

Tutorial: Working with SharePoint WebServices

version: 0.1

Author: David Haasler

Table of contents

1	Overview.....	2
2	Extending extension.....	2
2.1	Adding web link.....	2
2.2	Extending code.....	4
2.3	Error sources.....	6
2.3.1	WebService unavailable.....	6
2.3.2	Login to web server is not working.....	6
3	Extending transformations.....	6
3.1	Error sources.....	7
3.1.1	List addressing.....	7
4	Relocating authentication.....	8
4.1	Adapting web.config.....	8
4.2	Extending module.....	9
4.3	Extending extension.....	9

Requirements

The developer needs full web server access for this tutorial. In addition, there must be administrator access to the SharePoint server and to the SharePoint installation.

Since changes must also be made in SharePoint for using the SharePoint WebServices, the developer must know how to handle the software.

Moreover, the developer should have already familiarised himself with the onion.net SharePoint Integration.

For doing the basics, it is assumed that an own module or extension is already integrated in the project at hand. Moreover, it is assumed that the SharePoint Integration has been correctly installed and works.

This tutorial does not give a general introduction to SharePoint, SharePoint WebServices, the onion.net SharePoint Integration or the extension development.

- > [Developing own modules and extensions](#)
- > [Overview of the existing SharePoint Web Services](#)
- > [Microsoft SharePoint howto page](#)

Description

This tutorial describes the use of SharePoint WebServices using the example of the method “UpdateListItems”. List elements in a SharePoint list can be deleted or updated with this method. In addition, adding new elements is possible.

In this tutorial, values from the transformations are stored as list elements of a SharePoint list using the WebServices.

Signs and symbols



Boxes marked with an arrow symbol and a green border contains instruction of what to do next.



This kind of boxes contains tips and tricks.

Source code is shown in blue boxes.

1 Overview

The later procedure will be to call an extension method in the transformations. The XML is transferred at that stage, which is needed by the WebService later on. The extension method is only the mediator so to speak between the transformations and WebService. It makes the connection to the WebService and calls the method with the relevant parameters. Possible errors are then logged and an appropriate response in the form of the text “success” or “error” is returned, which can then be treated accordingly in the transformations.

The WebService merely throws an exception of the type “Microsoft.SharePoint.SoapServer.SoapServerException” in the case of an error. This does not tell us very much. In some cases an exception is not thrown, but merely an error code referred to in the return XML of the WebService. Therefore each part of this tutorial is followed by a section “error sources”, which addresses the most frequent problems and solutions. These sections are not exhaustive and are constantly extended where appropriate.

2 Extending extension

Since, for the extension of the transformations, we need to know exactly what the WebService method expects, it makes good sense to first create the extension method and familiarise ourselves a little with the WebService.

