Настройки приложения. Варианты хранения.

Введение

Необходимость в записи/чтении настроек приложения и какой-либо другой актуальной информации небольшого объема возникает совсем не редко. Разработчики в среде NET, в зависимости от объема данных, их структуры и т.д., имеют несколько вариантов их хранения[1]:

- Базы данных
- Плоские файлы
- Реестр
- файлы в формате INI
- файлы в формате JSON
- файлы в формате XML
- Файлы конфигурации

Основной предмет разговора здесь – файлы конфигурации, поэтому все остальное будет рассмотрено лишь поверхностно и в самом минимальном объеме[2].

Базы данных

Если объем информации значителен и представляет несколько связанных между собой блоков данных, альтернативы базам данных нет.

Плоские файлы

Обычно это текстовые файлы с различными разделителями (запятая, точка с запятой, табулятор и т.д.) представляющими набор записей. В таком файле нет метаданных со структурной информацией, а есть только данные.
Механизмы загрузки, записи, редактирования таких файлов известны и особых проблем эти операции обычно не вызывают. Приведу пример такого простого файла

```
key1,598
key2,Синий луг
key3,Участок
```

[1] Конечно, возможны и другие варианты, например можно сформировать класс собственных настроек и объект этого класса сериализовать/десериализовать по мере необходимости.
[2] Все примеры кода приводятся на VB.NET (применительно к Visual Studio 2010 (VS), Net Framework 4.0). Везде речь идет о приложениях Windows Forms.

Функция чтения подобного файла может выглядеть так

```
Private Function readLines(ByVal pth As String) As Dictionary(Of String, String)
    Dim ss() As String = File.ReadAllLines(pth, System.Text.Encoding.Default)
    Dim sd(1) As String
    Dim dict As New Dictionary(Of String, String)
    For Each s As String In ss
        sd = s.Split(","c)
        dict.Add(sd(0), sd(1))
    Next
    Return dict
End Function
```

Несложно разработать класс со всеми необходимыми процедурами для обработки таких файлов. Добавлю лишь одно замечание по вопросу хранения таких файлов. Совсем не обязательно хранить их по месту размещения приложения (Application.StartupPath) как это обычно бывает, а целесообразно использовать некоторое общее место хранения такой информации. Класс Application предоставляет следующие возможности:

- Свойство Application.UserAppDataPath возвращает строку представляющую путь для данных пользователя. В случае локального пользователя (профиль пользователя хранится в системе, в которую пользователь осуществил вход) используют Application.LocalUserAppDataPath.
- Свойство Application.CommonAppDataPath возвращает строку представляющую путь для данных являющихся общими для всех пользователей.

Путь может быть различным в зависимости от операционной системы и способа развертывания приложения. Если путь не существует, то он создается обычно в следующем формате:
Основной путь \CompanyName \ProductName \ProductVersion.
Типичный основной путь, применительно к WinXP, имеет вид: C:\Documents and Settings \имя_пользователя \Application Data. Или в случае локального пользователя C:\Documents and Settings \имя_пользователя \Local Settings \Application Data.
Для Win7 эти пути будут выглядеть следующим образом:
Application.UserAppDataPath
C:\Users\имя_пользователя\AppData\Roaming\CompanyName\ProductName \ProductVersion
Application.LocalUserAppDataPath
C:\Users\имя_пользователя\AppData\Local\CompanyName\ProductName \ProductVersion
Application.CommonAppDataPath
C:\ProgramData\CompanyName\ProductName\ProductVersion

Реестр

В NET реестром пользуются не часто, скорее редко. Класс System.Windows.Forms. Application позволяет получить доступ к реестру. Так свойство Application. UserAppDataRegistry возвращает объект класса RegistryKey представляющий раздел реестра для данных приложения конкретного пользователя. Если раздел не существует, то он создается в следующем формате: CurrentUser \Software \CompanyName \ProductName \ProductVersion.
Свойство Application. CommonAppDataRegistry возвращает объект класса RegistryKey представляющий раздел реестра для данных приложения, являющихся общими для всех пользователей. Если раздел не существует, то он создается в следующем формате: LocalMachine \Software \CompanyName \ProductName \ProductVersion.
Методы и свойства класса RegistryKey обеспечат чтение и сохранение данных. Классы Registry и RegistryKey относятся к пространству имен Microsoft.Win32.
Свойства и методы для управления реестром можно также получить через объект My.Computer.Registry.

