HarePoint Explorer for SharePoint Manual

For SharePoint Server 2013, 2010, 2007 and Windows SharePoint Services 3.0.

Product version 2.2.1

July 24, 2017

https://www.harepoint.com

Introduction

HarePoint Explorer for SharePoint is a universal accessory instrument for simplifying the development of solutions for SharePoint Server 2013, 2010, 2007 and Windows SharePoint Services 3.0.

HarePoint Explorer for SharePoint offers two tools to a developer. First one is a browser that allows examining the structure of any object within SharePoint object model. Second one allows creating scripts in C# or Visual Basic using objects selected in the browser. Any number of browser windows and scripts can be opened in one and the same application.

The .NET Reflection technology was widely used for creating this program. All objects are reviewed from these positions. This means that all information obtained in the program can be moved to a developed solution practically without changes.

The described scheme provides the developer with full freedom of action, limited only to his expertise. This approach allows to call absolutely any method of any object and get values of any properties and fields including hidden (internal, private, protected).

Another significant feature of the program is the fact that initially it was created as internal accessory utility constantly used during the development of the HarePoint`s own solution - HarePoint Analytics for SharePoint. All capabilities within the program have appeared only on demand of developers themselves.

Though the product was created for operations with SharePoint object model, it perfectly fits for exploration of any other object model created within the frames of .NET technology! For example, during the creation of HarePoint Analytics for SharePoint, HarePoint Explorer for SharePoint was used for work with the object model of Active Directory, Microsoft Message Queuing and even for taking Microsoft certification exams!

Contents

What's new?

HarePoint Explorer for SharePoint is being updated about every 2-5 months. A list of the recent changes in the program is available as a text file through the Internet here.

HarePoint Explorer for SharePoint - History Log

Version 2.2.0 — released 13 May, 2011

- Scripts runs in async mode.
- Results of script execution can be shown in one window.
- Tab close on middle button click.

Version 2.0.0 — released 30 October, 2009

- Project is completely rewritten!
- Interface is changed.
- Ability to work in asynchronous mode is added.
- Support for Visual Basic is added.
- Automatic generation of script for objects of type SPFarm, SPManager, SPTimerService, SPWebApplication, SPSite and SPWeb is added.
- Ability to call non-public methods is added.
- Support for a national language in the window of script building is added.
- And a number of minor changes that simplifying the work and increase performance.

Version 1.0.4 — released 17 March, 2009

- Small performance improvements.

Version 1.0.3 — released 9 October, 2008

- A possibility to save scripts for repeated use has been added.

Version 1.0.2 — released 2 September, 2008

- A problem with appearance of the "Loading..." string in the node of the object explorer tree which occurred by repeated clicking on the node has been fixed.

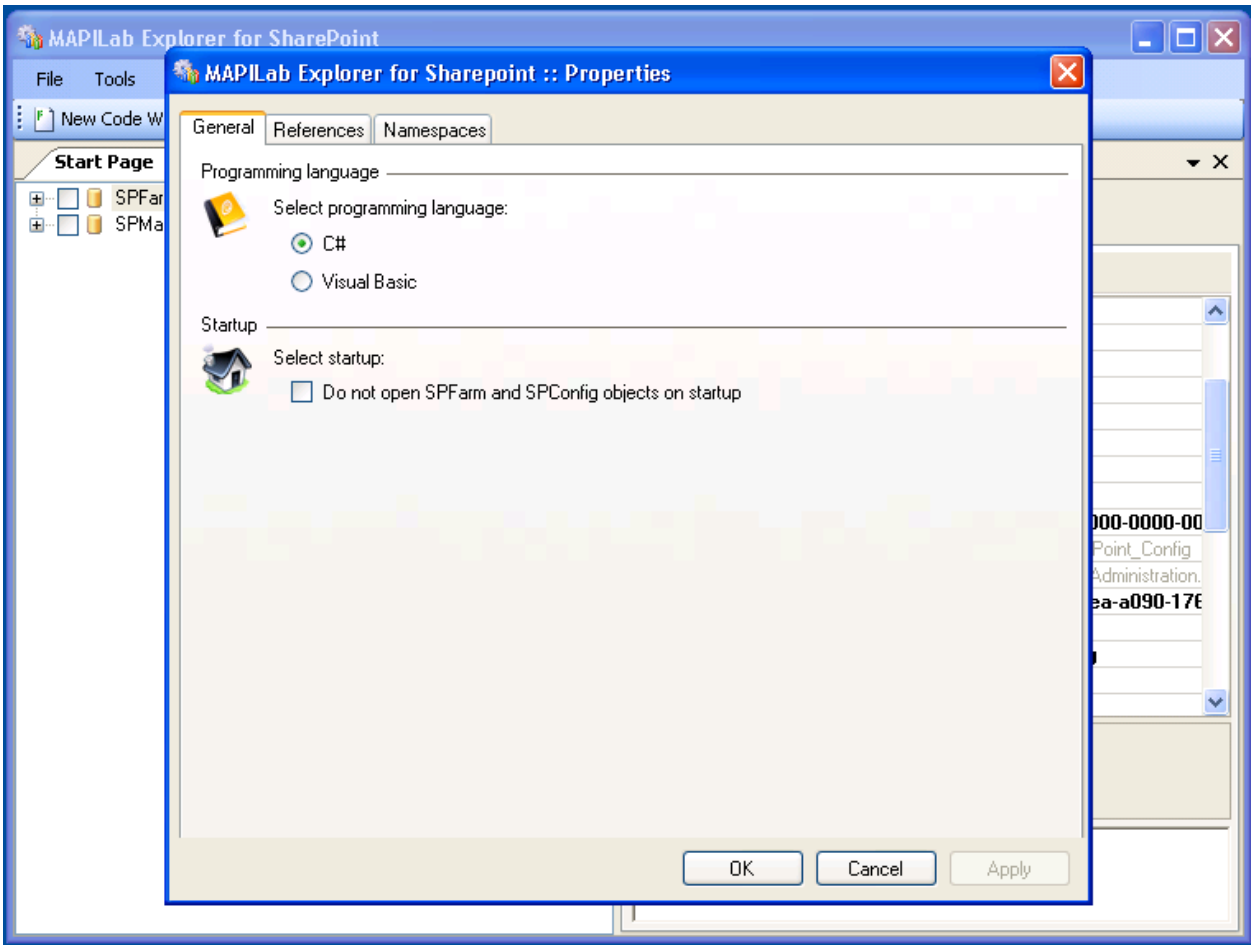
Version 1.0.1 — released 1 August, 2008

- First public version.

Settings

In order to work with the program **HarePoint Explorer for SharePoint** was more comfortable, there are a number of parameters that can be customized.

To open the settings you simply select the **File** menu, click **Properties**.



Programming language

In order to change the programming language, click the **General** tab, and in paragraph **Programming language** select **C#** or **Visual Basic**.

Launch a startup browser window

At your choice, you can cancel the launch of startup browser window with the objects SPFarm and SPManager. To do this, just click the **General** tab, and in paragraph **Startup** check the box next to **Do not open SPFarm and SPConfig objects on startup**.

Selecting default references

You can change default references required to run scripts. To do this, go to the **References** tab and click the buttons **Add** and **Remove**.

Note: When you click on the **Remove** button all selected references will be removed from the list.

Selecting default namespaces

You can change default namespaces required to run scripts. To do this, go to the **Namespaces** tab and click the buttons **Add** and **Remove**.

Note: When you click on the **Remove** button all selected namespaces will be removed from the list.

Asynchronous and synchronous modes

During work in program **HarePoint Explorer for SharePoint**, you could perform a series of actions in synchronous and asynchronous modes.

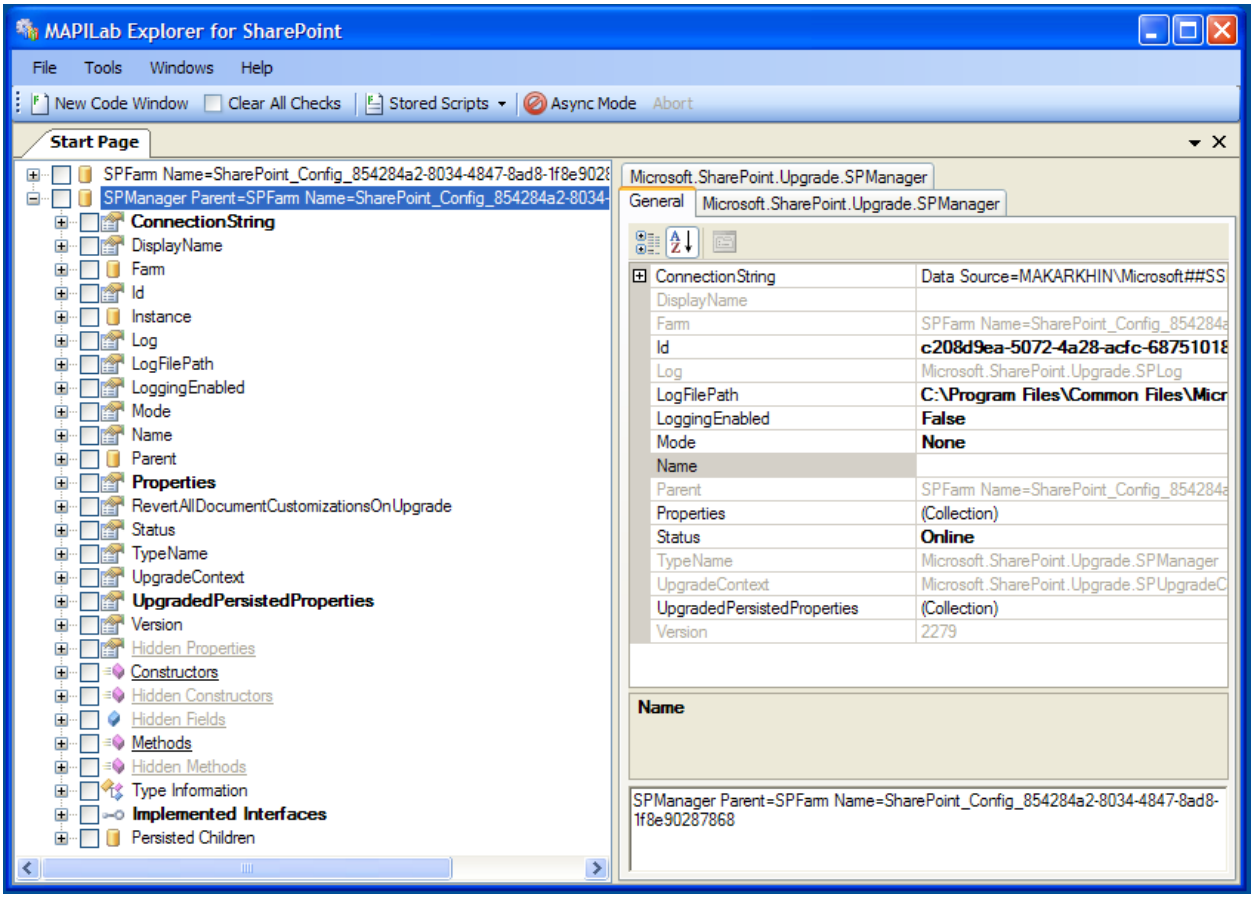
Use of asynchronous mode allows you to prevent hanging of the program and you always have the opportunity to "kill" the process. To do this, use the button **Abort** in the **Tools** menu, which is duplicated on the toolbar.

Unfortunately, not all steps possible to make in asynchronous mode, so it is important to be able quickly switch to synchronous mode. Therefore, while working with the tree object in the **Browser of object model** in the asynchronous mode, on press and hold the **Ctrl** key, the tree will expand in synchronous mode.

Also, for long work in synchronous mode there is a switch (**Async** in the **Tools** menu) that is duplicated on the toolbar.

Browser of object model

The browser of object model displays the structure of any .NET object using the .NET Reflection technology.



Browser window consists of two sections.

In the left window part, there is a tree that shows detailed information of an object. The root node symbolizes the object itself. When the program is launched, browser window with root nodes is opened. First node shows condition of object of type **Microsoft.SharePoint.Administration.SPFarm** (object for work with SharePoint farm), second – **Microsoft.SharePoint.Upgrade.SPManager** (object that provides essential information about configurational database of SharePoint). Name of root object is formed vie the function ToString().

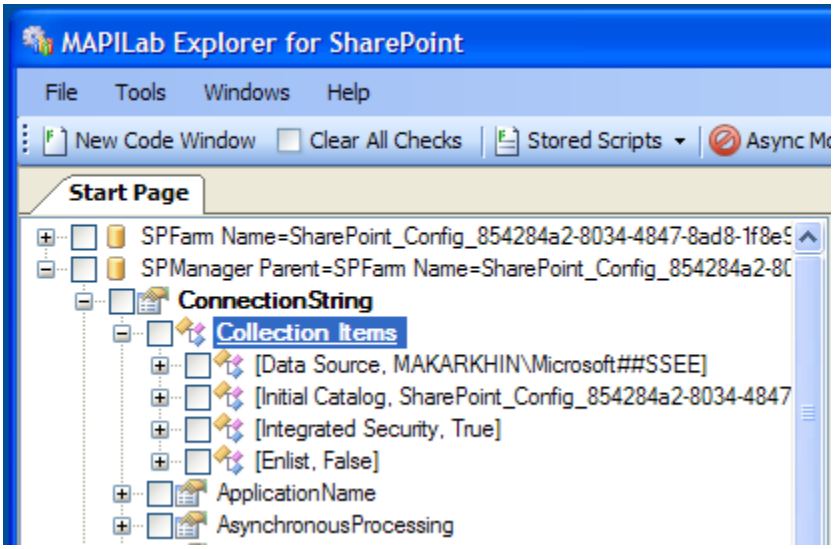
Root node of the object consists of:

1. Detailed list of public properties sorted in alphabetical order.
2. List of hidden properties (private, protected, internal).
3. Lists of public and hidden constructors.
4. Lists of public and hidden fields.
5. Lists of public and hidden methods.
6. Information about type of root object.
7. List of interfaces realized by object.

8. List of child objects (objects inherited from **Microsoft.SharePoint.Administration.SPPersistedObject**). Such objects are marked in the tree with icon .

Each node in the tree can be flagged to select the necessary nodes for further use in a script (see usage of script window).