2.1 Adding web link

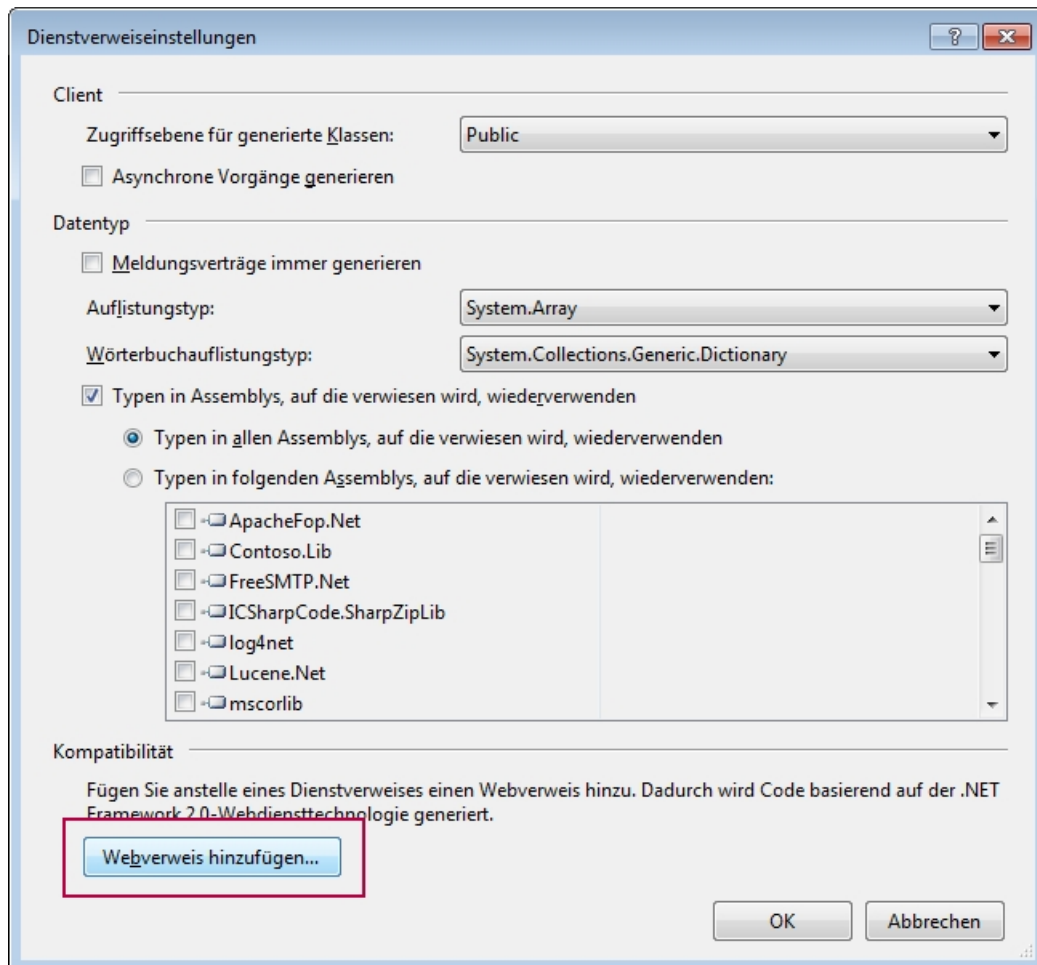
In order to use the methods of a WebService in the Visual Studio, the WebService must be added as a link.



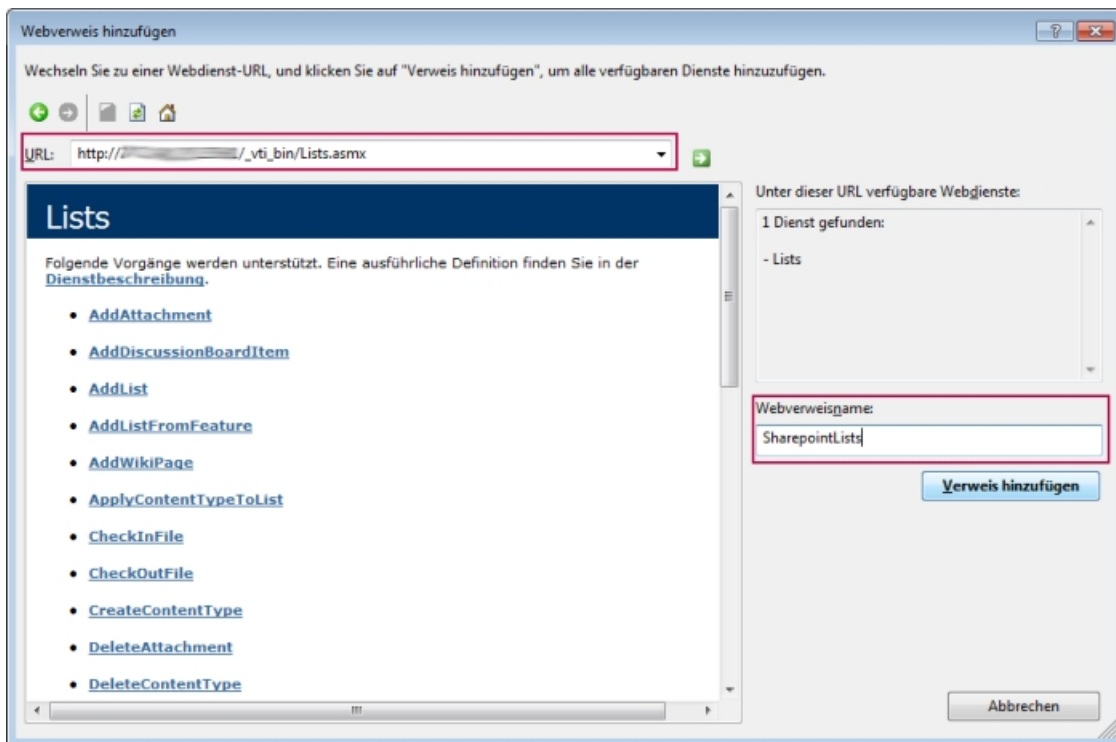
Right-click on the project and select “Add web link”.