Файлы в формате INI

Файлы этого формата широко использовались во времена DOS, но и сейчас имеют свою нишу и своих почитателей. В конкретном случае нужно оценивать эффективность использования данной конструкции без фанатизма и предубеждений.
Обычный подход это использование функций Win32 API (WritePrivateProfileString, GetPrivateProfileString и т.п.). Но в сети без проблем можно найти различные предложения по обработке файлов этого формата и без использования Win32 API.

Файлы в формате JSON

JSON (JavaScript Object Notation) - простой текстовый формат обмена данными, основан на подмножестве языка программирования JavaScript. Описание формата и документация по последней версии http://json-schema.org. Так как здесь мы рассматривает варианты хранения небольшого объема данных, то можно ограничиться простой структурой объекта (в терминах JSON) который представляет собой неупорядоченный набор пар ключ/значение. Такая конструкция имеет следующий вид

```
{"key1":"598",
"key2":"Синий луг",
"key3":"Участок"
}
```

Для работы с JSON форматом NET предлагает классы пространства имен System.Runtime.Serialization.Json.
Дадим простой пример функции чтения JSON файла данных приведенного выше.

```
Imports System.IO
Imports System.Text
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Json
```

```
…
    Private Sub readJSON()
        Dim JSN As New DataContractJsonSerializer(GetType(jsnInfo))
        Dim obJSN As New jsnInfo
        Dim ss As String = File.ReadAllText("C:\00\tst_01.JSON", System.Text.Encoding.Default)
        Dim st As New MemoryStream(Encoding.UTF8.GetBytes(ss))
        obJSN = JSN.ReadObject(st)
        st.Close()
        MsgBox(String.Format("Json: {0},  value1 =  {1},  value2 =  {2},  value3 =  {3}", ss, obJSN._key1Value,
obJSN._key2Value, obJSN._key3Value))
    End Sub
…

<DataContract()>
Public Class jsnInfo
    <DataMember(Name:="key1")>
    Public _key1Value As String
    <DataMember(Name:="key2")>
    Public _key2Value As String
    <DataMember(Name:="key3")>
    Public _key3Value As String
End Class
```

Учитывая простоту рассматриваемой структуры можно отказаться от использования специализированных классов и реализовать все необходимые функции в виде собственного небольшого класса. Приведу пример подобного класса.

```
Public Class simpleJson
    Private _JJ As Dictionary(Of String, String)
    Public Sub New()
        _JJ = New Dictionary(Of String, String)
    End Sub
    ''' <summary>
    ''' читает содержимое файла Json представленное строкой text
    ''' </summary>
    Public Function readJson(ByVal text As String) As Boolean
        If text.Length = 0 Then Return False
        _JJ.Clear()
        Dim ss() As String = text.Split(","c)
        For Each s As String In ss
            s = s.Replace("{"c, "")
            s = s.Replace("}"c, "")
```

```
            s = s.Replace(vbCrLf, "")
            s = s.Replace(vbCr, "")
            s = s.Replace(ChrW(34), "")
            Dim ss2() As String = s.Split(":"c)
            _JJ.Add(ss2(0), ss2(1))
        Next
        Return True
    End Function
    ''' <summary>
    ''' возвращает текущее представление Json в виде строки
    ''' </summary>
    Public Function writeJson() As String
        If _JJ Is Nothing Then Return ""
        If _JJ.Count = 0 Then Return ""
        Dim s As String = ""
        Dim ss(_JJ.Count - 1) As String
        Dim i As Integer = 0
        Dim ks As Dictionary(Of String, String).KeyCollection = _JJ.Keys()
        For Each k As String In ks
            ss(i) = ChrW(34) & k & ChrW(34) & ":" & ChrW(34) & _JJ(k) & ChrW(34) & "," & vbCrLf
            i += 1
        Next
        ss(0) = "{" & ss(0)
        s = ss(_JJ.Count - 1)
        s = s.Remove(s.Length - 3, 1)
        ss(_JJ.Count - 1) = s & "}"
        s = String.Join("&", ss)
        s = s.Replace("&", "")
        Return s
    End Function
    ''' <summary>
    ''' возвращает массив значений
    ''' </summary>
    Public Function getValues() As String()
        Dim vs As Dictionary(Of String, String).ValueCollection = _JJ.Values()
        Return vs.ToArray
    End Function
    ''' <summary>
    ''' возвращает массив ключей
    ''' </summary>
    Public Function getKeys() As String()
        Dim ks As Dictionary(Of String, String).KeyCollection = _JJ.Keys()
```

```
    Return ks.ToArray
End Function
''' <summary>
''' возвращает значение по ключу key
''' </summary>
Public Function getValue(ByVal key As String) As String
    Dim s As String = ""
    If _JJ.ContainsKey(key) Then
        s = _JJ(key)
    End If
    Return s
End Function
''' <summary>
''' устанавливает значение value по ключу key
''' </summary>
Public Function setValue(ByVal key As String, ByVal value As String) As Boolean
    Dim bb As Boolean = False
    If _JJ.ContainsKey(key) Then
        _JJ(key) = value
        bb = True
    End If
    Return bb
End Function
''' <summary>
''' добавляет запись: значение value с ключом key
''' </summary>
Public Function addKeyValue(ByVal key As String, ByVal value As String) As Boolean
    Dim bb As Boolean = False
    If key.Length > 0 Then
        _JJ.Add(key, value)
        bb = True
    End If
    Return bb
End Function
''' <summary>
''' удаляет запись с ключом key
''' </summary>
Public Function delKeyValue(ByVal key As String) As Boolean
    Dim bb As Boolean = False
    If _JJ.ContainsKey(key) Then
        _JJ.Remove(key)
        bb = True
```

```
        End If
        Return bb
    End Function
End Class
```

