Sitecore CMS 6.5 or later

The Sitecore Web Service Reference Guide

A Developer's Guide to the Sitecore Service Layer

Contents

Sitecore Web Services and SOA

To realize the Service Oriented Architecture (SOA) paradigm, the Sitecore service layer is based on web services.

The Sitecore web service contains 21 methods.

You can also build your own service layer, extend the readymade service functionality, provide more security, and make it open to other technologies.

As is the case with any web service, the Sitecore web service is designed to:

- Expose and describe itself.

  A web service defines its functionality and attributes so that other applications can understand it. To make its functionality available to other applications, a web service contains a Web Service Description Language (WSDL) file.

- Enable other parties to locate it on the web.

  A web service can be registered in a Universal Description, Discovery, and Integration (UDDI) directory so that applications can locate it.

- Be invoked.

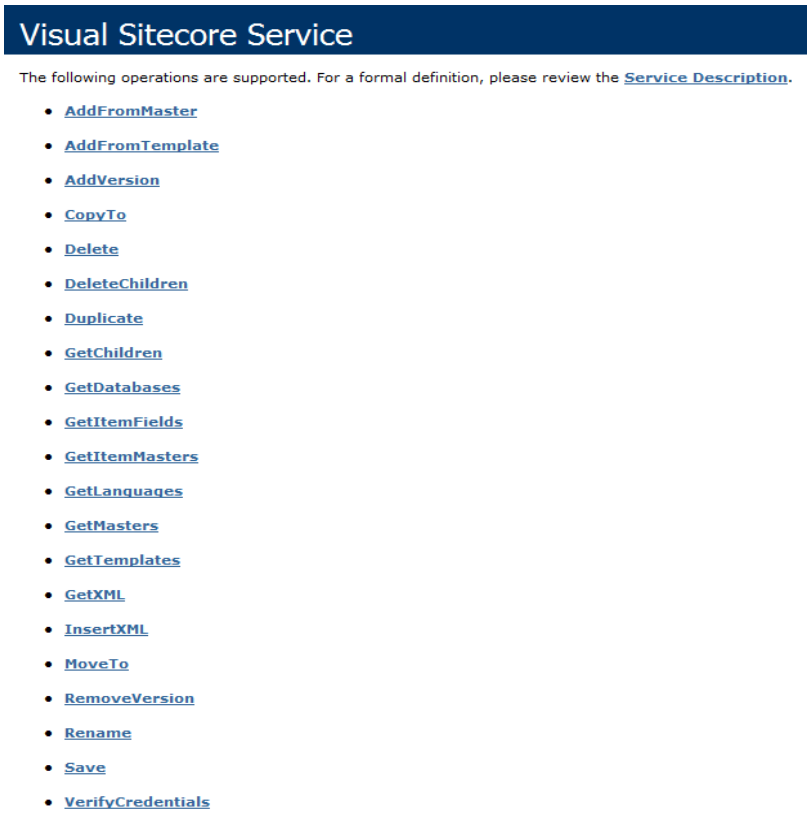
  Once a web service has been located and examined, a remote application can invoke the service using a standard Internet protocol.

  Offer appropriate internal business processes as value-added services that can be used by other organizations.

  Integrate its internal business processes and dynamically link them with those of its business partners.

Discovering the Web Service

To explore the standard Sitecore web service that is present in any Sitecore release, type `http://yourhost/sitecore/shell/webservice/service.asmx` and you see a list of the methods that the web service offers. The following image is an example of a Sitecore web service:



If you click on any of the methods, you can see that these operations are generic enough to provide a flexible interface that is open to SOAP 1.1 and 1.2 messages. This interface flexibility eases the service oriented integration.

Moreover, to access the WSDL file of the web service in your browser, click the *Service Description* link or type `http://yourhost/sitecore/shell/webservice/service.asmx?WSDL`

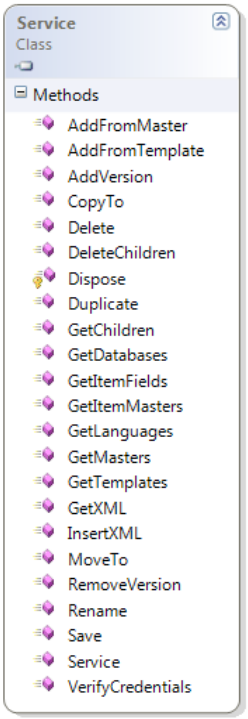
In the WSDL file, you can see:

- The service port that defines the address or connection point to the web service. The service port is represented by a simple HTTP URL string.
- The service target namespace.
- The operations that define the way the message is encoded, for example, *literal*.