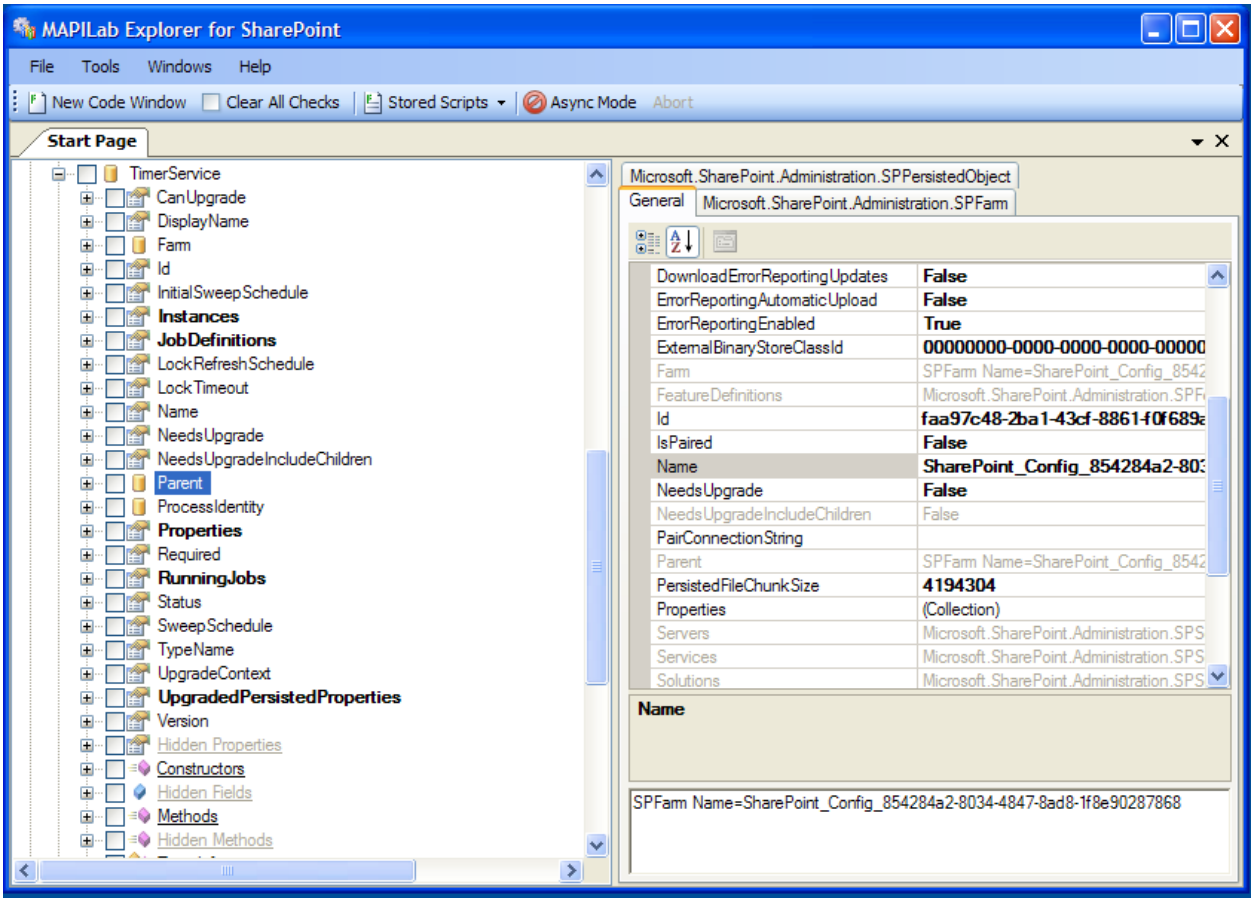
All objects that realize **IEnumerable**, interface are typed with bold font. Besides child nodes listed above, they also contain node **Collection Items**, in which the list of objects obtained by search in collection via methods of interface **IEnumerator** is located. Class **System.String** is exclusion as objects of this type are present everywhere.



In the right section of browser window, you can find a set of tabs, each containing the standard list of properties. Set of tabs includes:

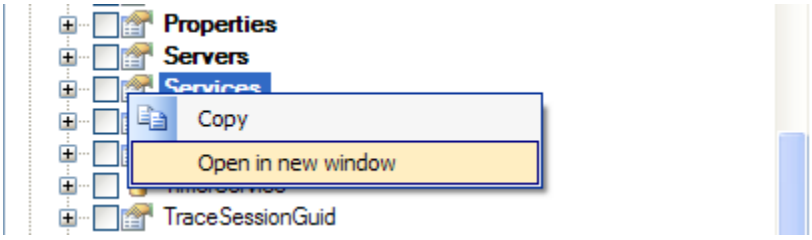
1. **General tab**. Contains object properties that can be changed. Moreover, under object properties, a large text field is located, in which object string representation received via method **ToString**() is shown.
2. **Property tab of object type.** Contains properties of type of object, selected in the tree.
3. **Tab contains information about type of property, field or method** declared in description of root object type.

Example. On the image shown above, property **Parent** of object of type **Microsoft.SharePoint.Administration.SPTimerService.** is selected. In description of class **SPTimerService** , it is said that property **Parent** returns object of type **SPPersistedObject** – exactly this type is presented on the last tab. In our specific case, property **Parent** returned object of type **SPFarm** inherited from **SPPersistedObject** – that is why type **SPFarm** is presented on the second tab.

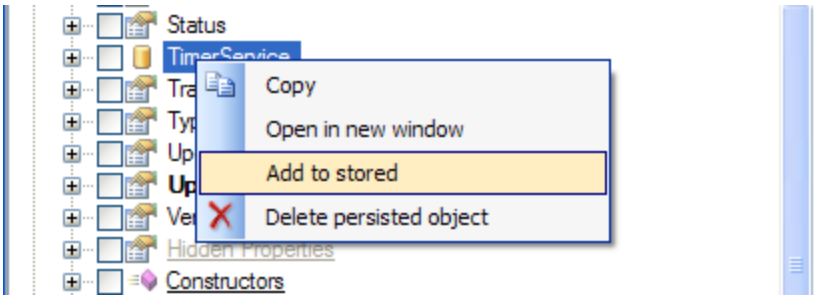


This program feature is useful, if properties return values like null – it can be always seen what they must return.

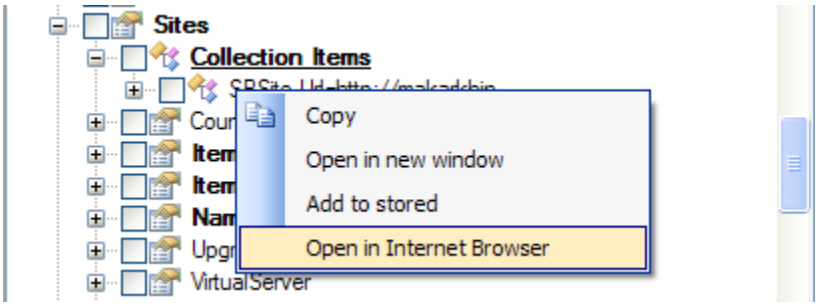
If more detailed examination of any object is needed, it can be opened in separate browser window via context menu item **Open in new window**.



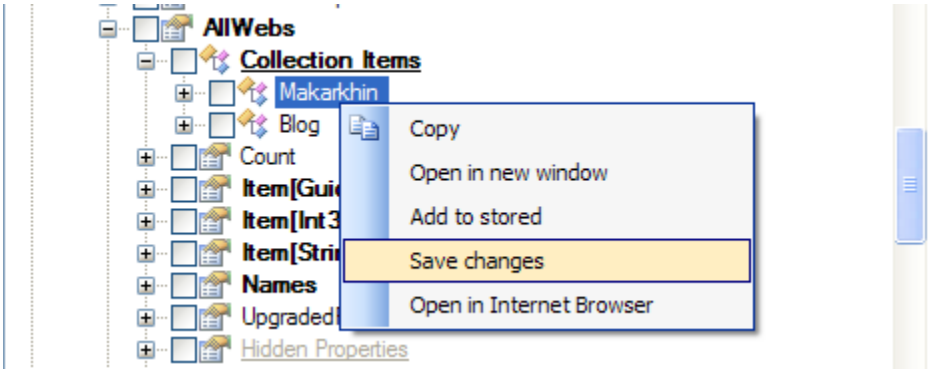
To simplify access to objects such as **SPFarm**, **SPManager**, **SPTimerService**, **SPWebApplication**, **SPSite** and **SPWeb**, you can automatically generate scripts to get them in future work. To do that use the context menu item **Add to stored**.



Also, objects like **SPSite** and **SPWeb** can be accessed from the Internet browser by selecting **Open in Internet Browser** from the context menu.

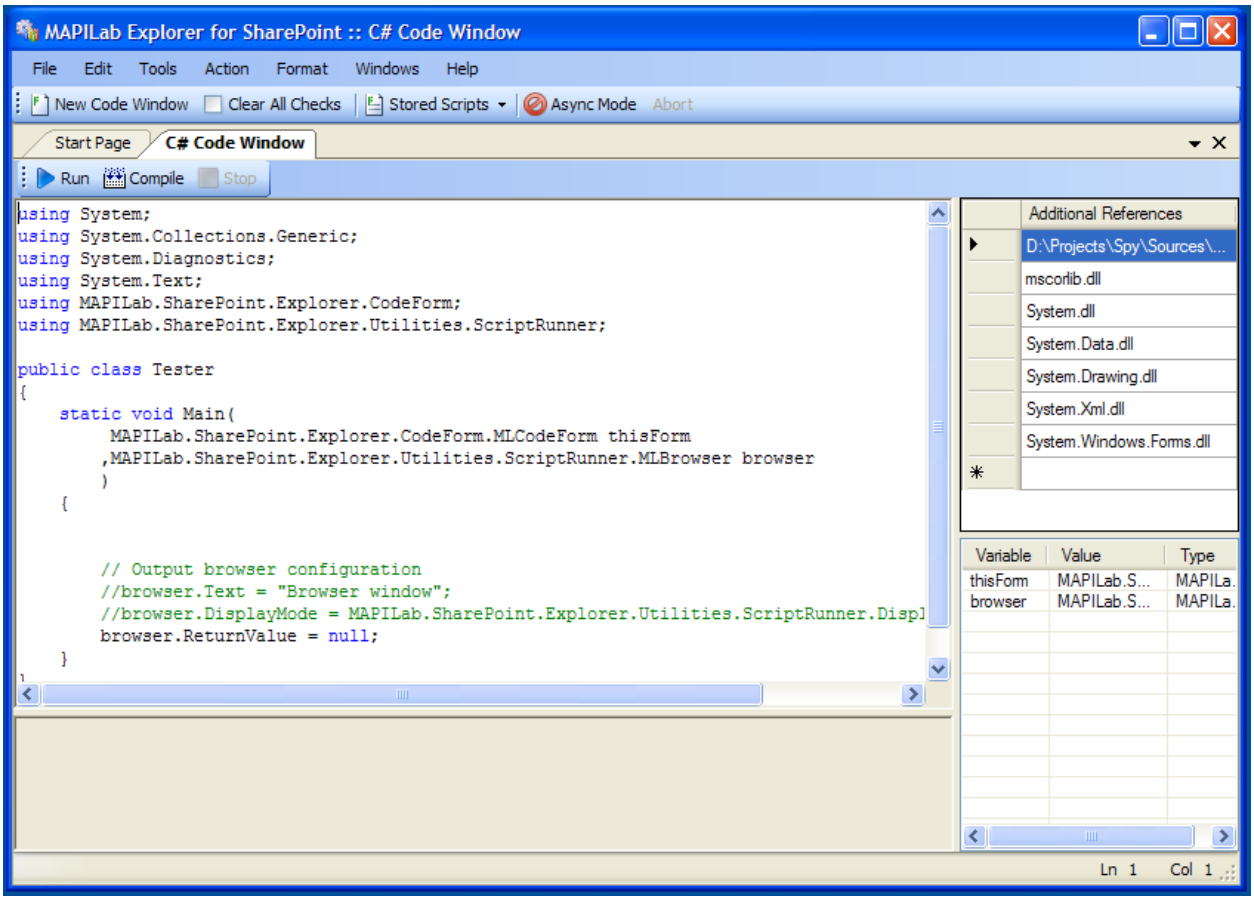


Change the values of the corresponding fields in the tab **General** to change the object properties. You should confirm changes to number of objects (eg, **SPWeb**) by selecting **Save changes** from the context menu.



Window of script building

Actually, it is a window, in which small programs in C# or Visual Basic language can be created using all objects selected in the browser. So the knowledge of programming in C# or Visual Basic is a necessary requirement.



In order to create new script window, select item **Tools** in the menu **New Code Window** or click the corresponding button on the toolbar.

The window is divided into several zones.

In the bottom right part of the window, there is a table, in which names, value and types of variables available for usage in scripts are listed. This list always contains an object of **CodeForm** – it is a type of script window itself and **MLBrowser** - new browser for the return value. Moreover, objects flagged in opened browser windows also appear here.

In the upper right part of the window, there is a list of references to other .NET assemblies necessary for compilation of script programming code. This list can be always completed by placing the full path to fail of necessary assembly in it.

In the central part of the window, editor of programming code in is located. It contains the description of script in C# or Visual Basic. Examples of script building are depicted in item Building scripts.

The lower part of the window is used for outputting results of script compilation and for debugging information.

Building scripts

Building scripts is performed via writing program code in C# or Visual Basic language. All work with scripts is done in the script building window.

