPath, DateTime.Today.AddYears(-1));

      DbConnection conn = cmd.Connection;
      DateTime now = DateTime.Now;
```

```
      // Open pop3
      NETGenium.Email.Pop3 pop3 = new NETGenium.Email.Pop3(0, config["pop3Url"],
config["pop3Name"], config["pop3Password"]);
      try
      {
        pop3.Open();

        for (int index = 0; index < pop3.Count; index++)
        {
          foreach (string file in Directory.GetFiles(dir))
          {
            File.Delete(file);
          }

          string backupdir2 = backupdir + now.ToString("yyyy-MM-dd") + "\\";
          if (!Directory.Exists(backupdir2))
          {
            Directory.CreateDirectory(backupdir2);
          }

          string eml = backupdir2 + Guid.NewGuid().ToString().ToUpper() + ".eml";

          NETGenium.Email.Message message = pop3.ReadMessage_SaveEmlAndAttachments(index, eml,
dir);
          Email.Eval(config, message, eml, true, "ID " + index, dir, ServiceName, now, cmd,
conn);

          // pop3.DeleteMessage(index); // Mazání zpráv z POP3 serveru až po odladění služby
          if (index == 99) break;
        }

        // Close pop3
        pop3.Close();
      }
      catch (Exception ex)
      {
        try { pop3.Close(); }
        catch { }

        if (ServiceLogger.ImportantError(ex))
        {
          L.E("Download e-mails from POP3", ex);
        }
        else
        {
          L.W("Download e-mails from POP3", ex);
        }

        return;
      }
    }
  }
}
```

- Logika zpracování příchozí e-mailové zprávy je řešena v samostatném souboru „Email.cs“.
- Obsah souboru „Email.cs“:

```
using NETGenium;
using NETGenium.Email;
using System;

namespace CRMService
{
  internal class Email
  {
    public static bool Eval(Config config, Message message, string eml, bool pop3message,
string name, string dir, string serviceName, DateTime now, DbCommand cmd, DbConnection conn)
    {
      L.N(name);

      foreach (string file in Directory.GetFiles(dir))
      {
        L.N(Path.GetFileName(file));
      }

      L.N(message.DateOriginal.ToString());
      L.N(message.From.EmailAddress);
      L.N(message.To);
      L.N(message.Subject);
      L.N(message.Text);