Для обработки сложных структур JSON данных можно рекомендовать библиотеку Newtonsoft.Json (http://james.newtonking.com/json). Библиотека занимает порядка 500 Кб.

Файлы в формате XML

В настоящее время файлы в формате XML можно найти везде. Рассматриваемые ниже конфигурационные файлы это тоже XML. Можно сформировать собственную схему и с успехом использовать этот формат с максимальным эффектом.
Для примера приведем код небольшого класса настроенного на работу с файлами следующего вида

```
<?xml version="1.0" encoding="UTF-8"?>
<Config>
      <Категория1>
            <key11>1598</key11>
            <key12>Синий луг</key12>
            <key13>Участок1</key13>
      </Категория1>
      <Категория2>
            <key21>2598</key21>
            <key22>Красный луг</key22>
            <key23>Участок2</key23>
      </Категория2>
      <Категория3>
            <key31>3598</key31>
            <key32>Зеленый луг</key32>
            <key33>Участок3</key33>
      </Категория3>
</Config>
```

Это своего рода INI в формате XML. Так как еще вначале было оговорено, что речь идет о файлах небольшого размера то представляется целесообразным загрузить XML-документ целиком в приложение и далее работать с его элементами. Также класс предоставляет возможность с нуля сформировать документ и сохранить его в формате XML.

```
Imports System.Xml
Public Class XmlForIniInfo
    Private _category As List(Of String)
```

```
Private _record As List(Of catrec)
Private _strRoot As String = "Config"
Private _delim As String = ";"
Private Structure catrec
    Dim nmCat As String
    Dim rec As String
    Dim value As String
    Sub New(ByVal ct As String, ByVal rc As String, ByVal vl As String)
        nmCat = ct
        rec = rc
        value = vl
    End Sub
End Structure

Public Sub New()
    _category = New List(Of String)
    _record = New List(Of catrec)
End Sub

''' <summary>
''' возвращает все строки {категория;ключ;значение}
''' </summary>
Public ReadOnly Property getAll() As String()
    Get
        Dim ss As New List(Of String)
        Dim s As String = ""
        Try
            For Each r As catrec In _record
                s = r.nmCat & _delim & r.rec & _delim & r.value
                ss.Add(s)
            Next
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
        Return ss.ToArray
    End Get
End Property

''' <summary>
''' возвращает строки {ключ;значение} для категории ct
''' </summary>
Public ReadOnly Property getCategory(ByVal ct As String) As String()
```

```
    Get
        Dim ss As New List(Of String)
        Dim s As String = ""
        Try
            Dim rr = From r In _record Where r.nmCat = ct
            For Each r As catrec In rr
                s = r.rec & _delim & r.value
                ss.Add(s)
            Next
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
        Return ss.ToArray
    End Get
End Property

''' <summary>
''' возвращает значение для категории ct и ключа rc
''' </summary>
Public ReadOnly Property getCatRec(ByVal ct As String, ByVal rc As String) As String
    Get
        Try
            Dim rr = From r In _record Where r.nmCat = ct And r.rec = rc
            Return rr.ToArray(0).value
        Catch ex As Exception
            MsgBox(ex.Message)
            Return ""
        End Try
    End Get
End Property

''' <summary>
''' Get/Set имя корневого элемента
''' </summary>
Public Property rootName As String
    Get
        Try
            Return _strRoot
        Catch ex As Exception
            MsgBox(ex.Message)
            Return ""
        End Try
```

```
    End Get
    Set(value As String)
        _strRoot = value
    End Set
End Property

''' <summary>
