



FISCHER & CONSULTANTS

welcomes you to the
DevFest 2007 session

Configuring .NET Applications

Speaker
Michael Fischer

Copyright © 2007

Fischer & Consultants GmbH (F&C)
Martinstrasse 1
44137 Dortmund
www.appfact.com

Contact: Michael Fischer
Email mifi@appfact.de
Phone +49 (172) 6801489
Fax +49 (941) 599212011

Note

At the time of this session being published, there was no design time support for application settings in the beta of Vulcan.NET. This feature - a part of the Visual Studio Integration - was unofficially announced for the release product. To demonstrate the concepts of System.Configuration at this early point in time, the samples in this presentation use C# instead of Vulcan.NET.

Contents

- | | |
|--|--|
| 1. History | 6. More from
System.Configuration |
| 2. Basics | |
| 3. SimpleCSharpSample:
Configuration without
Code | 7. Questions and
Discussion |
| 4. Do it Yourself I:
ConfigurationManager | |
| 5. Do it Yourself II:
Your own Settings
Classes
(MSDN Sample) | |

Slide 2 of 21

Abstract:

How do you like it?

Configuring Vulcan.NET Applications

Configuring applications is as old as software development itself and storing configuration data had numerous fashions. The methods range from hard-coded settings, the legendary INI files and the (rightfully) hated registry over entries in SQL tables up to the „dernier cri“: XML files. The .NET Framework 2.0 offers System.Configuration to Vulcans, a very rich set of classes to store and retrieve application settings.

In this session, Michael Fischer examines the configuration of Vulcan.NET applications in detail. You will see easy ways to define and use settings for the whole application or a particular user, learn about roaming settings, make friends with dynamic properties in the Visual Studio form editor and receive a ready to use generic dialog for user settings. And - just in case you haven't done much work with Vulcan in Visual Studio yet - you will get a first impression of the new development environment.

1. History

My personal history of configuration storage

- n **DATA lines in dedicated configuration modules**
 - à GW-Basic: hard-coded settings
- n **INI files**
 - à Clipper: self-written access class based on FOpen
 - à VO: self-written access class based on Win32 API functions
- n **Registry**
 - à VO: self-written access class based on Win32 API functions
- n **SQL**
 - à VO: direct access to a *Setting* table using the SQL classes
- n **XML**
 - à VO: direct access using the MSXML4 parser via OLE
 - à C#: application settings using System.Configuration

Slide 3 of 21

About the DATA lines

In 1982, the company I worked for used GW-Basic. Back then, screen output on terminals was embedded in escape sequences controlling the position, color and other aspects. Different products had different sets of escape sequences. There was a module with a hard-coded escape sequence set in DATA lines for each terminal we supported. The DATA lines with the control sequences were read upon program initialisation. We only had to link the application with the correct module to produce a version for a particular terminal.

1. History

Problems of the past

- n **Hart-coded settings**
 - à can only be modified by a developer
- n **INI files**
 - à robust and simple
 - à not suited for complex settings like trees
- n **Registry**
 - à §\$%*# registry
- n **SQL**
 - à no connection to the database – no settings (where do you store the connection string then?)
- n **XML**
 - à VO: no native support, permission problems

Slide 4 of 21

XML

Ein Problem war z.B. der relativ naive Ansatz, über den MS XML-Parser direkt in XML-Dateien im Verzeichnis der Applikation zu schreiben, also C:\Programme\MeineApplikation\Config.XML. Das funktioniert solange, wie der Benutzer Schreibrechte auf diesem Verzeichnis hat, die benötigt der XML-Parser nämlich. Viele Firmen haben aber über Policies Standard-Benutzern Schreibrechte auf das Programme-Verzeichnis entzogen.

2. Basics

Application Settings in .NET 2.0

Applies only to windows applications (System.Windows.Forms).

n Design-time support for VB.NET and C#

- à UI for Settings
- à Access to settings in the form editor

n Simple access to settings in the code

- à *My* namespace in VB.NET
- à *Properties* namespace in C#

No code needed for this.