If you do not have this item in the context menu, select “Add service link” instead. Click on “Extended...” in the dialogue that opens and select “Add web link”.

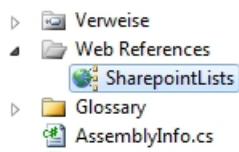


Enter the URL to the WebService in the opened dialogue (in our case “http://<Site>/_vti_bin/Lists.asmx”) and assign a name, e.g. “SharepointLists”.



If you click on the green arrow, the available methods are listed.

After you have clicked on “Add reference”, the Webservice in the Project Folder Explorer should look as follows



2.2 Extending code

In order to use the methods of the WebService, an appropriate instance must be present first of all.



To start with, insert the class of the WebService that has just been inserted for use with the using directive.

```
using Contoso.Renderengine.SharepointLists;
```



We can then create a global object on it:

```
private Lists lists = new Lists();
```

Since SharePoint is linked with the active directory of the domain, a login is usually necessary.



Extend the constructor of the extension to include the following code:

```
lists.Credentials = new NetworkCredential("username", "password"); lists.Url  
= "http://www.my-sharepoint.de/_vti_bin/lists.asmx";
```



If SharePoint runs on a server other than the Render Engine and the login data is assigned to another domain, this must be transferred as a third parameter when creating the NetworkCredentials. Note: the protocol must always be placed in front (even if an IP address is specified instead of a readable URL).

We can now add an extension method, which is considered the desired mediator.



To do this, add the following code.

```
[ExtensionMethod(Name = "updateListItems")]  
public string UpdateListItems(string listID, XPathNavigator batchXml)  
{  
    try  
    {  
        XmlDocument batch = new XmlDocument();  
        batch.LoadXml(batchXml.InnerXml);  
        XmlNode updates = batch.SelectSingleNode("Batch");  
        lists.UpdateListItems(listID, updates);  
        return "success";  
    }  
    catch (Exception e)  
    {  
        Log.Error(e.Message);  
    }  
}
```

```

    return "error";
}
}

```

We want to use the WebService method “UpdateListItems ()”. This is documented in the [Microsoft Library](#). The method can be universally applied for the three use cases Delete, Update and Add. What exactly is to be done is controlled via the “Batch” XML, which is transferred as a parameter. We will look at this in the next section, since we want to create it in the transformations.



Then build a new “Build” and update the onion.net Render Engine.

2.3 Error sources

2.3.1 WebService unavailable

Call the URL of the WebService locally in the browser. If you are given a list of the methods of the WebService(after a login where appropriate), you know that the WebService is available.

If you do not get a list of the methods although the login and URL are correct, a setting may have to be readjusted in SharePoint or a port opened on the server in the Firewall for the WebServices.

2.3.2 Login to web server is not working

Check whether you have specified the correct domain for creating the NetworkCredentials and whether the protocol (e.g. “HTTP”) is in front. This also has to be indicated if you do not specify a readable web address but just an IP address.

3 Extending transformations

As previously mentioned, the WebService method “UpdateListItems ()” requires an XML element and the GUID of the SharePoint list to be accessed.



Note: The XLink is not meant by this, but really only the **GUID!**

X L i n k :

sharepoint/!(en,064525c1-007a-47a6-b349-32990e2416ad)@189b38e1-5b2a-49c6-aefc-b601a26c0bee

GUID: 064525c1-007a-47a6-b349-32990e2416ad

The possible structure of the batch XML is documented in detail in the Microsoft Library.

In the case of a product evaluation it could look for example as follows:


```
<Batch OnError="Continue">
  <Method ID="1" Cmd="New">
    <Field Name="ID">New</Field>
    <Field Name="Title">2012-04-05 13:42 Erika Mustermann</Field>
    <Field Name="FullName">Erika Mustermann</Field>
    <Field Name="Bewertung">5</Field>
    <Field Name="Bewertungstext">Super Produkt! Jederzeit wieder! Nur zu
empfehlen!</Field>
  </Method>
</Batch>
```

The fields “FullName”, “Evaluation” and “Evaluation text” are self-added website columns to the data type of the list element. The SharePoint-internal name must be indicated here as the attribute “Name”. This can be determined in SharePoint itself. An alternative is the onion.net data view of SharePoint contents.