''' добавить запись: если категории нет то она будет создана;
''' если ключа нет то будет создана новая запись,
''' в противном случае будет отредактировано значение существующей записи
''' </summary>
''' <param name="ct">категория</param>
''' <param name="rc">ключ</param>
''' <param name="vl">значение</param>
Public Sub addRecord(ByVal ct As String, ByVal rc As String, ByVal vl As String)
    Try
        If Not _category.Contains(ct) Then
            _category.Add(ct)
            _record.Add(New catrec(ct, rc, vl))
        Else
            Dim rr = From r In _record Where r.nmCat = ct And r.rec = rc
            If rr IsNot Nothing AndAlso rr.Count > 0 Then
                _record.Remove(rr.ToArray(0))
            End If
            _record.Add(New catrec(ct, rc, vl))
        End If
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub

''' <summary>
''' удалить запись
''' </summary>
''' <param name="ct">категория</param>
''' <param name="rc">ключ</param>
Public Sub delRecord(ByVal ct As String, ByVal rc As String)
    Try
        Dim rr = From r In _record Where r.nmCat = ct And r.rec = rc
        If rr IsNot Nothing Then
            _record.Remove(rr.ToArray(0))
        End If
```

```
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub

''' <summary>
''' читать файл
''' (т.к. документ небольшой то нет особого смысла читать отдельные категории
''' лучше загрузить весь файл)
''' </summary>
''' <param name="pth">полное имя файла</param>
Public Sub readXmlDoc(ByVal pth As String)
    Dim cat As String = ""
    _category = New List(Of String)
    _record = New List(Of catrec)
    Try
        Dim fxml As XDocument = XDocument.Load(pth)
        _strRoot = fxml.Root.Name.ToString
        For Each e As XElement In fxml.Elements.Elements
            cat = e.Name.ToString
            _category.Add(cat)
            For Each r As XElement In e.Elements
                _record.Add(New catrec(cat, r.Name.ToString, r.Value.ToString))
            Next
        Next
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub

''' <summary>
''' сохранить файл
''' </summary>
''' <param name="pth">полное имя файла</param>
Public Sub saveXmlDoc(ByVal pth As String)
    Dim fxml As New XDocument()
    fxml.Declaration = New Xml.Linq.XDeclaration("1.0", "UTF-8", "yes")
    fxml.Add(New XElement(_strRoot))
    Dim ss As String = ""
    Dim xe As XElement
    Try
        For Each s1 As String In _category
```

```
                ss = s1
                xe = New XElement(ss)
                Dim rr = From r In _record Where r.nmCat = ss
                For Each s2 As catrec In rr
                    xe.Add(New XElement(s2.rec, s2.value))
                Next
                fxml.Root.Add(xe)
            Next
            fxml.Save(pth)
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
    End Sub
End Class
```

Файлы конфигурации

Различают файлы конфигурации компьютера (Machine.config), задающие параметры, влияющие на работу компьютера в целом и файлы конфигурации приложений. Для ASP.NET web-приложений это файлы web.config, а для приложений типа Windows Forms - app.config. Все конфигурационные файлы имеют формат XML. Конфигурационные файлы приложений используются для управления сборками, хранения настроек приложения, установки привилегий и т.п.

Добавляем файл app.config к проекту

Рассмотрим, как можно добавить файл app.config к проекту. Возможны следующие варианты.

1. В явном виде через меню Project/Add new item… > General > Application Configuration File.

Имя файла должно быть app.config. По умолчанию в файл записывается элемент `system.diagnostics`, в большинстве случаев его можно безболезненно удалить. Тогда содержимое файла будет иметь вид