And even more:

n Comprehensive abstract base classes for customized settings

- à System.Configuration namespace

Slide 5 of 21

Note

The classes in System.Configuration have been greatly enhanced in version 2.0 of the .NET framework. You will find a lot of articles and samples on the internet showing you how to deal with the limitations of the .NET framework 1.1. The former version did not support the storage of user scoped settings for example. With VS 2005 and Vulcan.NET you do not need this workarounds or enhancements any more. When you read an article about configuration and application settings, make sure that it is not outdated and has been written based on the 2.0 framework.

The samples in this session have been written for smart windows applications (using the controls from System.Windows.Forms). Configuring ASP.NET applications is a little bit different in certain areas.

2. Basics

Settings Properties

- n **Name**
used to access the settings at runtime.
- n **Type**
of the value. Can be any .NET type.
- n **Scope**
Application or **User**.
- n **Value**

2. Basics

Scopes

n Application

Application settings apply to **all users** of an application.

At runtime, they are **read only**. The application itself cannot change them at runtime for security reasons.

n User

User settings apply for the **current user** (Windows).

The application may change them at runtime. Further distinction:

- à non-roaming (local to the user on the computer, this is the default)
- à roaming (follow the user to any computer in the network)

Of course, application settings can be changed by the user from outside the application by directly modifying the XML configuration file with notepad or any other editor. As long as they have the necessary permissions. In a lot of companies standard users do not have the permission to change files stored underneath the programs path.

2. Basics

User Settings Path

n <app>.config in the program directory

à read-only defaults

n user.config stored underneath *Documents and Settings*

```
private void btnSettingDirectories_Click(object sender, EventArgs e)
{
    StringBuilder msg = new StringBuilder();
    msg.AppendFormat("Roaming: {0}\n", Application.UserAppDataPath);
    msg.AppendFormat("Local: {0}", Application.LocalUserAppDataPath);
    MessageBox.Show(msg.ToString(), "Storage of User Settings");
}
```



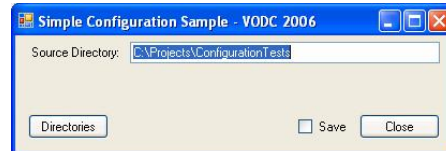
Slide 8 of 21

Note

The name of the company, the name of the application and the version, which are part of the path holding the xml file with the user settings, are retrieved from the attributes in the AssemblyInfo file of the project.

3. Sample: Configuration (almost) without Code

The Sample Application



- n Create a new C# Windows application
- n Fill in **AssemblyInfo.cs!**
 - à Company, application name and version are part of the path containing the user settings
- n Create the Form

Slide 9 of 21

Refactoring Sample

- Renaming the initial form from Form1.cs into MainForm.cs does not only rename the file and the form but rewrite all references to the form in consuming code, for example the instantiation of the form in Program.cs

Notes to the Form Editor

- Watch the positioning assistant while placing the controls in relation to the form and each other
- Try using the Anchor property (used both at design time and runtime)
- Properties: Misc, AutoComplete for a suggestion of directories on the file system

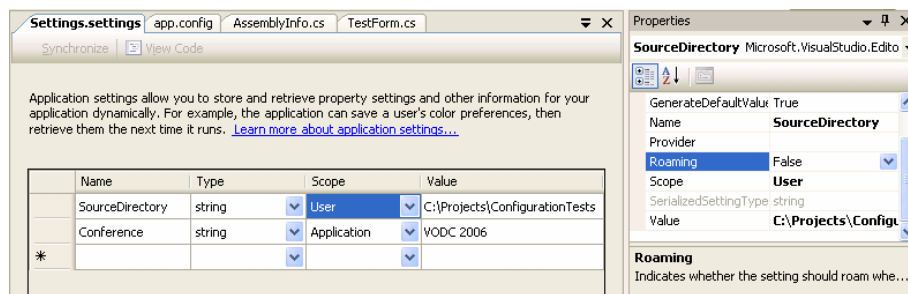
3. Sample: Configuration (almost) without Code