If you want to fill a website column of the “Lookup” type using the batch XML, you must structure this as follows.

```
<Field Name="ProductReference" List="{List-GUID der Referenzliste}">{List-ID
of the entry in the reference list}</Field>
```

A concrete example:

```
<Field Name="ProductReference"
List="064525c1-007a-47a6-b349-32990e2416ad">10</Field>
```

The list ID of the item is the unique number of the list item which is to be linked to within the list where the item is located. This can be found in the XLink of a list item. For the XLink »sharepoint//list(en,064525c1-007a-47a6-b349-32990e2416ad,11)@189b38e1-5b2a-49a6-aefd-b601a26e0bee«, it can be determined that the item has the ID »11« within the list.



Manually create a list item in SharePoint and call the XML of the item in the browser with the data view “content”.

3.1 Error sources

3.1.1 List addressing

Make sure that you really do **only** indicate the list **GUID**, and **not the XLink**. (e.g. »sharepoint//list(en,064525c1-007a-47a6-b349-32990e2416ad)@189b38e1-5b2a-49a6-aefd-b601a26e0bee«).

4 Relocating authentication

The information for establishing the connection to the SharePoint can still be found directly in our extension at the moment. This makes little sense for various reasons.

On the one hand, it means the code cannot be reused. If, for example, a SharePoint other than live is to be used in the preview (so as to not introduce tests produced in the preview into the productive SharePoint), then this is cannot be done just like that. There would have to be different extensions or different extension methods for live and preview.

Moreover, a change to the access data (perhaps carried out in 3 month-periods for security reasons) or the relocation of the SharePoint system to another server with another IP address, would mean an unnecessary change in the extension. Instead of letting the server administrator perform its administrative activities, a web developer must make a change in the code and update the Render Engines. The latter usually entails a restarting of the web application, which may not be wanted at any time of the day during operation.

It is therefore in our interest to relocate the login information into the web.config file of the Render Engine.



No later than now should you create a special onion.net user without administrative access to the SharePoint, since access is now being opened to more or less “everyone” who has access to the web.config.

4.1 Adapting web.config

First of all, we want to insert the login information into the web.config. Similarly to the configuration of the ImageServer, we create an element for it below the module configuration.

Extend the configuration of the project module as follows.

```
<module type="{Projektname}.Renderengine.{Projektname}Module,
{Projektname}.Renderengine">
  <sharepoint
    uri="http://www.my-sharepoint.de"
    username="username"
    password="password"
  />
</module>
```

If necessary, another attribute “domain” can be appended with the appropriate value.

4.2 Extending module

The configuration can be accessed in the module. An XML element “configuration” is transferred to the method “Configure ()” as a second parameter. This contains all the XML specified in the configuration within the <module> element. So in our case the <sharepoint> element.

We now want to read out the attributes of our new element <sharepoint> and transfer them to the extension in order to use them there for establishing a connection.



Extend the method “Configure()” of the class “{project name}Module” to include the following code.

```
XmlElement sp = configuration["sharepoint"];
if (sp == null) throw new ConfigurationException("Element 'sharepoint' ist missing",
configuration);
string uri = sp.GetAttribute("uri");
string username = sp.GetAttribute("username");
string pw = sp.GetAttribute("password");
string domain = sp.GetAttribute("domain");
if (String.IsNullOrEmpty(uri) || String.IsNullOrEmpty(username) || String.IsNullOrEmpty(pw))
    throw new ConfigurationException("Sharepoint configuration is not correct.", configuration);
extensions = new ContosoExtension(uri, username, pw, domain);
```

To start with, we specifically access the <sharepoint> element. If it is not there, an appropriate exception is thrown.

The relevant attributes are then read out and saved in variables. Here again, the necessary values are checked first and an exception thrown if necessary.

Last but not least, the values are also transferred when creating the extension.



Instead of the exceptions, you can simply log an appropriate error if the website is to also function without a correct SharePoint connection.

4.3 Extending extension

The transferred parameters must now be accepted in the constructor of the extension.



Extend/change the constructor of the extension as follows.

```
public ContosoExtension(string uri, string username, string password, string domain)
{
    lists.Credentials = !String.IsNullOrEmpty(domain) ? new NetworkCredential(username,
password, domain) : new NetworkCredential(username, password);
    lists.Url = uri + @"/_vti_bin/lists.asmx";
}
```

Depending on whether a domain is transferred, the NetworkCredentials are created with or without this domain. The “uri” is used to define the correct URL for accessing the WebService.