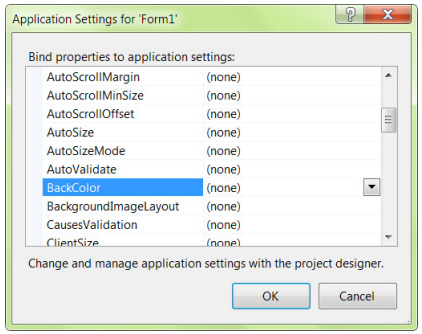
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
</configuration>
```

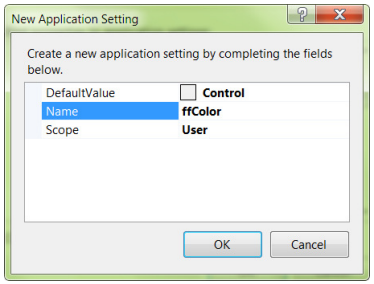
2. В окне Settings проекта добавить настройки приложения.

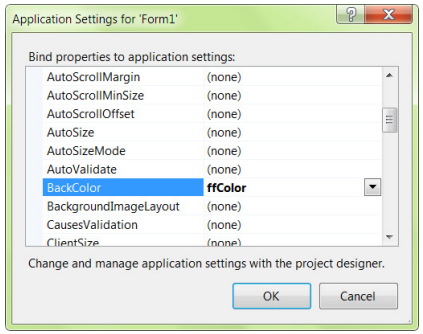
Соответственно будет сформирован файл app.config и добавлен к проекту. В него будут вставлены три элемента (в дополнение к `system.diagnostics`) `configSections`, `userSettings` и `applicationSettings`. Несложно проследить связь этих элементов с объектом My.Settings.

3. Установить в окне Properties настройки (ApplicationSettings) > (PropertyBinding)

Например, для Form1 выбираем Properties/(ApplicationSettings)/ (PropertyBinding)/ BackColor, и далее устанавливаем настройку цвета в соответствии с рисунком ниже.





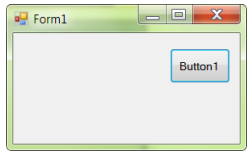


В app.config в разделе userSettings/ИмяПрограммы.My.MySettings появился новый элемент

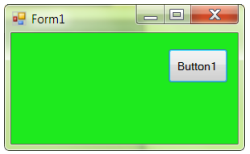
```
<setting name="ffColor" serializeAs="String">
    <value>Control</value>
</setting>
```

Появились изменения и на вкладке Settings проекта

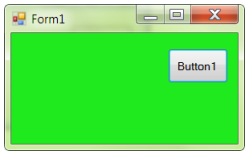
Теперь запустим отладку проекта. Открывается форма – цвет (BackColor) по умолчанию. Программно измененяем цвет формы. Закрываем приложение и заново загружаем его. Цвет открывшейся формы такой же, как перед предыдущим закрытием приложения. Обратите внимание – никаких специальных действий по сохранению настроек приложения не выполнялось. Аналогично можно отслеживать состояние и других объектов и их свойств.



Первая загрузка



Изменили цвет



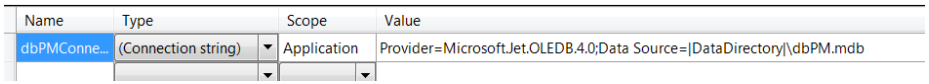
Повторная загрузка

4. Добавить новый источник данных для проекта (Data/Add New Data Source…).

В результате, к проекту будет добавлен файл app.config (если его еще не было) с новым элементом

```
<connectionStrings>
    <add name="testPrg.My.MySettings.dbPMConnectionString" connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=|DataDirectory|\dbPM.mdb" providerName="System.Data.OleDb" />
</connectionStrings>
```

Соответственно в окне Settings проекта добавится новая информация



Работа с файлом конфигурации

Сам файл app.config при работе скомпилированного приложения не используется. При запуске приложения в режиме отладки используется файл testPrg.vshost.exe.config являющийся копией app.config. Аналогично для окончательно скомпилированного приложения используется файл testPrg.exe.config, который должен размещаться в одной папке с файлом приложения testPrg.exe.
При работе с конфигурационными файлами обращаются к пространству имен System.Configuration. Наиболее часто эта работа связана с разделами конфигурации appSettings и connectionString. Раздел appSettings содержит пользовательские настройки (не путать с applicationSettings). Этот раздел может содержать произвольное количество элементов add с парами ключ/значение. Пример записи такого раздела:

```
    <appSettings>
        <add key="NewKey1" value=" 1598" />
        <add key="NewKey2" value=" Синий луг" />
        <add key="NewKey3" value=" Участок1" />
    </appSettings>
```

Раздел connectionString описывает способ соединения с данными. Данный раздел может содержать множество строк подключения.