When opening window of script creation, HarePoint Explorer for SharePoint generates it`s minimal program code:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
         MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {
        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;
        browser.ReturnValue = null;
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)


        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded
```

```
        browser.ReturnValue = Nothing
    End Sub
End Class
```

This code contains description of class **Tester**, that includes statistical method **Main**.

The method **Main** is a start point for executing the script. Signature of this method is generated automatically on basis of selected objects in browser windows. Not depending on the condition of browser windows, method **Main** always gets parameters **thisForm** and browser. Parameter **thisForm** contains reference to script building window itself. Object returned by method **Main** is displayed in new browser window (except for cases when null reference is returned).

Only name of class **Tester** and signature of method **Main** must remain unchanged in the script programming code – everything else can be changed as you like. It is possible to add descriptions of additional types, modify description of both class **Tester** and body of method **Main**, and etc.

As mentioned before, signature of method **Main** is generated automatically depending on the number of objects flagged in browser windows. Considering the fact that in browser window not only property or field, but also any method can be flagged, the result of generation of method main may strongly vary. To demonstrate this, let`s review two examples.

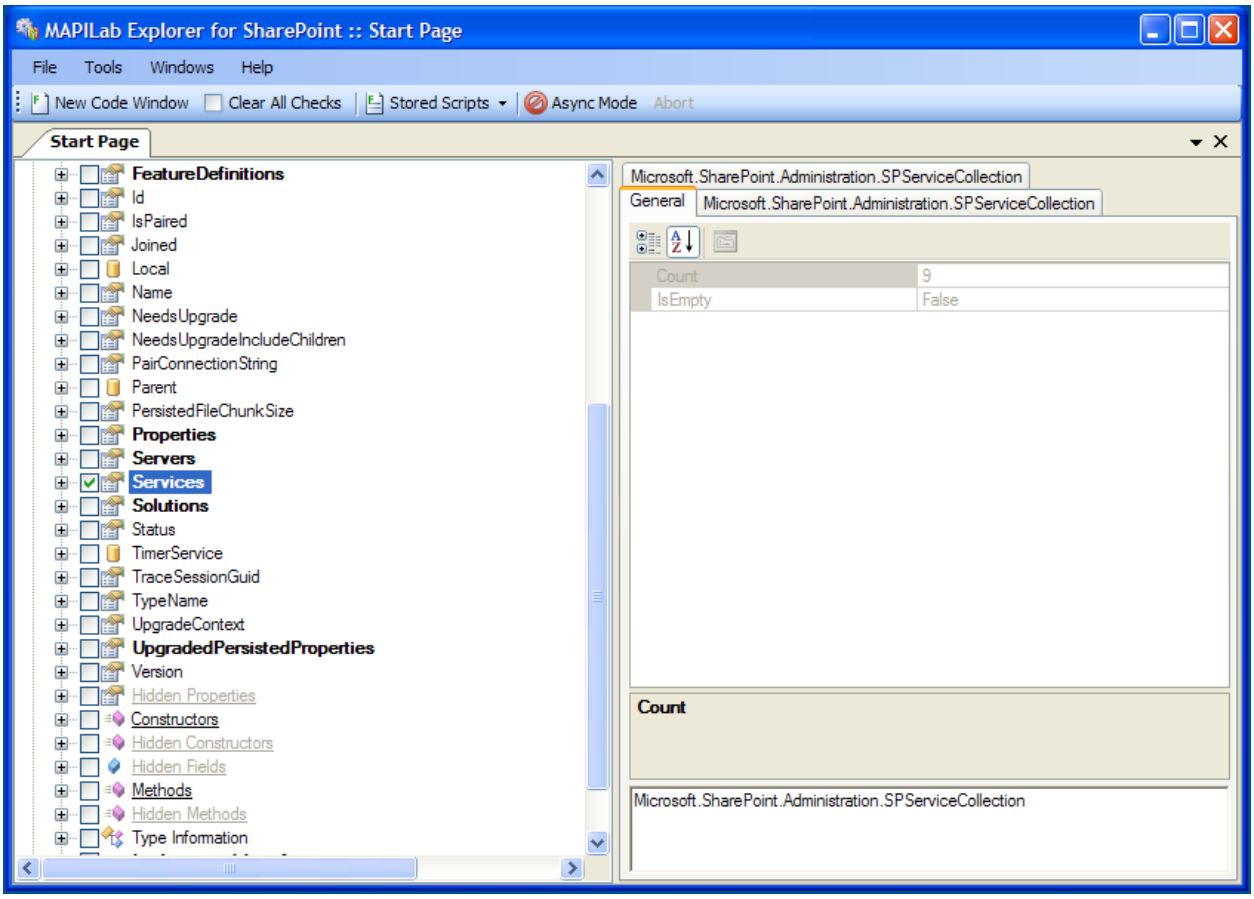
Example of use the parameter browser and its parameters

Parameter **browser** used to visualize and return one or more if necessary facilities during the execution of the script. Parameter browser has the following properties:

1. **Text** - Heading tab of object browser window. If the property is null, by default displays the name of the object or its type.
2. **DisplayMode** - Appearance object mode. Takes the values **Raw** and **Expanded**. By default is Raw. But for the convenience of displaying such objects as collections or arrays of objects you can use **Expanded**.
3. **RaturnValue** - Object(s) to display in the object browser window.

Let us demonstrate the use of the parameter **browser** to display a collection of services of SharePoint farm.

Let the browser window is marked property **Services**.



Create a new script window and execute the following code, which returns a collection of services in **Raw** mode:

C#

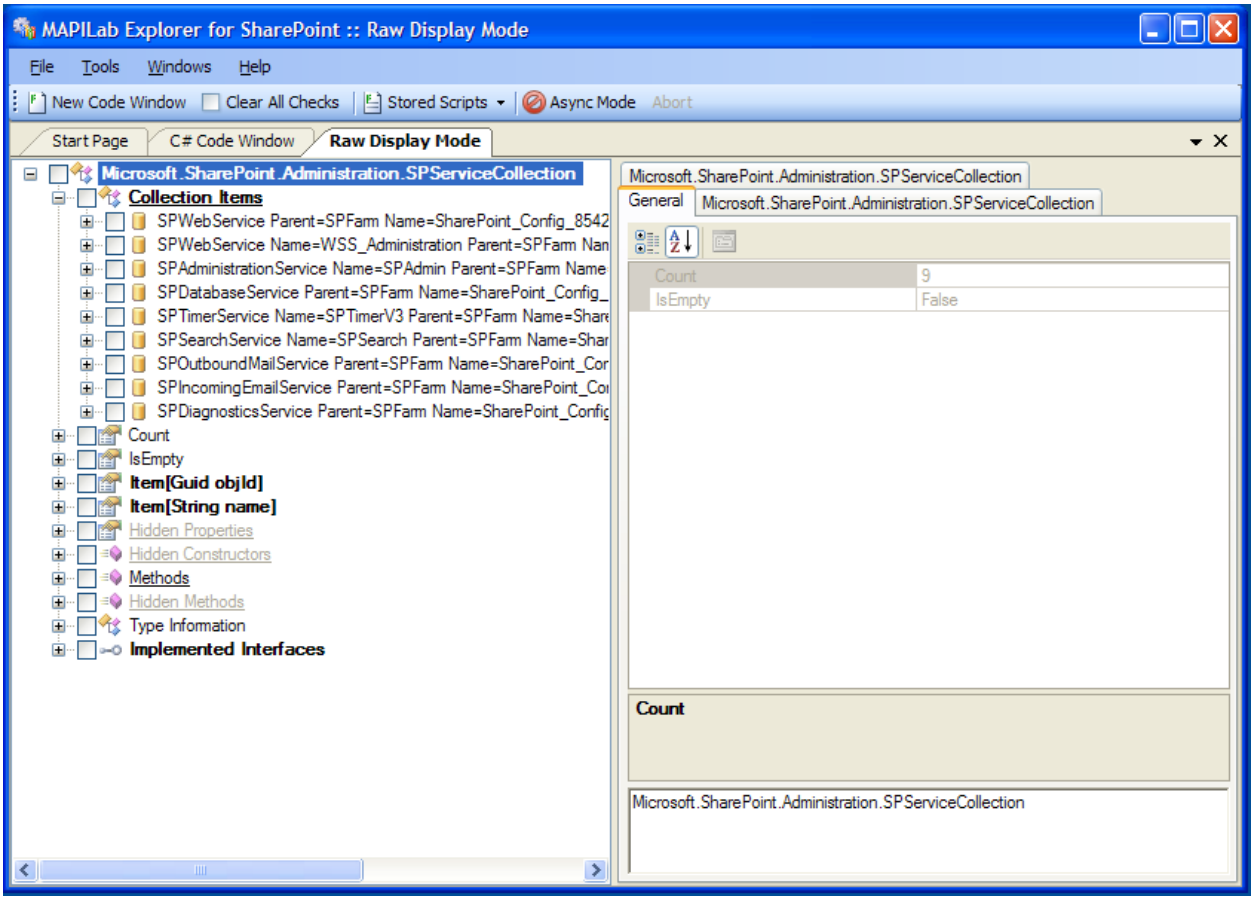
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
         Microsoft.SharePoint.Administration.SPServiceCollection services1
        ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {

        // Output browser configuration
        browser.Text = "Raw Display Mode";
```

```
        browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Raw;
        browser.ReturnValue = services1;
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal services1 As
Microsoft.SharePoint.Administration.SPServiceCollection,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        ' Output browser configuration
        browser.Text = "Raw Display Mode"
        browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Raw
        browser.ReturnValue = services1
    End Sub
End Class
```

As a result of running the script get the following result:



Now execute the following code, which returns a collection of services in **Expanded** mode:

C#

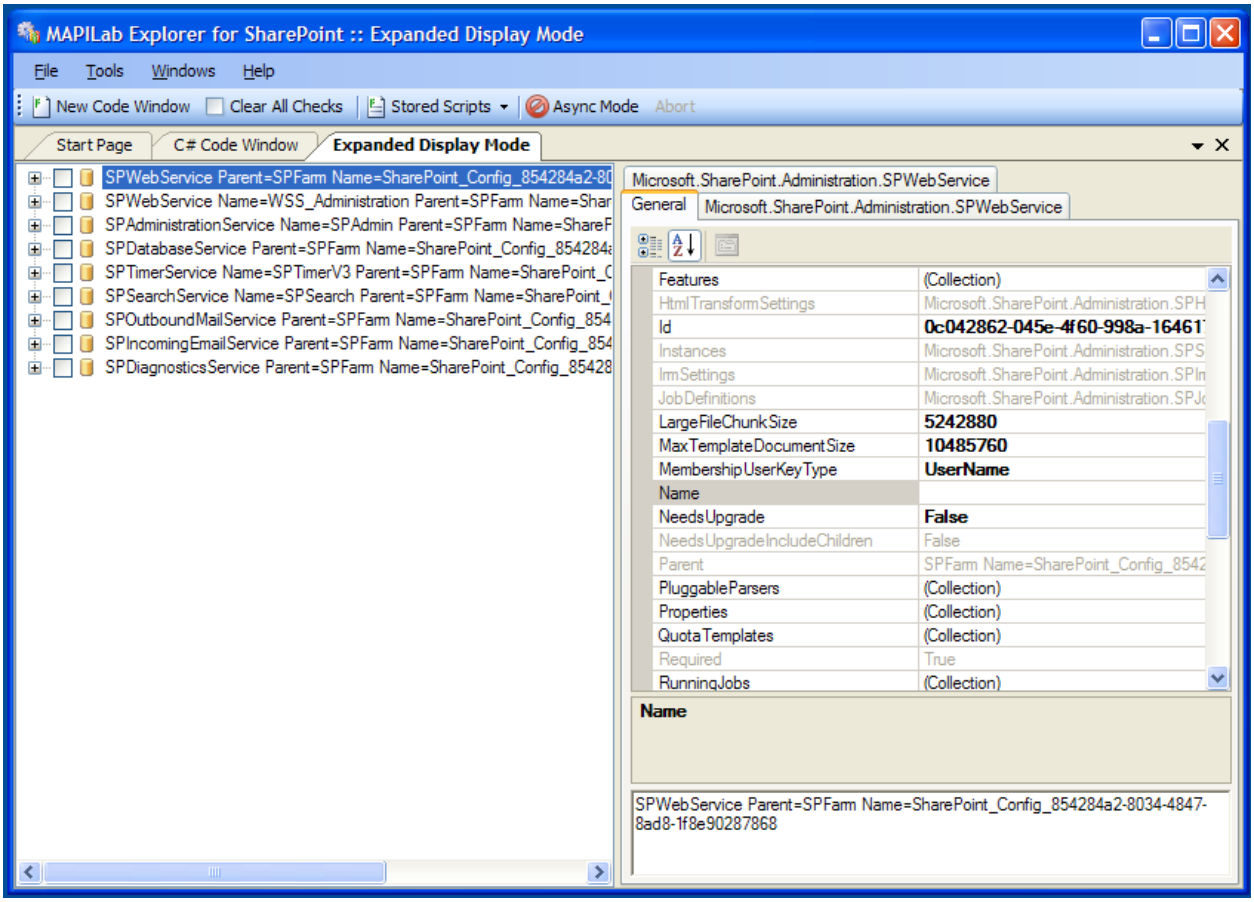
```
 using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
         Microsoft.SharePoint.Administration.SPServiceCollection services1
        ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {

        // Output browser configuration
        browser.Text = "Expanded Display Mode";
        browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;
```

```
        browser.ReturnValue = services1;
    }
}
```

Visual Basic

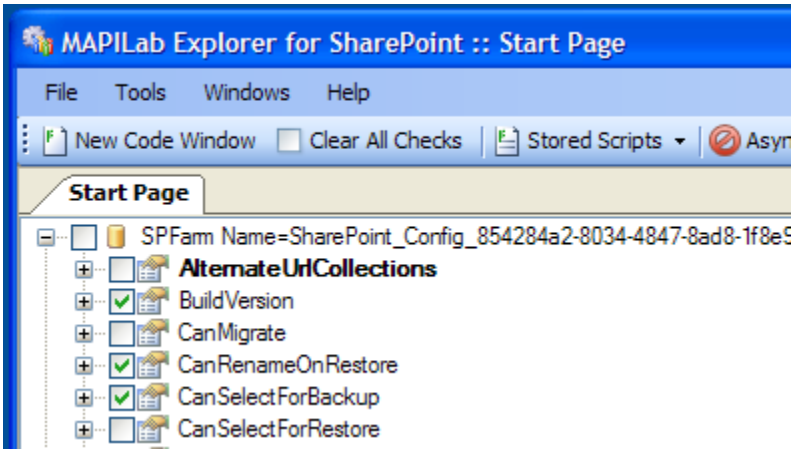
```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal services1 As
Microsoft.SharePoint.Administration.SPServiceCollection,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        ' Output browser configuration
        browser.Text = "Expanded Display Mode"
        browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded
        browser.ReturnValue = services1
    End Sub
End Class
```

As a result of running the script get the following result:



Example of generating method main when selecting property or field

For instance, several properties or fields are flagged in browser window as shown below.



In this case, when creating new script window, method **Main** will look the following way:

C#

```
using System;
using System.Collections.Generic;
```

```
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
         System.Version buildVersion1
        ,bool canRenameOnRestore2
        ,bool canSelectForBackup3
        ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {
        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;

        browser.ReturnValue = null;
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal buildVersion1 As System.Version,
                    ByVal canRenameOnRestore2 As Boolean,
                    ByVal canSelectForBackup3 As Boolean,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded

        browser.ReturnValue = Nothing
    End Sub
End Class
```

As seen in the example, the generated signature of method **Main** contains, besides obligatory parameter thisForm, automatically added parameters **buildVersion1**, **canRenameOnRestore2**, **canSelectForBackup3** relevant to objects selected in browser window.