API Reference Guide

The Sitecore web service API is in the `Sitecore.Visual.Service` class in the `Sitecore.Client.dll`.

The following class diagram describes the list of methods that are included in the Sitecore web service:



The following table describes each method in the Sitecore web service and its parameters:

Method	Description

`XElement AddFromMaster(`
`  string id,`
`  string masterID,`
`  string name,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the GUID of the parent item under which the new item should be created, the master ID that specifies the ID of the branch item, the `name` parameter that specifies the name of the new item, the database name and creates a new item based on a branch.

`XElement AddFromTemplate(`
`  string id,`
`  string templateID,`
`  string name,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the GUID of the parent item under which to create the new item, the template ID specifies the template, the name of the new item, and creates a new item based on that template.

Method	Description

`XElement AddVersion(`
`  string id,`
`  string language,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, the acronym of the language, the database name, the user's credentials, creates a new version of the item in this language, and includes the number of the new version in the SOAP response.

`XElement CopyTo(`
`  string id,`
`  string newParent,`
`  string name,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, the new parent item's GUID, the name of the item's copy, the user's credentials, and creates a copy of the item under the parent item in the content tree.

`XElement Delete(`
`  string id,`
`  bool recycle,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, a boolean value for `recycle`, the database name, the user's credentials, and deletes the item.
	If `recycle` is set to `false`, the item and its descendants are deleted permanently.
	If recycle is set to `true`, the item and its descendants are moved to the Recycle Bin in Sitecore.

`XElement DeleteChildren(`
`  string id,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the parent item's GUID, the database, the user's credentials, and deletes the children of this item.

`XElement Duplicate(`
`  string id,`
`  string name,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID and name, the database name, the user's credentials, and duplicates this item in the same location as the original item.

`XElement GetChildren(`
`  string id,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the GUID of an item, the database name, the user's credentials, and returns a list of the names of the item children and their GUIDs.

`XElement GetDatabases(` `  Credentials credentials)`	Takes the user credentials and returns a list of the database names that the Sitecore instance uses.

`XElement GetItemFields(`
`  string id,`
`  string language,`
`  string version,`
`  bool allFields,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, the acronym of the language that is set for the whole site, the version number, the boolean value for whether or not to return all fields, the database name, the user's credentials, and returns a list of the fields that this item contains.
	
	Note
	If `allfields` is set to `false`, the web service returns all the fields except for system fields. System fields belong to the Standard Template or one of the Standard Templates base templates such as *Appearance*, *Layout*, and *Workflow*.

Method	Description

`XElement GetItemMasters(`
`  string id,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, the database name, the user credentials, and returns a list of the template names and GUIDs that are configured in the insert options.

`XElement GetLanguages(`
`  string databaseName,`
`  Credentials credentials)`
	Takes the name of the database, the user credentials, and returns a list of the languages that are set for the entire site.
	
	Note
	If there is a language that is set for one or only some of the items, it is not included in the response.

`XElement GetMasters(`
`  string databaseName,`
`  Credentials credentials)`
	Takes the database name, the user's credentials, and returns the masters.
	
	Note
	The Master logic has been replaced by the Branch Templates logic.
	You can use Branch templates to insert items using structures that are predefined by Sitecore administrators. A branch template consists of a branch template definition item, including all of its descendants.
	When a user chooses a branch template from the insert options, Sitecore copies the entire collection of descendants defined under the branch template definition item, and places that collection as a group of subitems under the currently selected item.

`XElement GetTemplates(`
`  string databaseName,`
`  Credentials credentials)`
	Takes the database name, the user's credentials, and returns a list of the templates in the site.

`XElement GetXML(`
`  string id,`
`  bool deep,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, the database name, the user's credentials, and if `deep` = `true` the item's descendants. It then returns the XML representation of an item.
	
	Note
	For example, you can use the XML to back up or persist items. You can also use the `InsertXML` method to insert the items into the same or a different database.

Method	Description

`XElement InsertXML(`
`  string id,`
`  string xml,`
`  bool changeIDs,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, XML representation, the database name, `the changeIDs` parameter, and inserts an item, with or without descendants, into a database based on the representation.
	
	Note
	If you insert one or more items into the same database as they were copied from, the item version of the source item is overwritten if the `changeIDs` parameter is set to `false`. If the `changeIDs` parameter is set to `true`, the copied items are inserted as duplicates.