```
<!-- коллекция строк соединения с базами данных -->
<connectionStrings>
  <add name="ConnStr1" connectionString="LocalSqlServer: data source=127.0.0.1;Integrated Security=SSPI;Initial
Catalog=aspnetdb"
    providerName="System.Data.SqlClient" />
  <add name="ConnStr2" connectionString="Data Source=localhost;uid=sa;pwd=;Initial Catalog=Pubs"
    providerName="System.Data.SqlClient" />
  <add name="ConnStr3" connectionString="LocalSqlServer: data source=127.0.0.1;Integrated Security=SSPI;Initial
Catalog=aspnetdb"
```

```
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

Приведем код небольшого класса для работы с этими данными.

```
Imports System.Collections.Specialized
Imports System.Text
Imports System.Configuration

Public Class UCMan
    Private Sub New()
    End Sub
    ''' <summary>
    ''' возвращает всю коллекцию AppSettings
    ''' </summary>
    Public Shared Function getAllAppSettings() As List(Of String)
        Dim lst As New List(Of String)
        Try
            Dim appSettings As NameValueCollection = ConfigurationManager.AppSettings()
            Dim keys As String() = appSettings.AllKeys
            For i As Integer = 0 To appSettings.Count - 1
                'в формате: номер,имя,значение
                lst.Add(String.Format("{0},{1},{2}", i + 1, keys(i), appSettings(i)))
            Next
        Catch ex As Exception

        End Try
        Return lst
    End Function
    ''' <summary>
    ''' возвращает запись из коллекции AppSettings по ключу (имя)
    ''' </summary>
    ''' <param name="ks">имя</param>
    Public Shared Function getValueAppSetting(ByVal ks As String) As String
        Dim appSettingsValue As String = ""
        Try
            Dim kss() As String = ConfigurationManager.AppSettings.AllKeys
            If kss.Contains(ks) Then
                appSettingsValue = ConfigurationManager.AppSettings(ks)
            End If
        Catch ex As Exception
            appSettingsValue = ""
```

```
    End Try
    Return appSettingsValue
End Function
''' <summary>
''' добавить запись в коллекцию AppSettings
''' </summary>
''' <param name="ks">имя (ключ)</param>
''' <param name="appValue">значение</param>
''' <remarks></remarks>
Public Shared Function addAppSetting(ByVal ks As String, ByVal appValue As String) As Boolean
    Dim bb As Boolean = False
    Try
        Dim config As System.Configuration.Configuration = _
            ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)
        Dim settings = config.AppSettings.Settings
        If IsNothing(settings(ks)) Then
            settings.Add(ks, appValue)
        Else
            settings(ks).Value = appValue
        End If
        config.Save(ConfigurationSaveMode.Modified)
        ConfigurationManager.RefreshSection(config.AppSettings.SectionInformation.Name)
        bb = True
    Catch ex As Exception
        bb = False
    End Try
    Return bb
End Function
''' <summary>
''' удалить запись из коллекции AppSettings
''' </summary>
''' <param name="ks">имя (ключ)</param>
Public Shared Function delAppSetting(ByVal ks As String) As Boolean
    Dim bb As Boolean = False
    Try
        Dim kss() As String = ConfigurationManager.AppSettings.AllKeys
        If Not kss.Contains(ks) Then Return bb
        Dim config As System.Configuration.Configuration = _
            ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)
        config.AppSettings.Settings.Remove(ks)
        config.Save(ConfigurationSaveMode.Modified)
        ConfigurationManager.RefreshSection("appSettings")
```

```
            bb = True
        Catch ex As Exception
            bb = False
        End Try
        Return bb
    End Function
    ''' <summary>
    ''' удалить все записи из коллекции AppSettings
    ''' </summary>
    Public Shared Function delAppSettings() As Boolean
        Dim bb As Boolean = False
        Try
            Dim config As System.Configuration.Configuration = _
                    ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)
            config.AppSettings.Settings.Clear()
            config.Save(ConfigurationSaveMode.Modified)
            ConfigurationManager.RefreshSection("appSettings")
            bb = True
        Catch ex As Exception
            bb = False
        End Try
        Return bb
    End Function
    ''' <summary>
    '''возвращает запись из коллекции ConnectionStrings по ключу (имя)
    ''' </summary>
    Public Shared Function getValueConnectionString(ByVal ks As String) As String
        Dim bb As Boolean = False
        Dim res As String = ""
        Try
            For Each cs As ConnectionStringSettings In ConfigurationManager.ConnectionStrings
                If cs.Name = ks Then Return bb.ToString
            Next
            Dim conValue As String = ConfigurationManager.ConnectionStrings(ks).ConnectionString
            Dim proValue As String = ConfigurationManager.ConnectionStrings(ks).ProviderName
            'в формате: имя,строка подключения,провайдер
            res = String.Format("{0},{1},{2}", ks, conValue, proValue)
        Catch ex As Exception
            res = ""
        End Try
        Return res
    End Function
```

```
    ''' <summary>
    ''' добавить запись в коллекцию ConnectionStrings
    ''' </summary>
    ''' <param name="ks">имя (ключ)</param>
    ''' <param name="conValue">строка подключения</param>
    ''' <param name="proValue">провайдер</param>
    ''' <remarks></remarks>
    Public Shared Function addConnectionString(ByVal ks As String, ByVal conValue As String, ByVal proValue As String) As
Boolean
        Dim bb As Boolean = False
        Try
            For Each cs As ConnectionStringSettings In ConfigurationManager.ConnectionStrings
                If cs.Name = ks Then Return bb
            Next
            Dim res As String = ""
            Dim config As System.Configuration.Configuration = _
                    ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)
            Dim csSettings As New ConnectionStringSettings(ks, conValue, proValue)
            Dim csSection As ConnectionStringsSection = config.ConnectionStrings
            csSection.ConnectionStrings.Add(csSettings)
            config.Save(ConfigurationSaveMode.Modified)
            ConfigurationManager.RefreshSection("connectionStrings")
            bb = True
        Catch ex As Exception
            bb = False
        End Try
        Return bb
    End Function
    ''' <summary>
    ''' для файла machine.config возвращает список имен machineConfig.Sections
    ''' </summary>
    Public Shared Function getMachineConfigSections() As List(Of String)
        Dim machineConfig As Configuration = ConfigurationManager.OpenMachineConfiguration()
        Dim sections As ConfigurationSectionCollection = machineConfig.Sections
        Dim lst As New List(Of String)
        For Each section As ConfigurationSection In sections
            lst.Add(section.SectionInformation.Name)
        Next
        Return lst
    End Function
    ''' <summary>
    ''' возвращает XML-представление для AppSettings раздела конфигурации
```

```
    ''' </summary>
    Public Shared Function getAppSettingsSectionRawXml() As String
        Dim config As System.Configuration.Configuration = _
                ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)
        Dim appSettingSection As AppSettingsSection = _
                CType(config.GetSection("appSettings"), AppSettingsSection)
        Return appSettingSection.SectionInformation.GetRawXml()
    End Function
End Class
```