Let`s perform simple modification of generated method. Place string representation of **SharePoint** version in the header of script window and put string **Current SharePoint version is** … in output box of results:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
         System.Version buildVersion1
        ,bool canRenameOnRestore2
        ,bool canSelectForBackup3
        ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {
        // Put SharePoint version to code windows caption
        thisForm.Text = buildVersion1.ToString();

        // Make debug output
        Debug.Print("Current SharePoint version is {0}", buildVersion1);

        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;

        browser.ReturnValue = null;
    }
}
```

Visual Basic

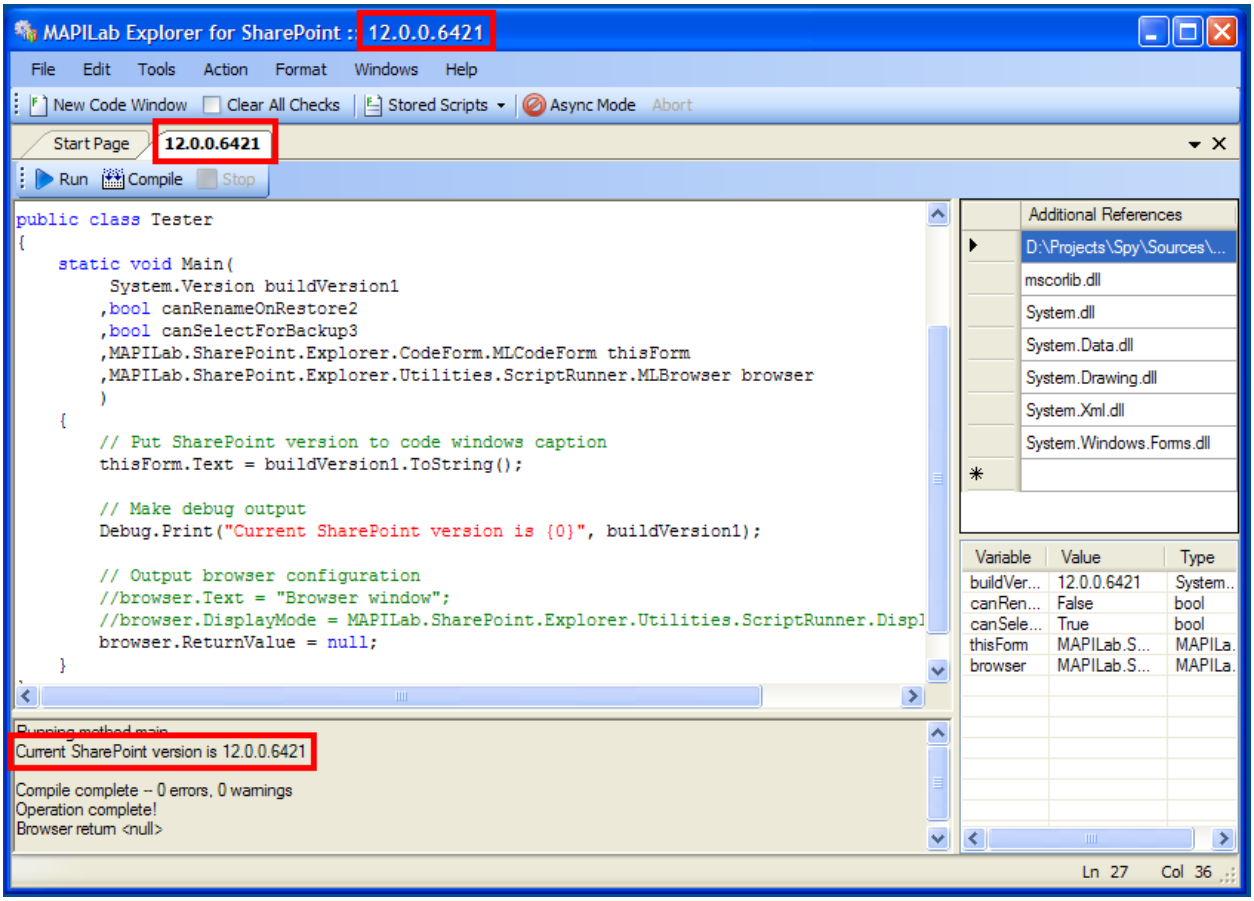
```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
```

```
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal buildVersion1 As System.Version,
                    ByVal canRenameOnRestore2 As Boolean,
                    ByVal canSelectForBackup3 As Boolean,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        'Put SharePoint version to code windows caption
        thisForm.Text = buildVersion1.ToString()

        'Make debug output
        Debug.Print("Current SharePoint version is {0}", buildVersion1)
        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded

        browser.ReturnValue = Nothing
    End Sub
End Class
```

In order to be sure of no errors in the programming code, it can be compiled by clicking buttons **F6**, **F7** or selecting menu item **Action** – **Compile**.

In order to execute script, click button **F5** or select menu item **Action** – **Run**.

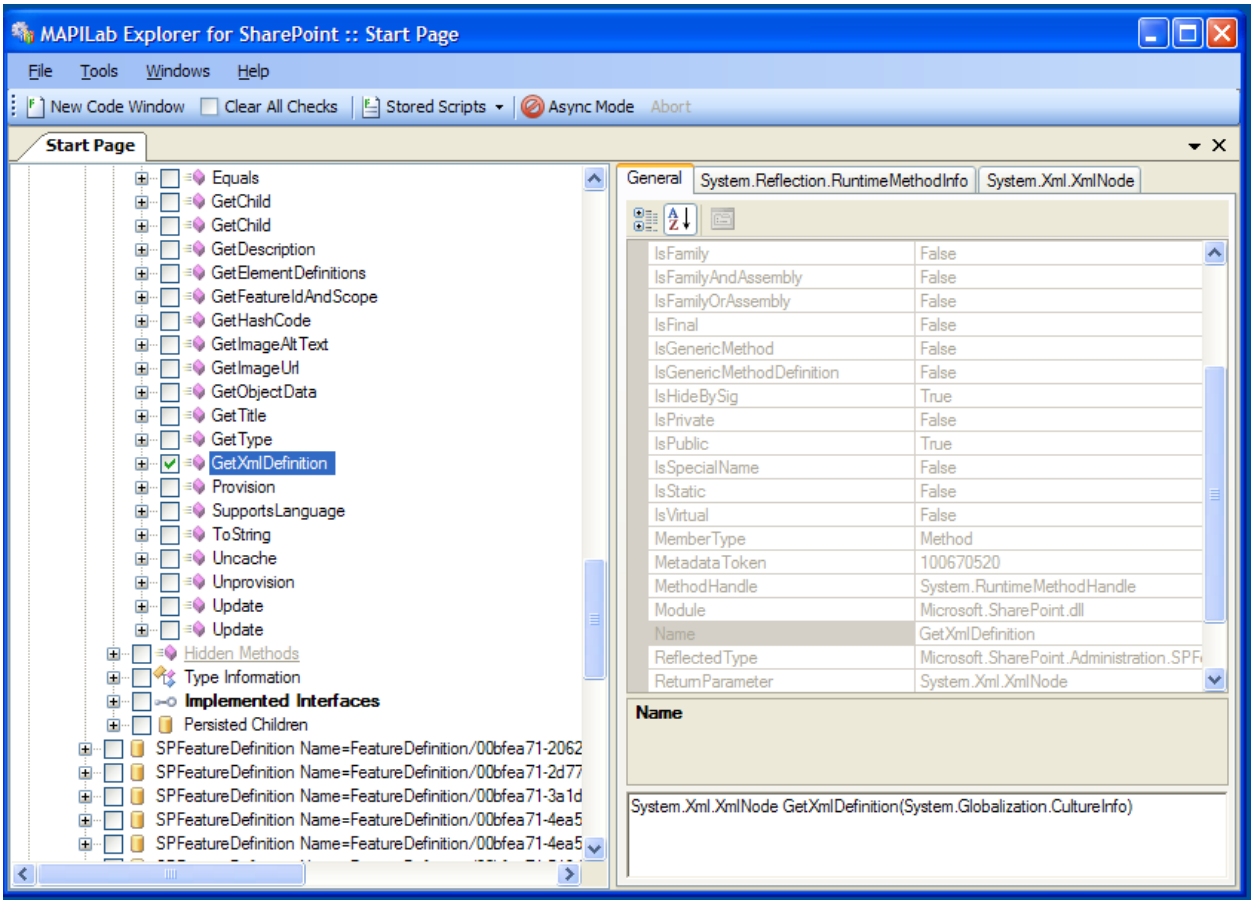
After executing the given script, window will become as follows (pay attention to points marked with red rectangles):



Example of generating method main when selecting method

When some method is selected in the browser tree, not only signature of method Main, but also the part of its part is generated – template of call of selected method is put in it beforehand. Let`s demonstrate it.

Assume that one of methods (one for simplicity) is flagged in browser window as shown on the image below (by the way, method selected for the example allows receiving description of SharePoint feature in XML format).



In this case, when creating new script window, method **Main** will be the following:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
         Microsoft.SharePoint.Administration.SPFeatureDefinition
mo_getXmlDefinition1
        ,System.Reflection.MethodInfo getXmlDefinition1
        ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {
        // Calling method GetXmlDefinition
        System.Globalization.CultureInfo value1 = ;
```

```
        System.Xml.XmlNode value2 =
            mo_getXmlDefinition1.GetXmlDefinition(
                value1);

        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;
        browser.ReturnValue = null;
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal mo_getXmlDefinition1 As
Microsoft.SharePoint.Administration.SPFeatureDefinition,
                    ByVal getXmlDefinition1 As System.Reflection.MethodInfo,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        ' Calling method GetXmlDefinition
        Dim value1 As System.Globalization.CultureInfo =
        Dim value2 As System.Xml.XmlNode =
mo_getXmlDefinition1.GetXmlDefinition(value1)

        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded

        browser.ReturnValue = Nothing
    End Sub
End Class
```

As seen in the example, besides object of type **MethodInfo** selected in the browser, the list of parameters of method **Main** includes also parameter **mo_getXmlDefinition1** containing reference to an object that must be called. Moreover, method body already contains template that can simplify the call of selected method. It remains only to supply it (actually to define value of variable **value1**) and use the result of call that will be placed in variable **value2**. Modify the code for xml describing property of

SharePoint feature to be shown in pop-up window, and after that – object of type **XmlNode** to appear in new browser window:

C#

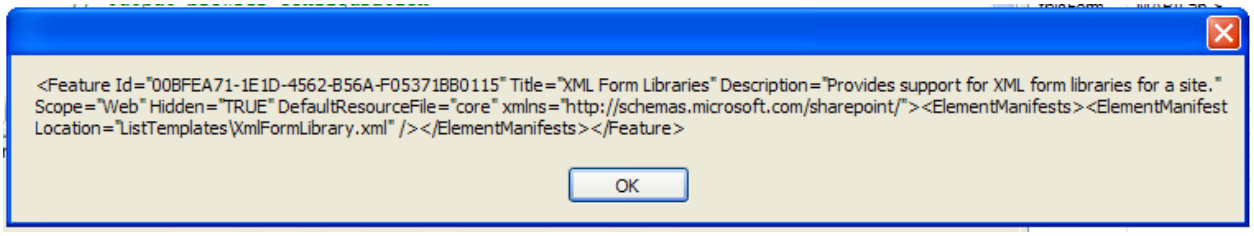
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
         Microsoft.SharePoint.Administration.SPFeatureDefinition
mo_getXmlDefinition1
        ,System.Reflection.MethodInfo getXmlDefinition1
        ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {
        // Calling method GetXmlDefinition
        System.Globalization.CultureInfo value1 =
System.Threading.Thread.CurrentThread.CurrentCulture;

        System.Xml.XmlNode value2 =
            mo_getXmlDefinition1.GetXmlDefinition(
                value1);

        // Show Feature XML using MessageBox class
        System.Windows.Forms.MessageBox.Show(value2.OuterXml);

        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;

        // Return results to the new object browser
        browser.ReturnValue = value2;
    }
}
```

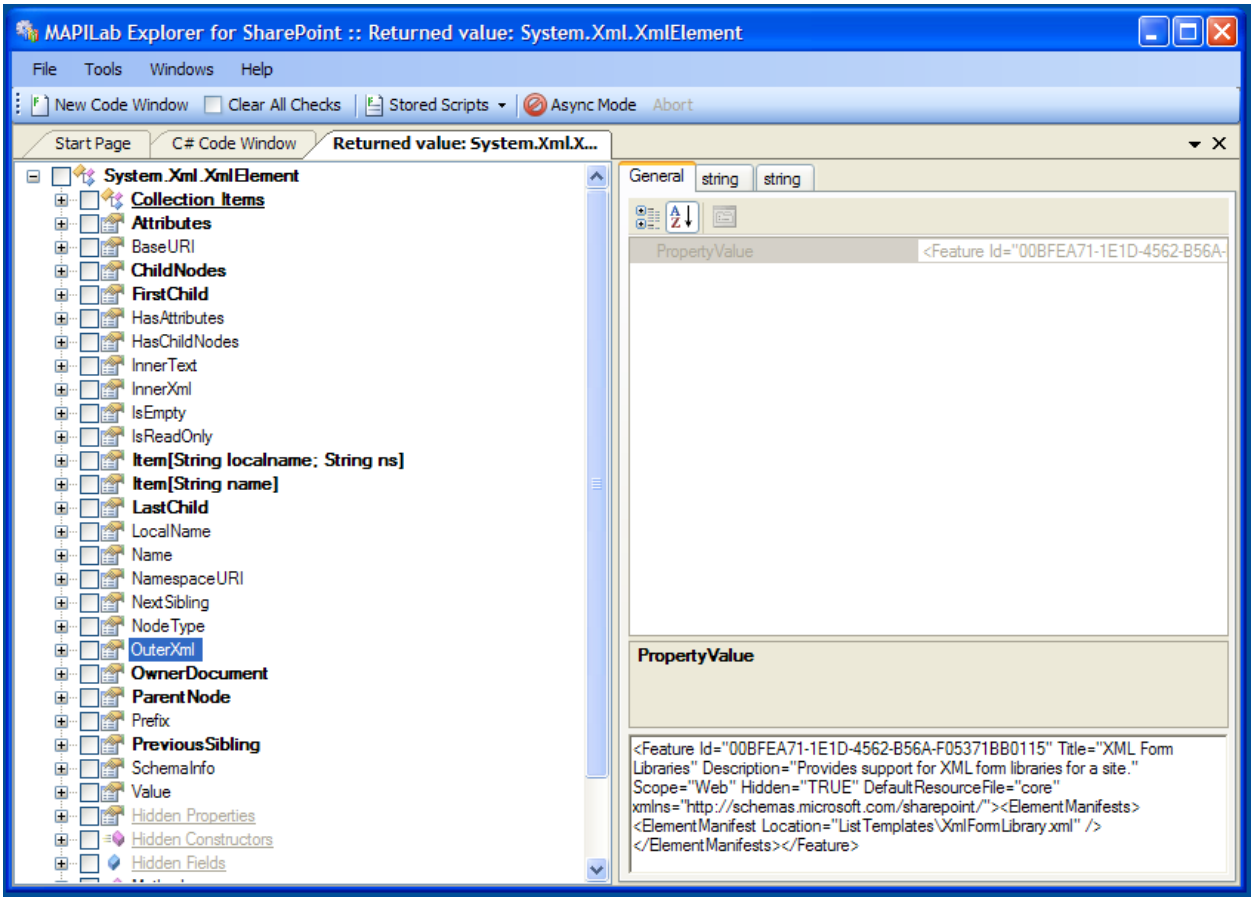
Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner
```

```
Public Class Tester
    Shared Sub Main(ByVal mo_getXmlDefinition1 As
Microsoft.SharePoint.Administration.SPFeatureDefinition,
                    ByVal getXmlDefinition1 As System.Reflection.MethodInfo,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        ' Calling method GetXmlDefinition
        Dim value1 As System.Globalization.CultureInfo
        value1 = System.Threading.Thread.CurrentThread.CurrentCulture

        Dim value2 As System.Xml.XmlNode
        value2 = mo_getXmlDefinition1.GetXmlDefinition(value1)

        'Show Feature XML using MessageBox class
        System.Windows.Forms.MessageBox.Show(value2.OuterXml)

        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded

        'Return results to the new object browser
        browser.ReturnValue = value2
    End Sub
End Class
```

As a result of script execution (button **F5**) we will sequentially see:





Here we have reviewed examples of operations with properties or methods separately, however nothing prevents from selecting both of them in the browser simultaneously. In this case, signature of method Main, which includes both first and second variant, will be generated.