`XElement MoveTo(`
`  string id,`
`  string newParent,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, the new parent item's GUID and moves the item so that it becomes a child of the new parent in the content tree.

`XElement RemoveVersion(`
`  string id,`
`  string language,`
`  string version,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, the acronym of the language, the number of the version to delete, the database name, the user's credentials, and deletes the specified version of the item.

`XElement Rename(`
`  string id,`
`  string newName,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's GUID, the new name to give to the item, the database name, the user's credentials, and renames the item.

`XElement Save(`
`  string xml,`
`  string databaseName,`
`  Credentials credentials)`
	Takes the item's XML representation, the database name, and updates the item based on the representation
	
	Note
	You can obtain the XML representation using the `GetItemFields` method.

`XElement VerifyCredentials(` `  Credentials credentials)`	Takes the user's credentials and returns `<status>OK</status>` and `<data>OK</data>` if the user is authorized and returns `<status>failed</status>` and `<error>Unknown username or password.</error>` if the user is not authorized.

Note
The fully qualified name of the return type of the methods is `System.Xml.Linq.XElement` and the fully qualified name of the type of the credentials parameters is `SitecoreWebService.Credentials`.

API Extension

There are other operations you might need to accommodate in your service layer. These operations could be:

- Installing an update package.
- Installing a regular Sitecore Zip package.
- Getting the URL of an item by ID.
- Publishing items from source to target database.
- Transferring items from source to target.

To implement these operations, you can:

- Create a new web service that contains the new methods.

  Or

- Override the existing web service in the same way as the Sitecore Rocks web service.

Note
Because the web service class is not sealed and the methods are not virtual, we recommend that you create a new web service.

Sitecore Rocks Web Service

The Sitecore Rocks web service adds two more methods to the ordinary Sitecore web service:

Method	Description

`XElement Execute(`
`  string typeName,`
`  SitecoreWebService.`
`  ArrayOfAnyType parameters,`
`  Credentials credentials)`
	Adds remote procedure call (RPC) functionality to the web service.
	The RPC classes must be located in the `Sitecore.Rocks.Server.Requests` namespace and have a public method called `Execute` that returns a string.
	
	`public string Execute()`
	
	The method can take any number of string parameters. These parameters are passed to the `Execute` method. The `typeName` parameter denotes the class to invoke, for example `Sitecore.Rocks.Server.Request.GetChildren`.

`XElement Login(` `  Credentials credentials)`	Logs the user into the system using his credentials.

Note
If you have Sitecore Rocks plugged into Visual Studio, you automatically get the Sitecore Rocks web service when you connect to your website. It is called `service2.asmx` and placed in the same folder as the ordinary Sitecore web service.

Invoking the Web Service

Developers can invoke the web service by:

- Using the service as a reference.
- Creating a proxy for the web service.

Using the Service as a Reference

To invoke the web service by adding it as a reference, you must:

- Set up the items in the Sitecore content tree.
- Invoke the web service.

Setting Up the Items in the Sitecore Content Tree

Before you can invoke the web service, the items must be available in the content tree.

To make the items available in the content tree:

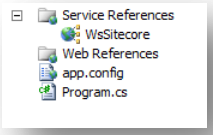
1. Log in to the Sitecore **Desktop** and click the **Sitecore** button, and then click **Content Editor**.
2. In the **Content Tree**, navigate to the `/templates/User Defined/` item and create a template called *Terminal Integration Point.*

   This template is used for integrating and importing items.
3. In the content tree, select the *Home* item
4. Click the **Home** tab, and then in the **Insert** group, select `/templates/User Defined/Terminal Integration Point` and name the new item *Terminal Integration Point and* Click **OK**
5. Navigate to `/sitecore/content/Home/Terminal Integration Point` and write down the item's GUID of *Terminal Integration Point* which is then used as the parent item under which to create the new items.
6. Navigate to `/sitecore/templates/Sample/Sample Item` and write down the item's GUID of *Sample Item* which is then used as the template to create the new items.

Invoking the Web Service

1. Create a Sitecore Visual Studio .NET console application project called **Terminal.Integration**
2. Set the **Terminal.Integration** project as the start-up project.
3. Add a service reference to the **Terminal.Integration** project.
4. In the **Add Service Reference** dialog.
5. Enter the URL of the Sitecore web service `http://yourhost/sitecore/shell/webservice/service.asmx`

6. Change the Namespace to `WsSitecore` and click **OK**.



You have now added a reference to the Sitecore web service.