Класс UCMan содержит функциональность, которая не требует создания экземпляров. Используется закрытый конструктор и все члены объявлены как Public Shared.
Процедуры данного класса:

`getAllAppSettings()`	Возвращает список всех элементов раздела appSettings (ключ/значение).
`getValueAppSetting(key)`	Возвращает значение элемента раздела appSettings по ключу.
`addAppSetting(key, value)`	Добавляет элемент в раздел appSettings. Если раздела нет, он будет создан.
`delAppSettings()`	Удаляет раздел appSettings.
`delAppSetting(key)`	Удаляет элемент раздела appSettings по ключу.
`getAppSettingsSectionRawXml()`	Возвращает раздел appSettings в XML формате.
`getValueConnectionString(key)`	Возвращает значение элемента раздела connectionString по ключу.
`addConnectionString(key, strValue, strProv)`	Добавляет запись в коллекцию ConnectionStrings (имя (ключ), строка подключения, провайдер).
`getMachineConfigSections()`	Для файла machine.config возвращает список имен machineConfig.Sections.

Приложение NET можно установить путем простого копирования всех сборок в одну папку на диске, но при большом числе сборок было бы удобно разместить их в нескольких подкаталогах. Это возможно если включить в файл конфигурации следующий раздел.

```
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="myLib1;myLib2"></probing>
    </assemblyBinding>
  </runtime>
```

Здесь имеется в виду, что поиск необходимых сборок будет выполняться в подкаталогах myLib1 и myLib2 каталога приложения.

Объект My.Settings

Объект My.Settings (объект класса MySettings) предоставляет доступ к параметрам приложения и позволяет динамически хранить и извлекать свойства и другие сведения для приложения. Обычно эти данные размещают через окно Settings проекта (или окно Properties). О добавлении параметров приложения к объекту My.Settings уже говорилось выше. Параметры можно и удалять: в окне Settings проекта нажать правую кнопку мыши на выделенном параметре и в контекстном меню выбрать Remove Setting. Однако следует учесть, что ссылки на параметры приложения в коде и в файле app.config автоматически не удаляются и их придется удалять вручную.
Каждый параметр имеет имя, тип, область действия, значение. Для области действия (Scope) предлагаются два варианта:

- область приложения (Application) – параметры из этой области не могут изменяться во время выполнения приложения;
- область пользователя (User) – параметры из этой области могут изменяться во время выполнения приложения.