Operations with persisted objects

Windows SharePoint Services 3.0 and Microsoft Office SharePoint Server contain a lot of different settings. These are settings of various services, connection strings for databases, indexing parameters, and administrative settings – this list can be continued for a long time.

Storage of majority of SharePoint settings is realized very smartly – developers used the opportunity that .NET objects can be easily serialized and deserialized. Data stream obtained after serialization is saved in the Objectstable of the configurational database (as a rule, it`s name starts with **SharePoint_Config__**). If restoration of saved object condition is necessary, saved data is read from the mentioned table and used for deserialization.

All types saved in the configurational database are inherited from the type **Microsoft.SharePoint.Administration.SPPersistedObject**. These types can be developed either by the Microsoft Company itself, or by any other third-party developers.

Persisted objects form a hierarchy. The base of this hierarchy is an object of type **SPFarm** – SharePoint farm object. It is simple to make sure of that by selecting objects with **Id** = **ParentId** from the table. Because of this, hierarchy of persisted objects always can be presented similarly to Windows registry or hierarchy of Active Directory objects.

In the browser, all objects inherited from **SPPersistedObject** are marked with icon  .

Due to special importance of this object type, HarePoint Explorer for SharePoint offers three modes of operations with them:

1. Access to parent object by means of property **SPPersistedObject.Parent**.
2. Access to child objects by means of special node in the object tree – **Persisted Children**.
3. Creation of script for direct access to persisted object.

As first and second modes needn`t to be commented, let us pay attention to the third mode. It is very important, since, as a rule, it is used when developing solutions for SharePoint.

Suppose that you need to know exactly which folder is used for storing log-files. By performing a simple search in documentation for Windows SharePoint Services 3.0 included in Windows SharePoint Services 3.0 SDK, we discover that path to folder of log-files is kept in property **LogLocation** of class **SPDiagnosticsService**. There we also find that class **SPDiagnosticsServices** has the following constructor:

```
SPDiagnosticsService (String, SPFarm)
```

Such constructor has all persisted objects of SharePoint. Since there can be only one object **SPDiagnosticsServices** in SharePoint farm, parameter name is not needed for its creation – it can equal to "". Second parameter – parent – is necessary; it can be taken from object browser by flagging object of type **SPFarm**.

After flagging object of type **SPFarm** and creating new script window, the program generates the following script code:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
         Microsoft.SharePoint.Administration.SPFarm sPFarmNameSharePoint1
        ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {


        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;
        browser.ReturnValue = null;
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal sPFarmNameSharePoint1 As
Microsoft.SharePoint.Administration.SPFarm, ByVal thisForm As
        MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm, ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)


        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded
        browser.ReturnValue = Nothing
    End Sub
End Class
```

Modify it so that, when executing it, object **SPDiagnosticsServices** is created, value of property LogLocation is place in output box, and method main will return created object for further investigation:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;
using Microsoft.SharePoint.Administration;

public class Tester
{
    static void Main(
         Microsoft.SharePoint.Administration.SPFarm sPFarmNameSharePoint1
        ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {
        // Create new SPDiagnosticsService object
        SPDiagnosticsService diagService = new SPDiagnosticsService("",
sPFarmNameSharePoint1);

        // Output log-files location
        Debug.Print("Log-files store: {0}", diagService.LogLocation);

        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;
        browser.ReturnValue = diagService;
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner
Imports Microsoft.SharePoint.Administration

Public Class Tester
    Shared Sub Main(ByVal sPFarmNameSharePoint1 As
Microsoft.SharePoint.Administration.SPFarm, ByVal thisForm As
```

```
        MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm, ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)
        ' Create new SPDiagnosticsService object
        Dim diagService As SPDiagnosticsService = new
SPDiagnosticsService("", sPFarmNameSharePoint1)

        ' Output log-files location
        Debug.Print("Log-files store: {0}", diagService.LogLocation)

        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded
        browser.ReturnValue = diagService
    End Sub
End Class
```

As a result of script execution, the following information will be placed in output box:

```
 Build started...
Compile complete -- 0 errors, 0 warnings
Running method main...
Log-files store: C:\Program Files\Common Files\Microsoft Shared\Web Server
Extensions\12\LOGS\
Operation complete!
```

After that, the object diagService itself will be shown in new browser window.

Stored Scripts

If you use HarePoint Explorer for SharePoint, from time to time you have to carry out one and the same operations. For example, to find a SPWeb object, you need to open many nodes in the browser of the object model.

To simplify this and many other routine operations, you could store necessary scripts for their further execution in the program.

Stored scripts are created in the same way as context ones are designed. To create such a script you need to press the **New Code Window** button located on the toolbar of the program.

To save this script please go to the menu File and then **Save As Stored**....

To execute the stored script, you need to select the Stored Scripts option in the menu **File**.

Please pay your attention that stored scripts cannot be created for the objects that are marked in the browser of the object model. That is why we recommend you to remove all selections with the help of the **Clear All Checks button**.

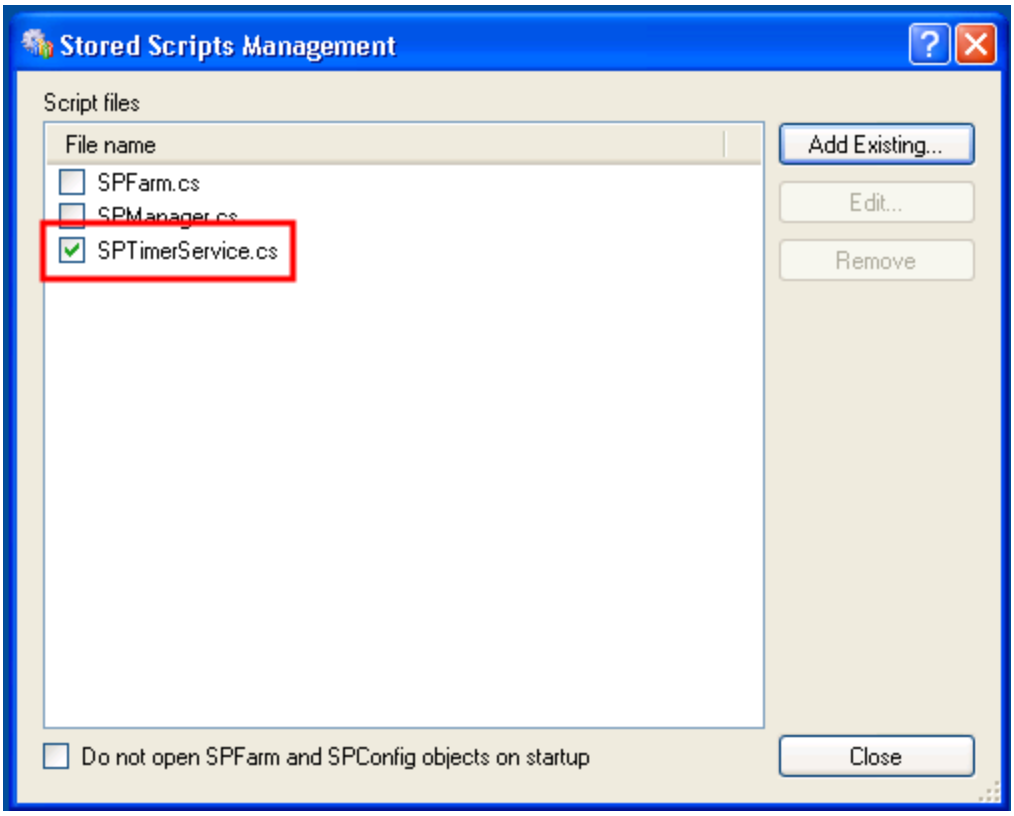
When you save the stored script, a list of references on other .NET assembly is placed to its heading: **///addref<reference name> (///addref<mscorlib.dll>**) for C# and **'addref<reference name>** (**'addref<mscorlib.dll>**) for Visual Basic.

Management of stored scripts

If you would like to look through a list of stored scripts, you need to open the **Stored Scripts Management** dialog using the following path **File - Stored Scripts - Manage**....

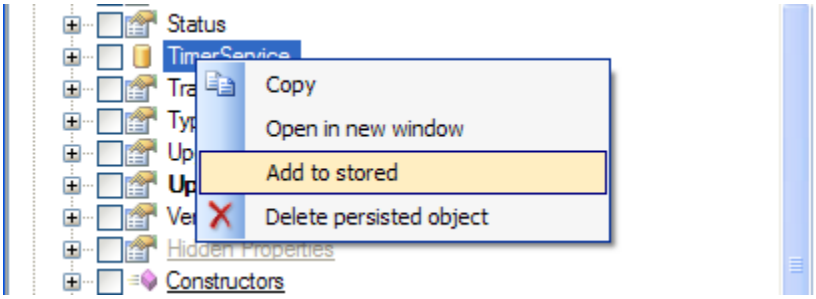
Here you can also add your stored scripts that were deleted earlier from this list, remove scripts from the list or change the text of the stored script.

If you would like to start the script automatically when you start your computer, please mark it with the flag.



Automatic script creation

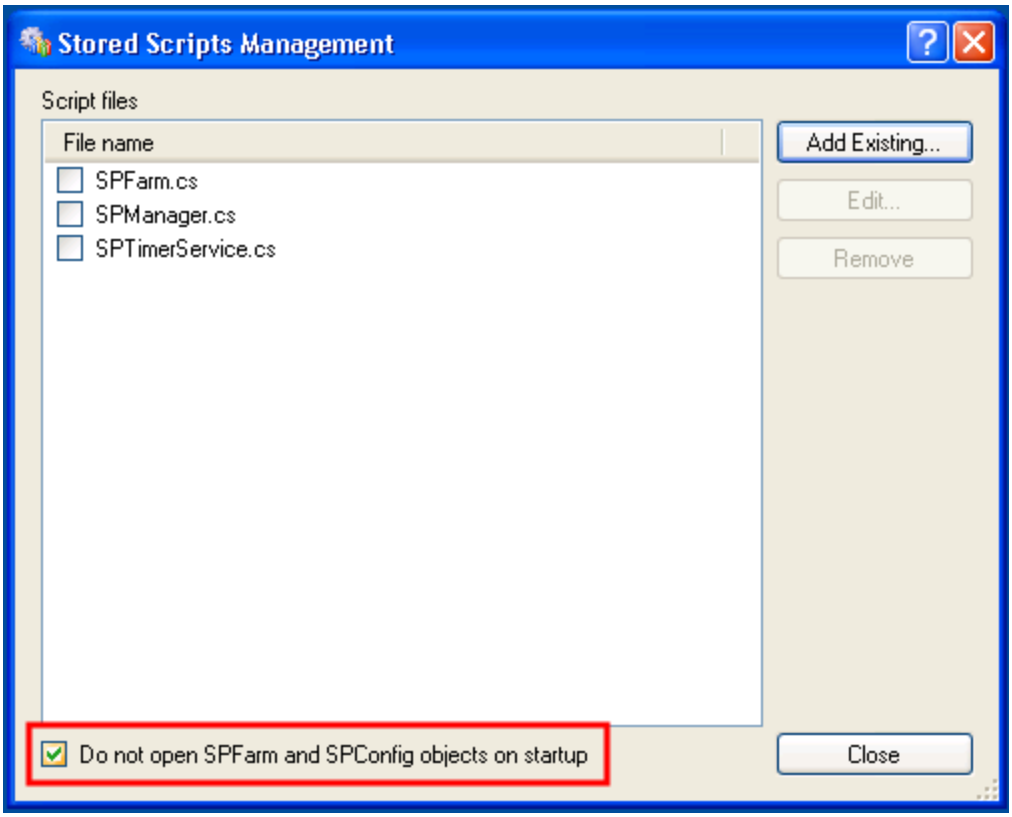
To create a script to load frequently used objects (**SPFarm**, **SPManager**, **SPTimerService**, **SPWebApplication**, **SPSite** and **SPWeb**) you can click **Add to stored** from the context menu of the required object. Then the script for this object will automatically added to the stored scripts that allow its use in future work.



Default startup script

By default on startup HarePoint Explorer for SharePoint runs the script, as a result in a browser window appears objects **SPFarm** and **SPManager**. It is basically a "root" of the farm.

To disable this it is enough to put the flag next to **Do not open SPFarm and SPConfig objects on startup** in **Stored Scripts Management**.


Stored Scripts Management
Script files
File name
SPFarm.cs
SPManager.cs
SPTimerService.cs
Add Existing...
Edit...
Remove
Do not open SPFarm and SPConfig objects on startup
Close

Usage examples

Here we will review some examples of HarePoint Explorer for SharePoint usage.

The first example shows how to receive information about different SharePoint objects via Explorer: sites, site collections, web-applications, etc.

Names of the second and third examples say for themselves – these are examples of introducing some modifications in the structure of SharePoint.

The fourth and fifth examples demonstrate opportunities of "non-targeted" use of Explorer.

Examples

1. Where is the list of sites located in the object model?
2. Site creation
3. List creation
4. Example of work with Active Directory
5. Example of work with Windows Forms
6. Debugging of regular expressions
7. Exploration of W3C Document Object Model (DOM)

Where is the list of sites located in the object model?

Probably, the first question, which a developer can ask at the first launch of HarePoint Explorer for SharePoint, is going to be the following: "And where are the sites here?"

The answer is simple. In order to find the list of sites created in SharePoint, the following must be done:

1. Find and expand node that conforms to property **Services**.
2. Expand node **Collection Items** to get the list of services included in SharePoint farm.
3. In the list of services find service, the name of which starts with **SPWebService Parent=SPFarm Name=SharePoint_Config_**.
4. Expand node of property **WebApplications** containing the list of web-applications.
5. In node Collection Items find object with the name **SPWebApplication Name=SharePoint - 80 Parent=SPWebService**.
6. Expand node of property **Sites**. This node contains the list of site collections created in the context of previously selected web-application.
7. In node **Collection Items** select one of the collections.
8. And finally, by expanding **AllWebs, Collection Items** sequentially, we will see the necessary list of sites.