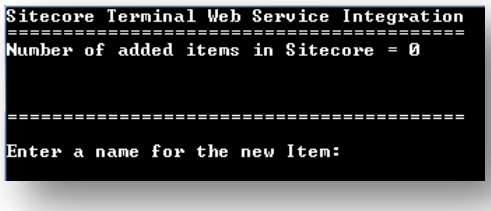
7. Open the `Program.cs` file and then in the main method:
   - Create a string with a reference to the item' GUID of the *Terminal Integration Point* instance.
   - Create a string with a reference to the template's GUID of the *Simple Item* template.
   - Create a string with a reference to the master database.
   - Instantiate a new proxy for the Visual Sitecore Service interface.
   - Create the web service credentials which must be used to invoke the web service.
   - Allow the console to read a name as input from the keyboard.
   - Use the name to create a new item in Sitecore using the `AddFromTemplate` method.
   - List all the items located under the *Terminal Integration Point* using the `GetChildren` method.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml.Linq;
using System.Xml;
using System.IO;

namespace Terminal.Integration
{
    class Program
    {
        static void Main(string[] args)
        {
            var visualSitecoreService =
                                        new WsSitecore.VisualSitecoreServiceSoapClient();
            string parentId = "{37F01BDF-AAF4-4C0C-87A0-CD26220F783D}";
            string templateId = "{1D7ADB59-3AB0-4131-93AB-AADE26297D65}";
            string database = "master";
            WsSitecore.Credentials credentials = new WsSitecore.Credentials();
            credentials.UserName = @"sitecore\admin";
            credentials.Password = "b";

            while (true)
            {
                Console.WriteLine("Sitecore Terminal Web Service Integration");
                Console.WriteLine("=========================================");
                var children = visualSitecoreService.GetChildren(parentId, database,
                credentials);
                using (XmlReader reader =
                           XmlReader.Create((new StringReader(children.ToString()))))
                {
                    XDocument loaded = XDocument.Load(reader);
                    var q = from c in loaded.Descendants("item")
                            select (string)c.Value;
                    Console.WriteLine("Number of added items in Sitecore = " +
                    q.Count().ToString() + "" + Environment.NewLine);
                    foreach (string name in q)
                    Console.WriteLine("Item name = {0}", name);
```

```
                        Console.WriteLine(Environment.NewLine);
                    }
                    Console.WriteLine("========================================" +
                    Environment.NewLine);
                    Console.WriteLine("Enter a name for the new Item:" +
                    Environment.NewLine);
                    string itemName = Console.ReadLine();
                    var result = visualSitecoreService.AddFromTemplate(parentId,
                    templateId, itemName, database, credentials);
                    Console.WriteLine("Item '" + itemName + "' is added.");
                    Console.WriteLine("Press key to continue");
                    Console.ReadKey();
                    Console.Clear();
                }
            }
        }
    }
```

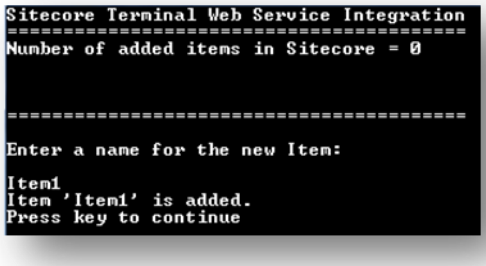
Note
You must replace the values of the `parentId` and `templateId` with the ones that you wrote down in the *Setting UP Items in the Sitecore Content Tree* section.

8. Compile the project and run the application.

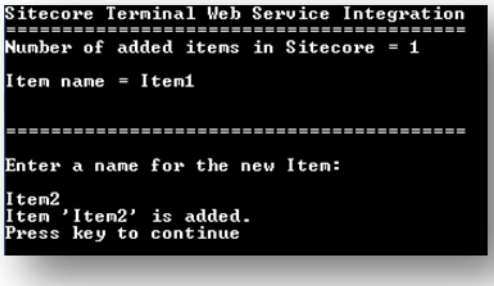


The `GetChildren` method shows that no items were created.

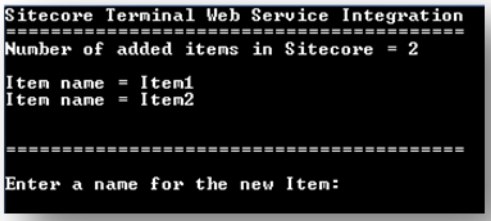
9. Create two new items: `Item1` and `Item2`.



Add *Item1.*

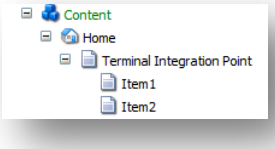


Add *item2.*



The `GetChildren` method shows that *Item1* and *Item2* were created.

10. You can also verify that the items were created in Sitecore by navigating to the `sitecore/content/Home/Terminal Integration Point` item.



Using the GetXML and InsertXML Methods

This section provides an example of how to use the GetXML and InsertXML methods in the Sitecore Web Service. The code in this example inserts an item with the ID = `exportRootItemId` under the item with the ID = `importParentItemId`.