Пример записи разделов app.config управляемых через объект My.Settings.

```
  <configSections>
        <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name="testPrg.My.MySettings" type="System.Configuration.ClientSettingsSection, System,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" allowExeDefinition="MachineToLocalUser"
requirePermission="false" />
        </sectionGroup>
        <sectionGroup name="applicationSettings" type="System.Configuration.ApplicationSettingsGroup, System,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name="testPrg.My.MySettings" type="System.Configuration.ClientSettingsSection, System,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
        </sectionGroup>
    </configSections>
    <userSettings>
        <testPrg.My.MySettings>
            <setting name="strUser1" serializeAs="String">
                <value>aaaaaaaaaaaaaaaa</value>
            </setting>
            <setting name="strUser2" serializeAs="String">
                <value>bbbbbbbbbbbbbbb</value>
            </setting>
        </testPrg.My.MySettings>
    </userSettings>
    <applicationSettings>
        <testPrg.My.MySettings>
            <setting name="strAppl1" serializeAs="String">
                <value>uuuuuuuuuuuuuu</value>
            </setting>
        </testPrg.My.MySettings>
```

```
</applicationSettings>
```

Нужно помнить, что если вы переместите приложение на новое место (или на другой компьютер) то измененные и сохраненные вами параметры не будут восстановлены, а будут использоваться параметры, определенные при компиляции приложения. Результаты изменений параметров области User записываются в файлы user.config (для локального пользователя (Win7) нужно искать в папке C:\Users\имя_пользователя\AppData\Local\ CompanyName\ProductName\ProductVersion). Пример файла user.config:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <userSettings>
        <testPrg.My.MySettings>
            <setting name="strUser1" serializeAs="String">
                <value>1111111111111</value>
            </setting>
            <setting name="strUser2" serializeAs="String">
                <value>2222222222222</value>
            </setting>
        </testPrg.My.MySettings>
    </userSettings>
</configuration>
```

Так как класс MySettings является наследуемым от ApplicationSettingsBase то для объекта My.Settings доступны его свойства и методы. Рассмотрим некоторые из них на примере.
Пусть в проекте были определены следующие параметры:

Name	Type	Scope	Value
strAppl1	String	Application	uuuuuuuuuuuuuu
strUser1	String	User	aaaaaaaaaaaaaaaa
strUser2	String	User	bbbbbbbbbbbbbb

1. Вывод текущего значения параметра:
```
TextBox1.Text = My.Settings.strAppl1
TextBox2.Text = My.Settings.strUser1
TextBox3.Text = My.Settings.strUser2
```
2. Изменение значения параметра:
```
My.Settings.strUser1= "1111111111111"
My.Settings.strUser2= "2222222222222"
```

```
My.Settings.Save()
```
[3]

3. Загрузка последнего набора сохраненных значений параметров приложения:
```
My.Settings.Reload()
```
4. Вывод значения параметра по умолчанию (того, что в app.config), текущее значение не изменяется:
```
TextBox1.Text = My.Settings.Properties("strUser1").DefaultValue
```
Результат: `aaaaaaaaaaaaaaaa`
Текущее значение: `1111111111111`
5. Общее число параметров:
```
TextBox1.Text = My.Settings.Properties.Count
```
Результат: 3
6. Вывод списка параметров (имя, значение по умолчанию, текущее значение):
```
Dim s As String = ""
For Each sp As SettingsProperty In My.Settings.Properties
    s &= sp.Name & ", " & sp.DefaultValue & ", " & My.Settings.Item(sp.Name) & vbCrLf
Next
MsgBox(s)
```
Результат:
```
strAppl1, uuuuuuuuuuuuuu, uuuuuuuuuuuuuu
strUser1, aaaaaaaaaaaaaaaa, 1111111111111
strUser2, bbbbbbbbbbbbbbb, 2222222222222
```
7. Установка значений всех user-параметров в начальные значения (те, что заданы в app.config), текущие значение будут потеряны:
```
My.Settings.Reset()
TextBox4.Text = My.Settings.strUser1
```
Результат: `aaaaaaaaaaaaaaaa`

Приложение может реагировать на события изменения параметров, для этого необходимо добавить обработку этих событий. Для открытия файла Settings.vb в редакторе кода нужно в окне Settings проекта нажать кнопку View Code и в открывшемся файле записать код обработки соответствующих событий. События параметров включают следующие:

- Событие SettingChanging инициируется перед изменением значения параметра.
- Событие PropertyChanged инициируется после изменения значения параметра.
- Событие SettingsLoaded инициируется после загрузки значения параметра.
- Событие SettingsSaving инициируется перед сохранением значения параметра.

[3] Метод Save можно и не использовать, если только добавить оператор My.Application.SaveMySettingsOnExit = True или в окне Application проекта установить переключатель Save My.Settings on Shutdown.