Definition of Settings in Visual Studio

n Editor

à Open via the Properties-Dialog, Settings tab

à Alternatively, double click on Properties, Settings in Solution Explorer



Slide 10 of 21

Synchronize

Using the Synchronize button on the Settings editor, you can delete all current user settings stored on your local machine for the given application (the ones stored underneath *Documents and Settings*).

View Code

Generates a partial Settings class in Settings.cs. Store your own configuration code there (like registering event handlers if you want to be informed about changed settings or if you want to add validators).

3. Sample: Configuration (almost) without Code

Code Generated

n **app.config**

- à XML-configuration file with alle settings defined
- à VS copies this file in the bin directory of the project with the name of the assembly and the extension .config

n **Settings.Designer.cs**

- à Belongs to the namespace *Properties*
- à Generated sealed partial class
- à Singleton
- à Ready to use properties to acces the settings
- à Default values for all settings as attributes (compiled into the assembly)

n **Settings.cs**

- à Second half of the generated partial class
- à for custom code

Slide 11 of 21

Note

The practice to create partical classes is very common to the editors in Visual Studio 2005. The one partial class contains the code generated by the editor and the second partial class, stored in another .cs file, is there to contain your own additional code. In a way this resembles a best practice in Visual Objects: Do not change the code generated by the window editor but write you own inherited class in a second module and place your code there.

3. Sample: Configuration (almost) without Code

app.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup, System, Versio
    <section name="SimpleCSharpSample.Properties.Settings" type="System.Configuration.ClientSe
    </sectionGroup>
    <sectionGroup name="applicationSettings" type="System.Configuration.ApplicationSettingsGroup,
    <section name="SimpleCSharpSample.Properties.Settings" type="System.Configuration.ClientSe
    </sectionGroup>
  </configSections>
  <userSettings>
    <SimpleCSharpSample.Properties.Settings>
      <setting name="SourceDirectory" serializeAs="String">
        <value>C:\Projects\ConfigurationTests</value>
      </setting>
    </SimpleCSharpSample.Properties.Settings>
  </userSettings>
  <applicationSettings>
    <SimpleCSharpSample.Properties.Settings>
      <setting name="Conference" serializeAs="String">
        <value>VODC 2006</value>
      </setting>
    </SimpleCSharpSample.Properties.Settings>
  </applicationSettings>
</configuration>
```

Slide 12 of 21

Wer möchte, kann die XML-Datei auch direkt bearbeiten, Visual Studio bietet eine recht ordentliche Code Completion auch für XML.

3. Sample: Configuration (almost) without Code

Settings.Designer.cs

```
namespace SimpleCSharpSample.Properties {  
  
    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]  
    [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Editors.SettingsDesigner.  
    internal sealed partial class Settings : global::System.Configuration.ApplicationSettingsBase {  
  
        private static Settings defaultInstance =  
            ((Settings)(global::System.Configuration.ApplicationSettingsBase.Synchronized(new Settings())));  
  
        public static Settings Default {  
            get {  
                return defaultInstance;  
            }  
        }  
  