      return true;
    }
  }
}
```

9.5 Konfigurační soubory služby

- Služby mohou obsahovat dva druhy konfiguračních souborů:
  - Hlavní konfigurační soubor (například „CRMService.exe.config“), jehož název musí vycházet z názvu aplikace (v tomto případě „CRMService.exe“).
  - Konfigurační soubory XML, které se používají v případech, kdy služba obsluhuje více databází najednou (například „netgenium1.xml“, „netgenium2.xml“, apod.).
- Zdrojový kód služby se spouští pro každý konfigurační soubor zvlášť. Mezi přípustné kombinace patří:
  - CRMService.exe.config
  - netgenium1.xml, netgenium2.xml, …
  - CRMService.exe.config, netgenium1.xml, netgenium2.xml, …
- Konfigurační soubory musí být umístěné ve stejném adresáři jako „CRMService.exe“. Pokud jsme ve fázi ladění služby ve Visual Studiu, není nutné vytvářet konfigurační soubor ručně, protože každý projekt konzolové aplikace automaticky obsahuje výchozí konfigurační soubor „App.config“. Tento soubor se při každé kompilaci projektu sám překopíruje do adresáře „Debug“ nebo „Release“ (podle zvoleného režimu kompilace) do souboru s názvem „abc.exe.config“. V našem případě se tedy obsah konfiguračního souboru „App.config“ při kompilaci projektu sám zkopíruje do souboru „CRMService.exe.config“.
- Příklad konfiguračního souboru:

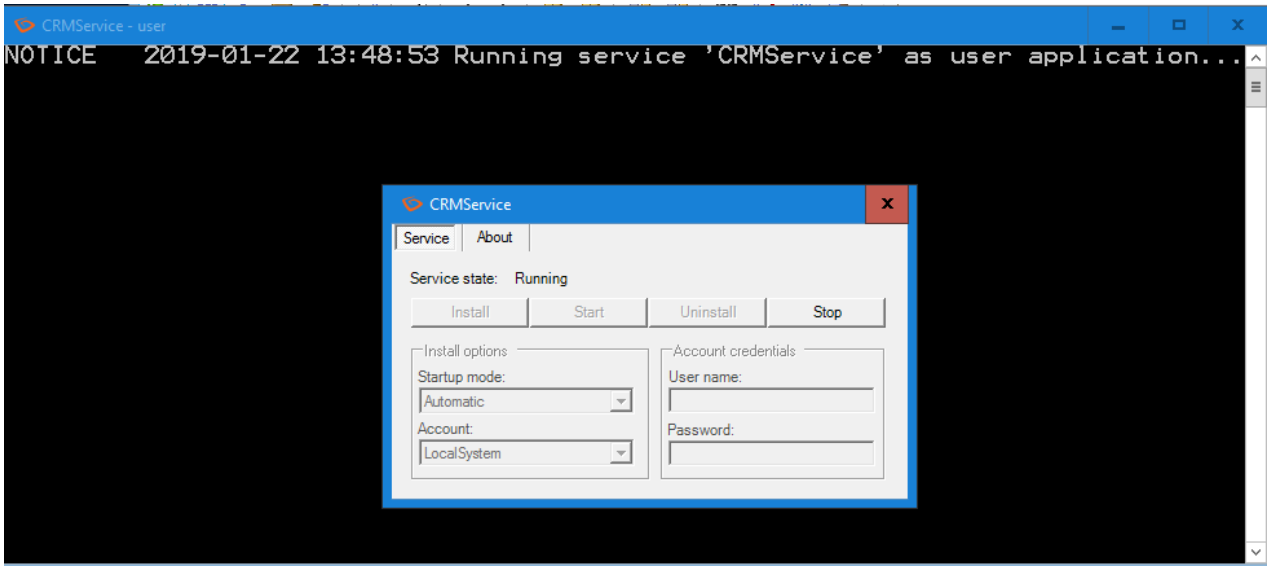
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>

    <add key="rootPath" value="C:\inetpub\wwwroot\netgenium" />

    <add key="stopFrom" value="00:00" />
    <add key="stopTo" value="06:00" />

    <add key="smtpServer" value="localhost" />

    <add key="errorFrom" value="crmservice@firma.cz" />
    <add key="errorTo" value="support@firma.cz" />

    <add key="pop3Url" value="pop3.firma.cz" />
    <add key="pop3Email" value="crm@firma.cz" />
    <add key="pop3Name" value="crm@firma.cz" />
    <add key="pop3Password" value="pop3heslo" />

  </appSettings>
</configuration>
```

- **rootPath** – absolutní cesta na disku k adresáři NET Genia
- **stopFrom** – počáteční čas technologické pauzy ve formátu „HH:mm“, během které služba nepracuje
- **stopTo** – koncový čas technologické pauzy ve formátu „HH:mm“, během které služba nepracuje
- **smtpServer** – adresa poštovního serveru odchozí pošty

- **errorFrom** – e-mailová adresa, ze které jsou zasílány notifikace o případných chybách ve službě
- **errorTo** – e-mailová adresa, na kterou jsou zasílány notifikace o případných chybách ve službě
- **pop3Url** – adresa poštovního serveru příchozí pošty (pro SSL zabezpečení je možné použít syntaxi „název serveru:číslo portu“)
- **pop3Email** – e-mailová adresa technologické schránky
- **pop3Name** – přihlašovací jméno k technologické schránce
- **pop3Password** – heslo k technologické schránce

9.6 Instalace služby

- K instalaci služby se používá soubor „nazev-sluzby.exe - user.lnk“, který vznikne automaticky spuštěním konzolové aplikace „nazev-sluzby.exe“.
- Po spuštění souboru „nazev-sluzby.exe - user.lnk“ je důležité zkontrolovat nastavení účtu, pod kterým se má služba spouštět. Výchozí nastavení je „LocalSystem“, což zajišťuje bezpečné spouštění služby bez budoucích možných komplikací s oprávněním služby.
- Instalace služby se provádí kliknutím na tlačítko „Install“.



- Odinstalace služby se provádí kliknutím na tlačítko „Uninstall“.

9.7 Výměna programových souborů služby

- Jakmile je služba jednou nainstalována, není možné měnit programové soubory služby, protože jsou neustále chráněny operačním systémem.
- Před výměnou programových souborů je nutné službu zastavit pomocí tlačítka „Stop“, a dialog nastavení služby zavřít pomocí ikony křížku.
- Následně je možné programové soubory služby přepsat, a službu znovu spustit pomocí tlačítka „Start“.