From the first sight, this process may appear as complicated and tiresome. But attention must be paid to the fact that even during realization of such simple task, we get a lot of information about the object model of SharePoint! Particularly, it becomes clear that besides web-services, there are many other useful services in SharePoint: diagnostic services, timers, administrative services, etc. It is obvious that there can be many web-applications, site collections and other items of infrastructure. All this information is irreplaceable for proper development of SharePoint solutions. Also it is important that all this information we get without write a single line of code, without any tests!

Site creation

In the previous example, we got to know how to find sites and site collections in the browser of SharePoint object model. Here we will find out how to supply collection with one more site.

First thing that needs to be done is to find and flag property **AllWebs** of a site collection (the way to do it was described in the previous example) and create new script window.

The program will automatically generate the script code:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
            Microsoft.SharePoint.SPWebCollection allWebs1
            ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
            ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
            )
    {
        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;

        browser.ReturnValue = null;
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal allWebs1 As Microsoft.SharePoint.SPWebCollection,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)
```

```
        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded

        browser.ReturnValue = Nothing
    End Sub
End Class
```

All that has to be done is to call one function **SPWebCollection.Add** and return the execution result for further investigation:

C#

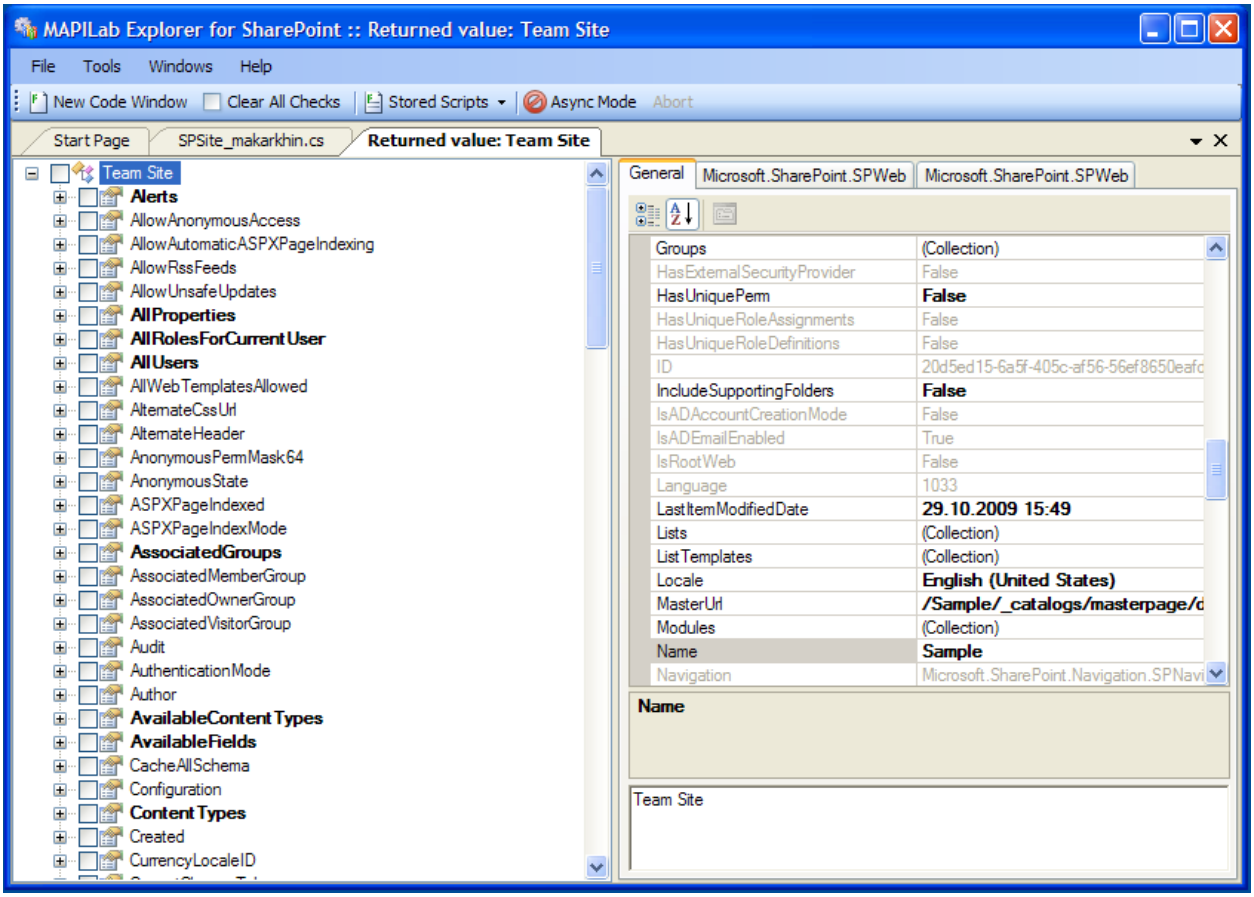
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
            Microsoft.SharePoint.SPWebCollection allWebs1
            ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
            ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
            )
    {
        allWebs1.Add("Sample");

        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;

        browser.ReturnValue = allWebs1["Sample"];
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner
```

```
Public Class Tester
    Shared Sub Main(ByVal allWebs1 As Microsoft.SharePoint.SPWebCollection,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        allWebs1.Add("Sample")

        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded

        browser.ReturnValue = allWebs1("Sample")
    End Sub
End Class
```

After executing the script, information about newly created site will be shown in the new browser window. Do not close this window – you will need it in the next example!

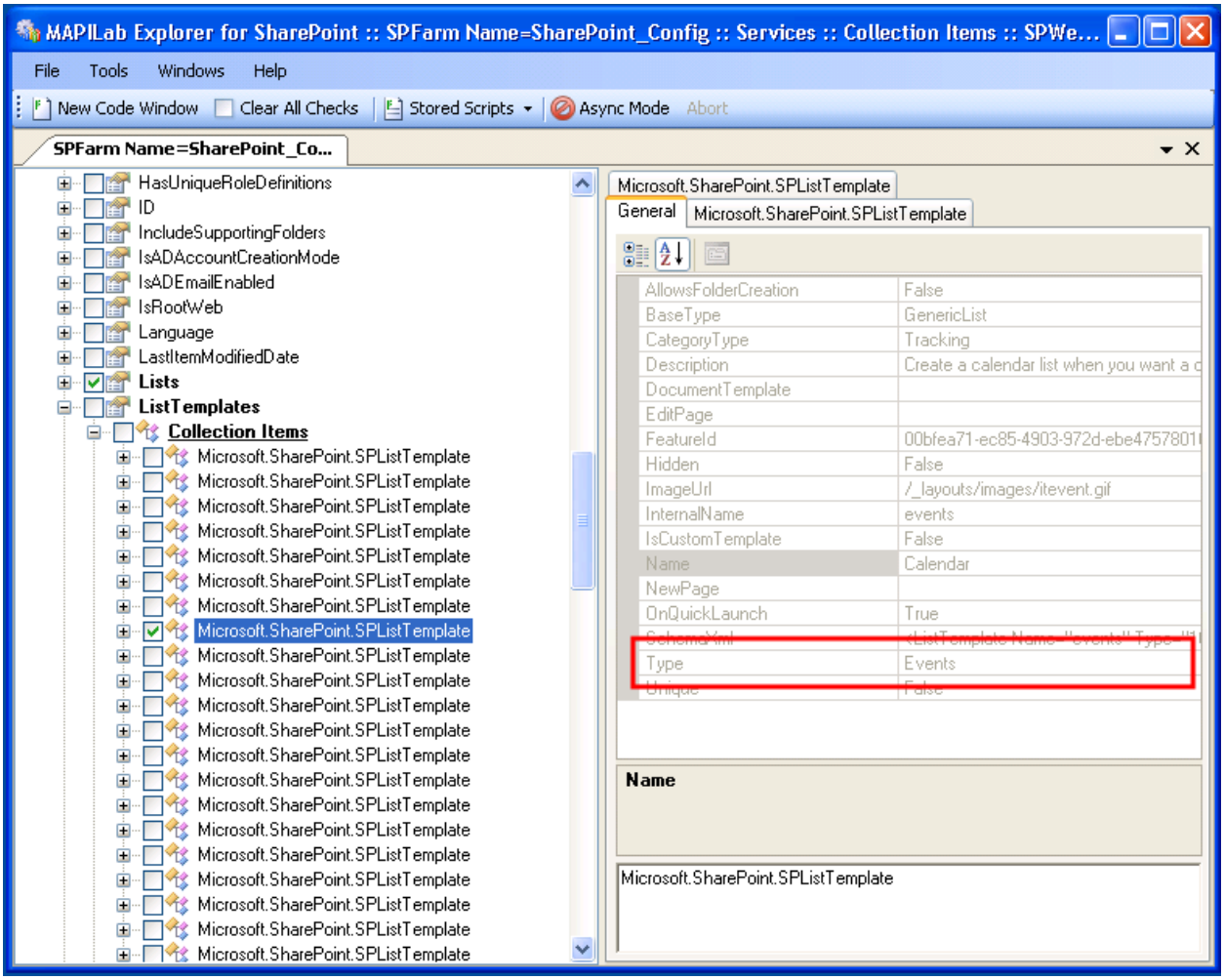


List creation

In the previous example, we created new empty site. Let us add a new list in it.

Since signature of method **Main** is generated considering objects selected in all browser windows, clear all set flags before proceeding by clicking **Clear All Checks** on the toolbar.

In order to create list, we need site collection and list template. References to these objects can be obtained differently, but we will use the most simple way – just find them in the tree and flag them: list collection is located in property **Lists**, list template can be found and flagged by searching across the collection **ListTemplates** (to be specific, select template with property **Type** = **Events**).



After opening new script window, we will see the following programming code:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
```

```
    static void Main(
            Microsoft.SharePoint.SPListCollection lists1
            ,Microsoft.SharePoint.SPListTemplate microsoftSharePointS2
            ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
            ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
            )
    {
        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;

        browser.ReturnValue = null;
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal lists1 As Microsoft.SharePoint.SPListCollection,
                    ByVal microsoftSharePointS2 As
Microsoft.SharePoint.SPListTemplate,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded

        browser.ReturnValue = Nothing
    End Sub
End Class
```

Let`s modify this code so that new list is created at execution. Reference to newly created list will be returned by method main for further investigation.

C#

```
using System;
```

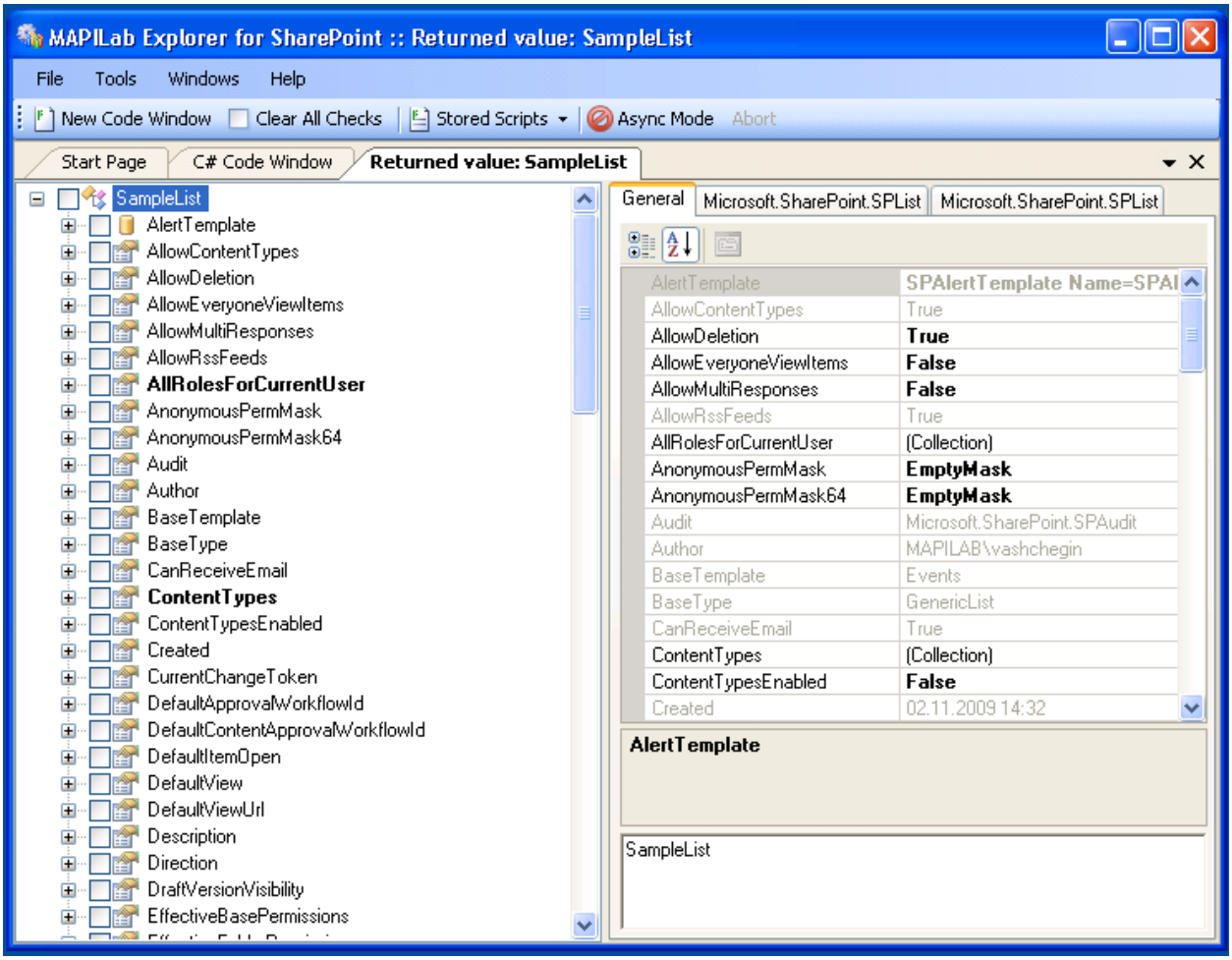
```
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;

public class Tester
{
    static void Main(
            Microsoft.SharePoint.SPListCollection lists1
            ,Microsoft.SharePoint.SPListTemplate microsoftSharePointS2
            ,MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
            ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
            )
    {
        System.Guid listId = lists1.Add("SampleList", "Sample description",
microsoftSharePointS2);

        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;

        browser.ReturnValue = lists1[listId];
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner

Public Class Tester
    Shared Sub Main(ByVal lists1 As Microsoft.SharePoint.SPListCollection,
                    ByVal microsoftSharePointS2 As
Microsoft.SharePoint.SPListTemplate,
                    ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        Dim listId As System.Guid
        listId = lists1.Add("SampleList", "Sample description",
microsoftSharePointS2)

        'Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded
```

```
        browser.ReturnValue =lists1(listId)
    End Sub
End Class
```

New browser window with information about created list will be opened as a result of executing this script:



Example of work with Active Directory

As mentioned in the introduction, HarePoint Explorer for SharePoint can be used for operations not only with SharePoint object model, but also with any other object model built on basis of .NET platform.

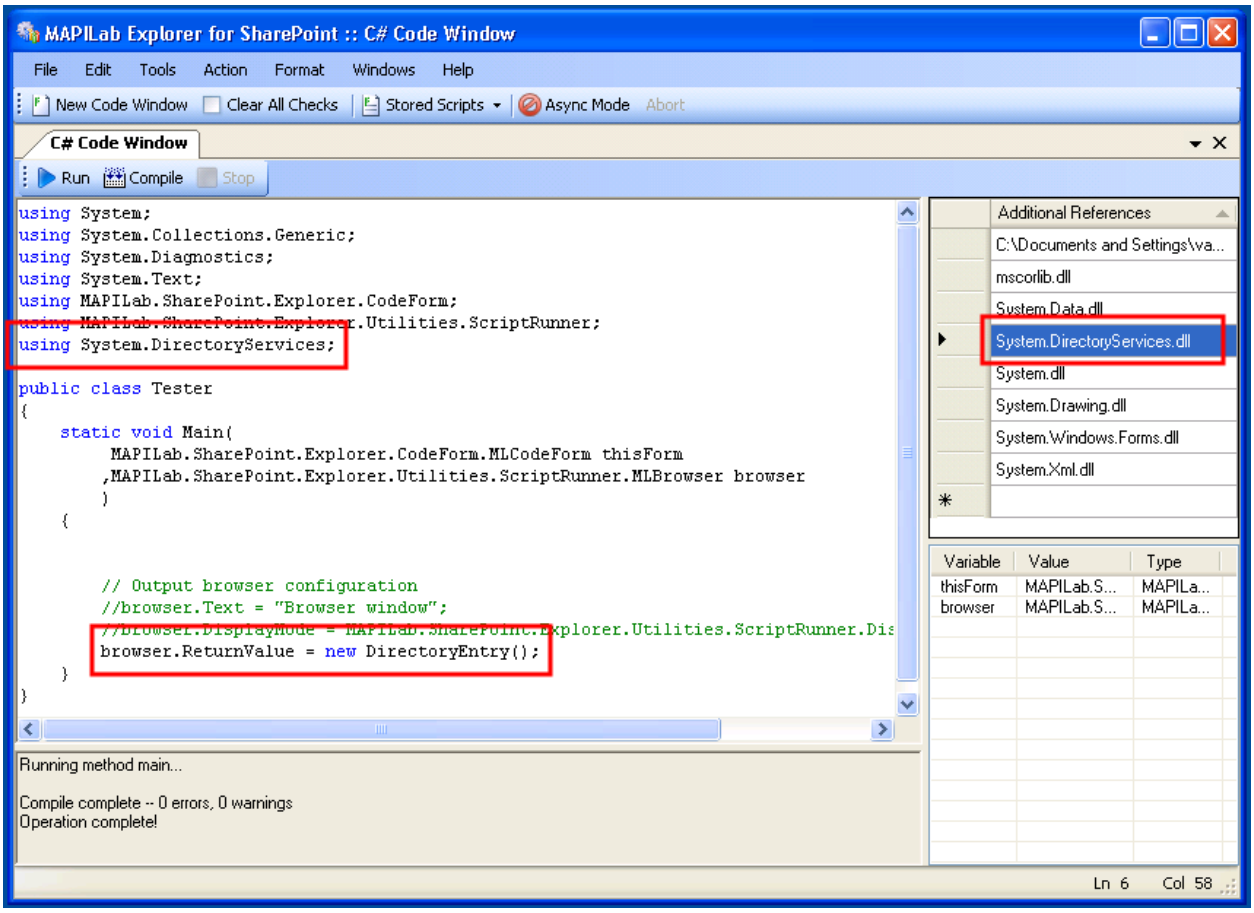
As an example, let us review the object model that provides access to Active Directory objects.

Each node of Active Directory in .NET model is presented via object of type **DirectoryEntry**. To access the root node, we can use constructor **DirectoryEntry** without parameters.

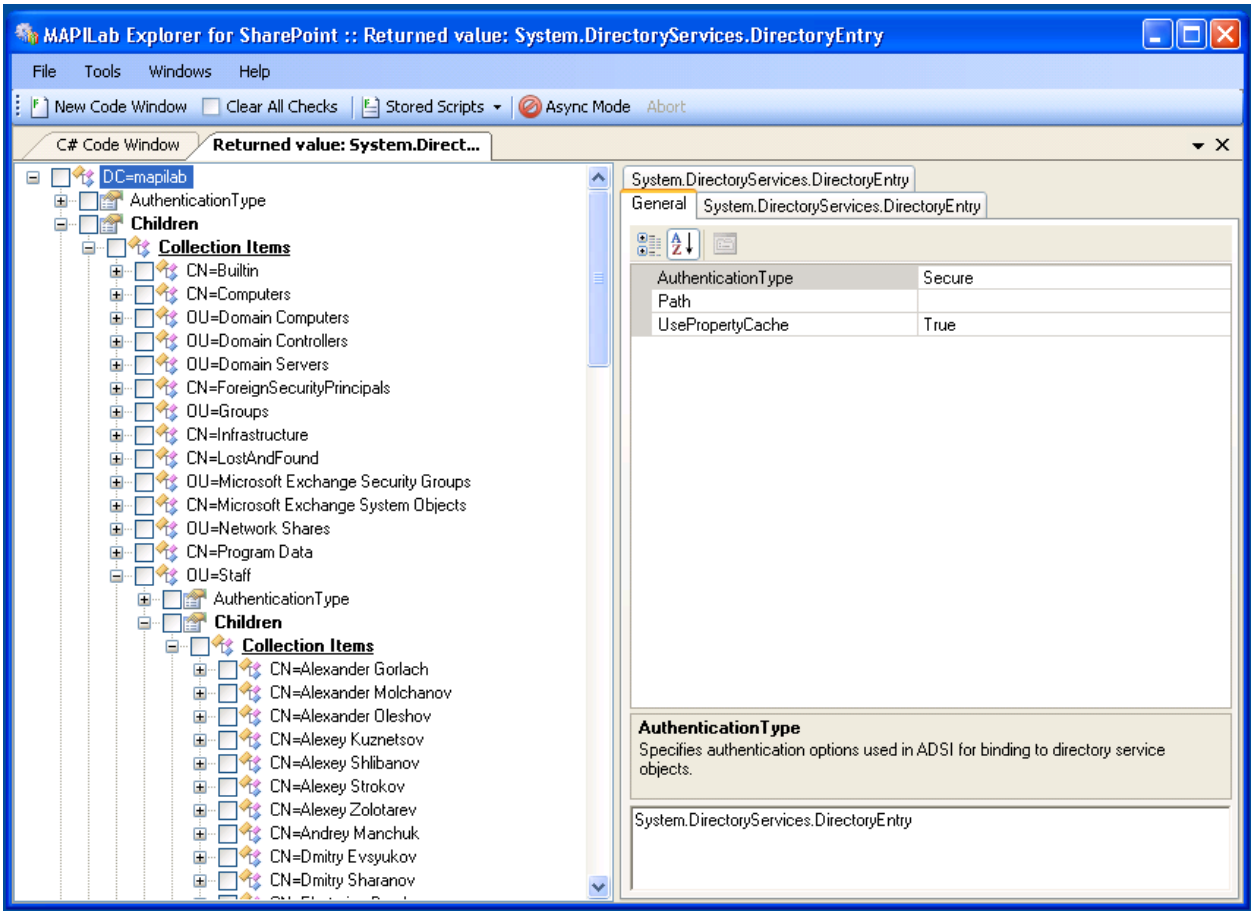
The following must be done for that:

1. Create new script window.

2. In the upper right window part, add reference to assembly **System.DirectoryServices.dll** to the list of additional references.
3. Create object **DirectoryEntry** with the help of operator **new** and return reference to created object.



As a result of script execution, new browser window will be opened. Using this window the whole hierarchy of Active Directory can be researched:



Example of work with Windows Forms

As another example of universality of HarePoint Explorer for SharePoint, let`s review the operations with class **System.Windows.Forms.Form**.

Open new script window and modify text of method **Main** the following way:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;
using System.Windows.Forms;

public class Tester
{
    static void Main(
         MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {
```

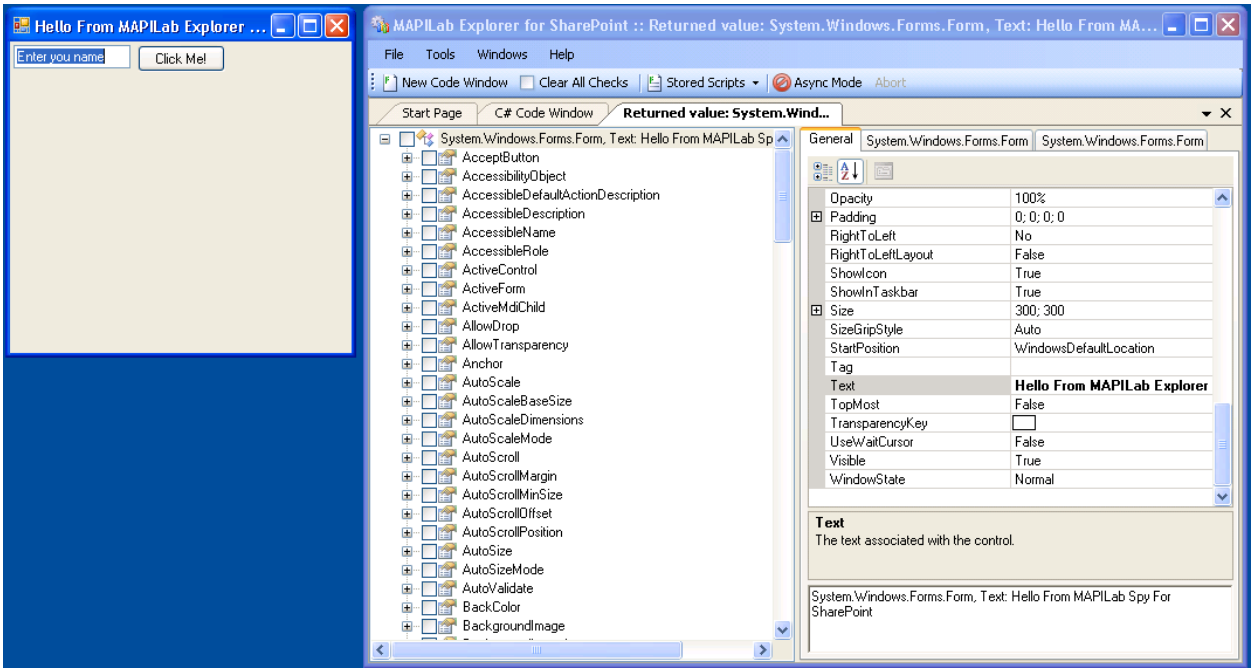
```
        Form form = new Form();
        form.Text = "Hello From MAPILab Explorer For SharePoint";
        FlowLayoutPanel layoutPanel = new FlowLayoutPanel();
        form.Controls.Add(layoutPanel);
        layoutPanel.Dock = DockStyle.Fill;
        TextBox textBox = new TextBox();
        textBox.Text = "Enter you name";
        layoutPanel.Controls.Add(textBox);
        Button button = new Button();
        button.Text = "Click Me!";
        layoutPanel.Controls.Add(button);
        form.Show();

        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;
        browser.ReturnValue = form;
    }
}
```

Visual Basic

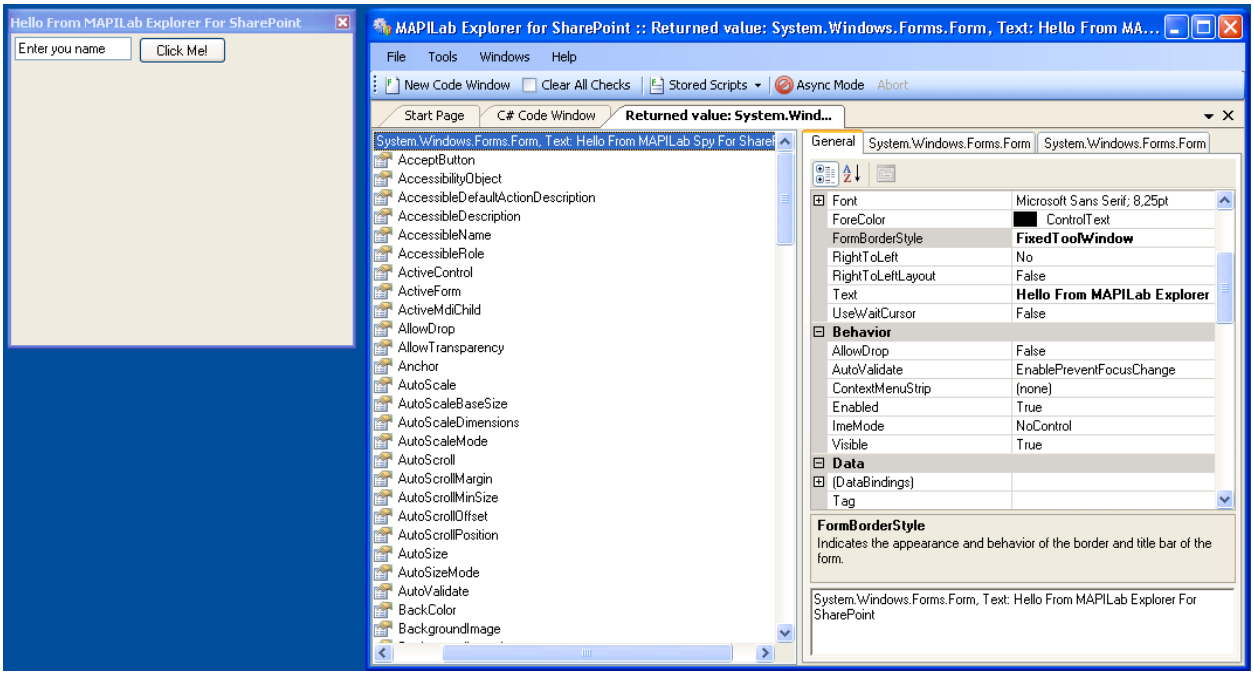
```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner
Imports System.Windows.Forms

Public Class Tester
    Shared Sub Main(ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        Dim form As Form = new Form()
        form.Text = "Hello From MAPILab Explorer For SharePoint"
        Dim layoutPanel As FlowLayoutPanel = new FlowLayoutPanel()
        form.Controls.Add(layoutPanel)
        layoutPanel.Dock = DockStyle.Fill
        Dim textBox As TextBox = new TextBox()
        textBox.Text = "Enter you name"
        layoutPanel.Controls.Add(textBox)
        Dim button As Button = new Button()
        button.Text = "Click Me!"
        layoutPanel.Controls.Add(button)
        form.Show()

        ' Output browser configuration
        'browser.Text = "Browser window"
```

```
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded

        browser.ReturnValue = form
    End Sub
End Class
```

The given program code creates new window, adds several controls in it, after that it is displayed using method **Show()** and returns reference to window object for further investigation. The result of script execution looks the following way:



Here, pay attention that properties of created window may be changed. By changing properties (without closing anything), our window can be made to look the following way:



Debugging of regular expressions

Let us consider an example of debugging of regular expression with the help of HarePoint Explorer for SharePoint.