        [global::System.Configuration.UserScopedSettingAttribute()]  
        [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]  
        [global::System.Configuration.DefaultSettingValueAttribute("C:\\Projects\\ConfigurationTests")]  
        public string SourceDirectory {  
            get {  
                return ((string)(this["SourceDirectory"]));  
            }  
            set {  
                this["SourceDirectory"] = value;  
            }  
        }  
    }  
}
```

Slide 13 of 21

Die generierte Settings Klasse vererbt von ApplicationSettingsBase aus dem System.Configuration Namespace und erhält so ihre Grundfunktionalität, z.B. den Indexer für die einzelnen Properties und die Persistenz der Settings über einen Provider.

3. Sample: Configuration (almost) without Code

Using Settings in your Code

You can directly access the settings via the *Properties* Namespace. The class generated by the settings editor is placed in this subnamespace of your project. Sample method from the TestForm Class:

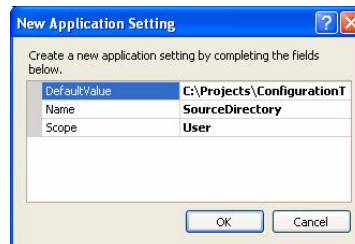
```
private void TestForm_Load(object sender, EventArgs e)
{
    string newText;
    newText = String.Format("{0} - {1}", this.Text, Properties.Settings.Default.Conference);
    this.Text = newText;
}

private void btnClose_Click(object sender, EventArgs e)
{
    if (chkSaveSettings.Checked)
    {
        Properties.Settings.Default.Save();
    }
    Close();
}
```

3. Sample: Configuration (almost) without Code

Using Settings in the Form Editor

- n Select a control
- n **Properties, Data, ApplicationSettings, PropertyBinding, Text**
 - à select the control property to bind to a configuration setting
 - à open the combo box and pick the setting you want to use
 - à you immediately see the default value in the form editor
- n You can also define new settings from within the form editor



3. Sample: Configuration (almost) without Code

Where do the settings come from at runtime

System.configuration looks for the in the following order:

n user.config

- à from a folder underneath *Documents and Settings*
- à only for user settings

n .config in the folder of the executable

n EXE or DLL

- à if no .config is present
- à Default values are stored as attributes in the generated *Settings* class!

4. ConfigurationManager

Starting point for manually coded access to configuration files

n New in .NET 2.0

à Replaces AppSettingsReader (which is deprecated)

n You need to add a reference to system.configuration

n Comprehensive tree of classes

à for Section Groups, Sections, Properties

```
Configuration config =  
    ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);  
  
ConfigurationSectionGroup appSettingsGroup =  
    config.SectionGroups["applicationSettings"];  
  
foreach (ConfigurationSection section in appSettingsGroup.Sections)  
{  
    MessageBox.Show(section.SectionInformation.Name);  
}
```

4. ConfigurationManager

- n Provides methods to write configuration settings
 - à never use XMLReader and XMLWriter for XML configuration files
 - à otherwise you'll face numerous security issues
- n Already works with Vulcan.NET

5. Your own Settings Classes

Demonstration of Bryan Porter's Sample from MSDN

- n Download from msdn.microsoft.com
- n You need to write your own classes that inherit from the base classes in System.Configuration
- n Intensive usage of attributes
- n Generic programming

See the last slide for the download link.

6. More from System.Configuration

Additional functionality:

- n Validators**
 - à Helper classes for validation of values
- n Encrypted configuration files**
 - à DpapiProtectedConfigurationProvider
- n Transition of user settings to a new version of an application**
 - à version number from the assembly settings is part of the folder name
 - à Properties.Settings.Default.Upgrade();
- n Other storages**
 - à Default is LocalFileSettingsProvider
 - à Base class SettingsProvider for customized providers

7. Questions and Discussion



Thank you very much for your time!



FISCHER & CONSULTANTS

Gesellschaft für Softwareentwicklung
und Unternehmensberatung mbH

Martinstrasse 1
44137 Dortmund

www.appfact.com
mifi@appfact.de

Articles

Porter, Bryan: Configure This; Parameterize Your Apps Using XML
Configuration In The .NET Framework 2.0; in MSDN Magazine 06.2006.
<http://msdn.microsoft.com/msdnmag/issues/06/06/ConfigureThis/>

Stoecker, Matt: Using Settings in C#; in Visual Studio 2005 Technical Articles.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/SettingsCS_RL.asp