As initial data, requests to FrontPage Server Extensions will be taken into consideration. These are the examples of basic requests sent from Microsoft Word to Microsoft Office SharePoint Server:

```
method=get
document:12.0.0.4518&service_name=&document_name=Lists/Links/Untitled_2.css&
old_theme_html=false&force=true&get_option=none&doc_version=&timeout=0&expan
dWebPartPages=true

method=put
document:6.0.2.6551&service_name=/test&document=[document_name=Shared
Documents
/MSF.doc;meta_info=[vti_timelastmodified;TW|09 Sep 2008 10:54:28 -0000]]&
put_option=edit&comment=&keep_checked_out=false

method=remove documents:12.0.0.4518&service_name=&url_list
[Lists/Links/Untitled_2.css;Lists/Links/Untitled_1.css;Lists/Links/Untitled_
3.css;Lists/Links/Untitled_4.css]
```

Our task is to create regular expressions for searching names of documents mentioned in the requests above as well as to analyze the results.

Let us open a new code window and change the text of the main method in the following way:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;
using System.Text.RegularExpressions;

public class Tester
{
    static void Main(
         MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
   {
        string str1 = @"method=get document:12.0.0.4518&service_name=
&document_name=Lists/Links/Untitled_2.css&old_theme_html=false&force=true
&get_option=none&doc_version=&timeout=0&expandWebPartPages=true";

        string str2 = @"method=put document:6.0.2.6551&service_name=
/test&document=[document_name=Shared Documents/MSF.doc;meta_info=
[vti_timelastmodified;TW|09 Sep 2008 10:54:28 -0000]]&put_option=
edit&comment=&keep_checked_out=false";

        string str3 = @"method=remove documents:12.0.0.4518&service_name=
&url_list=Lists/Links/Untitled_2.css;Lists/Links/Untitled_1.css;Lists/Links
/Untitled_3.css;Lists/Links/Untitled_4.css]";

        Regex regex1 = new
Regex(@"[&\[]document_name=(?<documentName>[^&;]+)");
        Regex regex2 = new
Regex(@"(url_list=\[|;)(?<documentName>[^;\]]+)");

        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;
        browser.ReturnValue = new object[] {regex1.Match(str1),
regex1.Match(str2), regex2.Matches(str3)};
        }
}
```

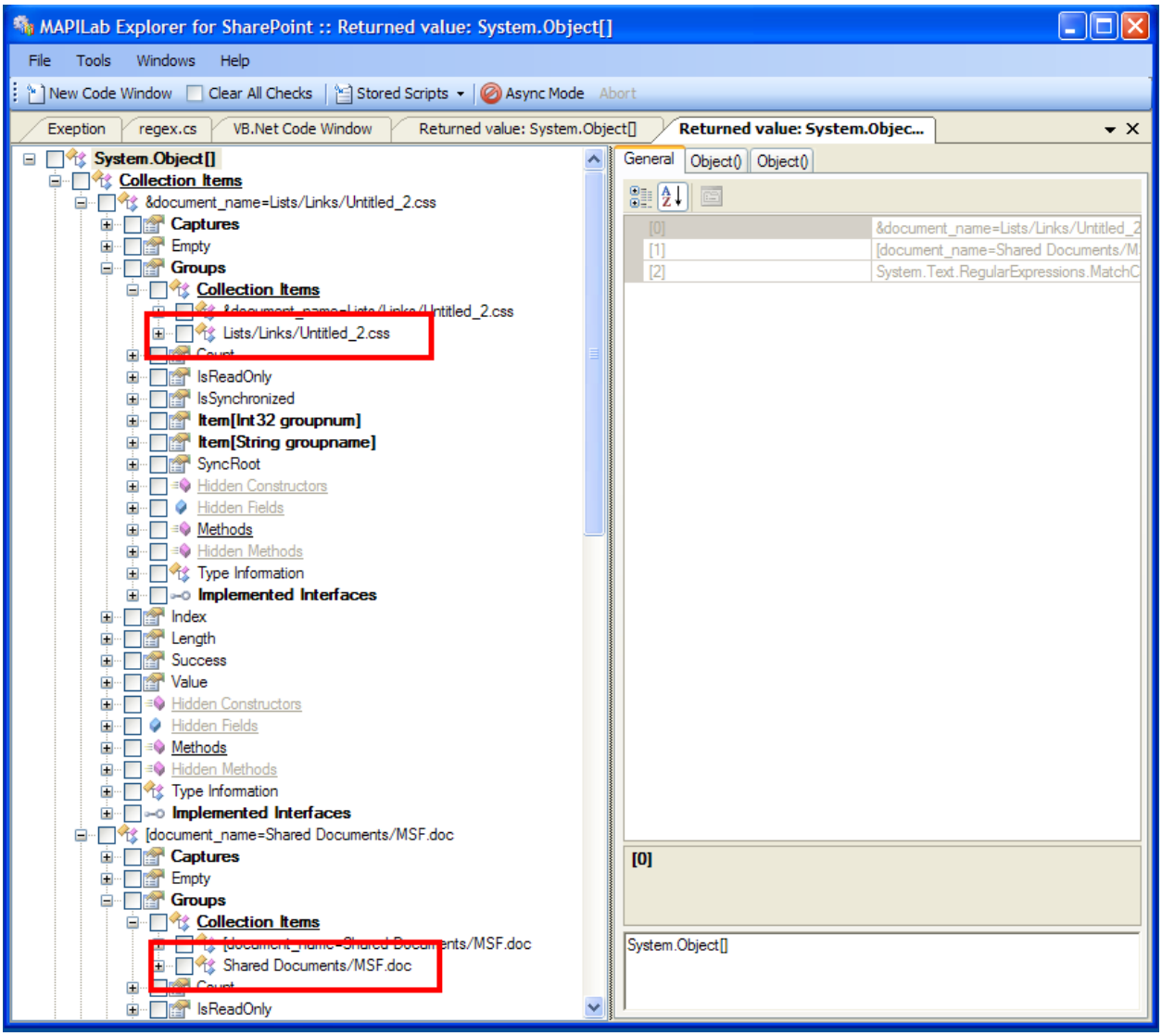
Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner
Imports System.Text.RegularExpressions
```

```
Public Class Tester
    Shared Sub Main(ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
                    ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)

        Dim str1 As String = "method=get document:12.0.0.4518&service_name=
&document_name=Lists/Links/Untitled_2.css&old_theme_html=false&force=true
&get_option=none&doc_version=&timeout=0&expandWebPartPages=true"
        Dim str2 As String = "method=put document:6.0.2.6551&service_name=
/test&document=[document_name=Shared Documents/MSF.doc;meta_info=
[vti timelastmodified;TW|09 Sep 2008 10:54:28 -0000]]&put option=
edit&comment=&keep_checked_out=false"
        Dim str3 As String = "method=remove
documents:12.0.0.4518&service_name=
&url_list=Lists/Links/Untitled_2.css;Lists/Links/Untitled_1.css;Lists/Links
/Untitled_3.css;Lists/Links/Untitled_4.css]"

        Dim regex1 As New
Regex("[&\[]document_name=(?<documentName>[^&;]+)")
        Dim regex2 As New Regex("(url_list=\[|;)(?<documentName>[^;\]]+)")

        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded
        browser.ReturnValue = New Object() {regex1.Match(str1),
regex1.Match(str2), regex2.Matches(str3)}
    End Sub
End Class
```

For the first two cases, one and the same regular expression regex1 can be used in this script. The third case is more difficult than the previous ones as the initial data contains several names of the document. A separate regular expression regex2 was specially created for it.

In order to check the correctness of functioning of these regular expressions let’s return their results in the form of object array. This object array will be shown in the browser of the object model.



If we open the nodes as it shown in the picture, we can see that all regular expressions were created correctly. Now you need only to copy them into the source code of the commercial product as we did it while we were developing HarePoint Analytics for SharePoint!

Exploration of W3C Document Object Model (DOM)

Let us consider an example of implementation of HarePoint Explorer for SharePoint.

Web service called **WebPartPages.asmx**, which is included into **Windows SharePoint Services**, gets requests from client applications in form of xlm-packages. This is an example of such a request:

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

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetWebPartPage xmlns="http://microsoft.com/sharepoint/webpartpages">
```

```
      <documentName>Lists/Links/EditForm.aspx</documentName>
    </GetWebPartPage>
  </soap:Body>
</soap:Envelope>
```

Our task is to consider the contents of this package.

To solve this problem let's use the **System.Xml.XmlDocument** class. Let's put the above mentioned example in any file, for instance, **C:\test.xml**. Then let's open a new code window and change the text of the 'main' method in the following way:

C#

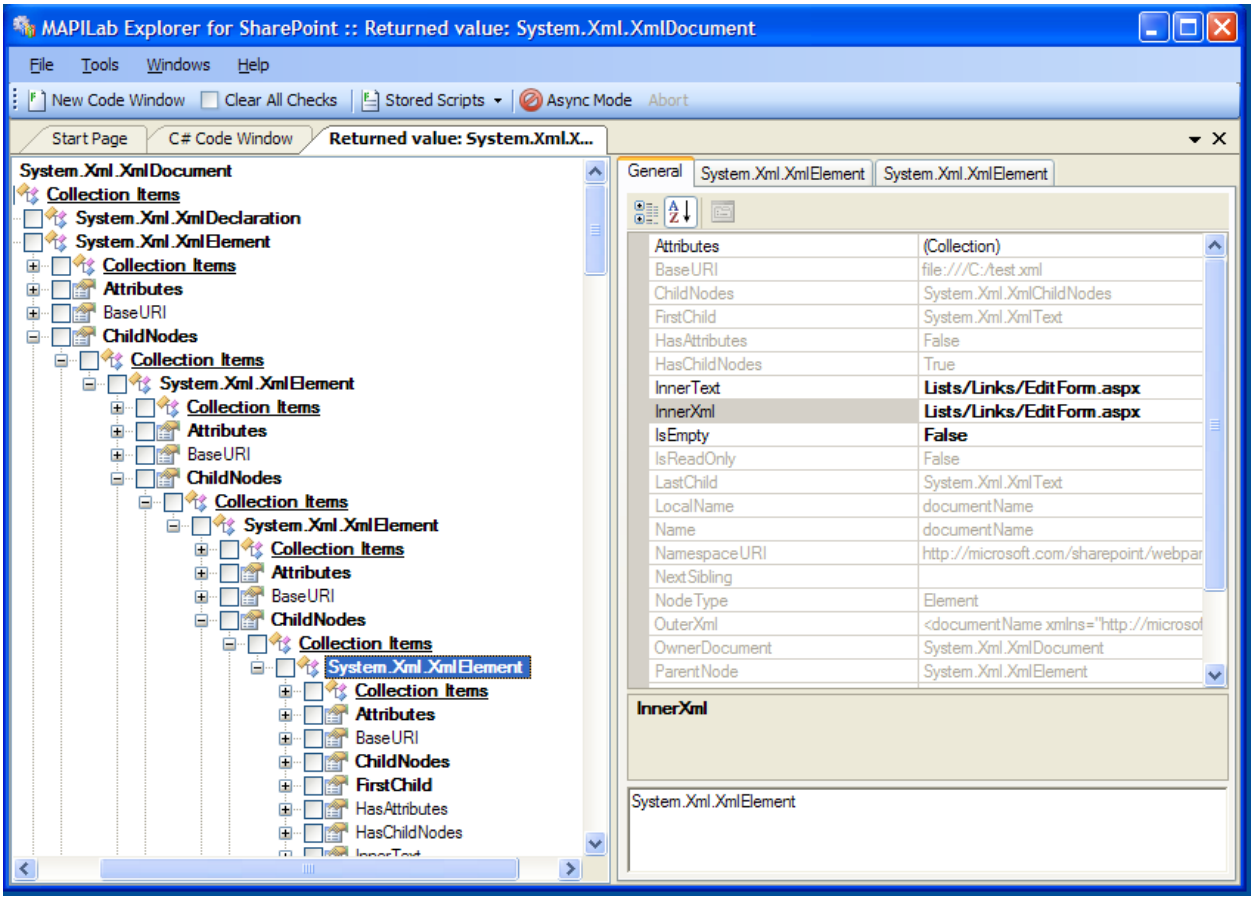
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using MAPILab.SharePoint.Explorer.CodeForm;
using MAPILab.SharePoint.Explorer.Utilities.ScriptRunner;
using System.Xml;

public class Tester
{
    static void Main(
         MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm thisForm
        ,MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser
browser
        )
    {
        XmlDocument document = new XmlDocument();
        document.Load(@"C:\test.xml");

        // Output browser configuration
        //browser.Text = "Browser window";
        //browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded;
        browser.ReturnValue = document;
    }
}
```

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.Text
Imports MAPILab.SharePoint.Explorer.CodeForm
Imports MAPILab.SharePoint.Explorer.Utilities.ScriptRunner
Imports System.Xml
```

```
Public Class Tester
    Shared Sub Main(ByVal thisForm As
MAPILab.SharePoint.Explorer.CodeForm.MLCodeForm,
ByVal browser As
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.MLBrowser)
        Dim document As XmlDocument = new XmlDocument()
        document.Load("C:\test.xml")

        ' Output browser configuration
        'browser.Text = "Browser window"
        'browser.DisplayMode =
MAPILab.SharePoint.Explorer.Utilities.ScriptRunner.DisplayMode.Expanded
        browser.ReturnValue = document
    End Sub
End Class
```

This source code loads the contents of the xml-file and then returns created **XmlDocument** for further examination with the help of the browser of the object model.



Now we can visually examine our xml in the form of the DOM-model. The node, which is marked out in the picture, includes the name of the document. It is called from the Microsoft SharePoint Designer.