```
        using System;
        using System.Collections.Generic;
        using System.Linq;
        using System.Text;
        using System.Xml.Linq;
        using System.Xml;
        using System.IO;
        namespace Terminal.Integration
        {
            class Program
            {
                static void Main(string[] args)
                {
                    var visualSitecoreService = new
WsSitecore.VisualSitecoreServiceSoapClient();
                    string exportRootItemId = "{5C913B55-7AF8-4E1D-9E39-0430D7A19AD5}";     //
Needs modification depending on your environment - Root Item ID of export item (ex.
/sitecore/content/Home)
                    string importParentItemId = "{B6B41E7D-5B61-44C4-88F4-CC7B4AEEE0DF}";   //
Needs modification depending on your environment - Parent Item ID of import item (ex.
/sitecore/content/Import Folder)
                    string strXMLPath = @" C:\Out.xml";  // Needs modification depending on
your environment - File path for Export/Import

                    string database = "master";

                    WsSitecore.Credentials credentials = new WsSitecore.Credentials();
                    credentials.UserName = @"sitecore\admin";
                    credentials.Password = "b";

                    while (true)
                    {
                        Console.WriteLine("Sitecore Terminal Web Service Integration");
```

```
                        Console.WriteLine("========================================" +
Environment.NewLine);
                        Console.WriteLine("Press any key to GetXML() to Out.xml" +
Environment.NewLine);
                        Console.ReadKey();
                        var resultXML = visualSitecoreService.GetXML(exportRootItemId, true,
database, credentials);

                        using (StreamWriter w = new StreamWriter(strXMLPath))
                        {
                            w.Write(resultXML);
                        }

                        Console.WriteLine("Press any key to InsertXML() from Out.xml" +
Environment.NewLine);
                        Console.ReadKey();

                        string readXML = String.Empty;
                        using (StreamReader r = new StreamReader(strXMLPath))
                        {
                            readXML = r.ReadToEnd();
                        }

                        XmlDocument doc = new XmlDocument();
                        doc.LoadXml(readXML);

                        var itemNode = doc.SelectSingleNode("sitecore/data/data/item");
                        var resultInsert = visualSitecoreService.InsertXML(importParentItemId,
itemNode.OuterXml, true, database, credentials);

                        Console.WriteLine(resultInsert);
                        Console.WriteLine(Environment.NewLine);
                        Console.WriteLine("Finished. Please close command window");
                        Console.ReadKey();
                        Console.Clear();
                    }
                }
            }
        }
```

Note
You must replace the values of the `exportRootItemId`, `importParentItemId`, `strXmlPath` variables and credentials on the appropriate values depending on your content tree structure to make it work.

Creating a Proxy for the Web Service

In this section, the developer can pass the service URL and the name of the invoked method to create a proxy for the web service, without having to add the service as a reference and build the application.

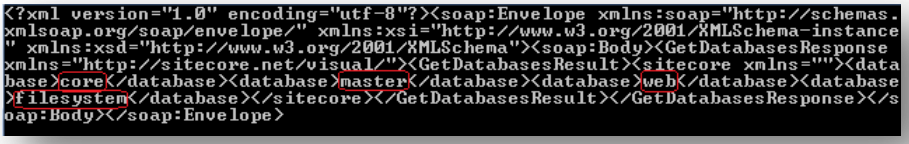
To create a proxy for the Sitecore web service:

1. Create a new console application and add the following code snippet to use the `GetDatabases` method:

```
        using System;
        using System.Collections.Generic;
        using System.Linq;
        using System.Text;
        using System.Xml;
        using Microsoft.Web.Services3;
        using Microsoft.Web.Services3.Addressing;
        using Microsoft.Web.Services3.Messaging;
        using System.Net;
        using System.IO;
        using System.Web;

        namespace SitecoreWebService
```

```
{
    class SitecoreWebService
    {
        public static void Main() { CallWebService(); }
        public static void CallWebService()
        {
            var _url = "http://yourhost/sitecore/shell/webservice/service.asmx";
            var  action = "http://sitecore.net/visual/GetDatabases";
            XmlDocument soapEnvelopeXml = CreateSoapEnvelope();
            HttpWebRequest webRequest = CreateWebRequest( url,  action);
            InsertSoapEnvelopeIntoWebRequest(soapEnvelopeXml, webRequest);
            // begin async call to web request.
            IAsyncResult asyncResult = webRequest.BeginGetResponse(null, null);
            // suspend this thread until call is complete.
            asyncResult.AsyncWaitHandle.WaitOne();
            // get the response from the completed web request.
            string soapResult;
            using (WebResponse webResponse = webRequest.EndGetResponse(asyncResult))
            using (StreamReader rd = new
            StreamReader(webResponse.GetResponseStream()))
            {
                soapResult = rd.ReadToEnd();
            }
            Console.Write(soapResult);
            Console.ReadKey();
        }
        private static HttpWebRequest CreateWebRequest(string url, string action)
        {
            HttpWebRequest webRequest = (HttpWebRequest)WebRequest.Create(url);
            webRequest.Headers.Add("SOAPAction", action);
            webRequest.ContentType = " text/xml;charset=\"utf-8\"";
            webRequest.Accept = " text/xml";
            webRequest.Method = "POST";
            return webRequest;
        }
        private static XmlDocument CreateSoapEnvelope()
        {
            XmlDocument soapEnvelop = new XmlDocument();
            //building the SOAP envelope
            soapEnvelop.LoadXml(@"<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>
            <GetDatabases   xmlns=""http://sitecore.net/visual/""><credentials>
            <Password>b</Password> <UserName>sitecore\admin</UserName></credentials>
            </GetDatabases></soap:Body></soap:Envelope>");
            return soapEnvelop;
        }
        private static void InsertSoapEnvelopeIntoWebRequest(XmlDocument
        soapEnvelopeXml, HttpWebRequest webRequest)
        {
            using (Stream stream = webRequest.GetRequestStream())
            {
                soapEnvelopeXml.Save(stream);
            }
        }
    }
}
```

2. The following image shows the SOAP response of the `GetDatabases` method with the databases that exist in Sitecore:



Support for Industry Standards

The following table describes the industry standards that the Sitecore web service supports:

Standard	Description
SOAP 1.1 and 1.2	Simple Object Access Protocol (SOAP) is an XML messaging standard used to format web service requests and responses. **Note** Security information is generally included in the SOAP message header and the message payload (for example, a purchase order) is part of the SOAP message body. For more information on SOAP, see http://www.w3.org/TR/SOAP
WSDL 1.1	WSDL is an XML format for describing network services containing RPC-oriented and message-oriented information. Developers and automated development tools can create WSDL files to describe a service and can make this description available over the Internet. Client-side developers and development tools can use published WSDL descriptions to obtain information about available web services and to build and create client proxies or program templates that access available services. For more information on WSDL, see http://www.w3.org/TR/wsdl
WS-I Basic Profile 1.1	Web Service-Interoperability (WS-I) is an organization that promotes web services interoperability across platforms, applications, and programming languages. The current release conforms to the WS-I Basic Profile 1.1. The ASMX 2.0 services category by definition provides support for the WS-I Basic Profile 1.1 and SOAP 1.2. This means support for XML 1.0, XML Schema Definition (XSD), WSDL, SOAP 1.1 and SOAP 1.2, and basic profile conformance validation at compile time. **Note** The Web Service Enhancements (WSE 3.0) and Windows Communication Foundation (WCF) categories pick up where ASMX 2.0 leaves off by providing support for some of the more advanced WS-* protocols.

Note
If you are creating your own service layer, we recommend that you make your services more generic by supporting SOAP (1.1 and 1.2) and REST which stands for Representative State Transfer.

Security

The Sitecore web service applies authentication and authorization by getting the user's credentials as a parameter and responding to the service consumer accordingly.

If you are creating your own service layer, we recommend that you make your services more secure to support the following important concepts:

- Transport-Level Security
- Application-Level Security

Transport-Level Security

The most commonly used transport-level data-communication protocol is Secure Socket Layer (SSL), otherwise known as Transport Layer Security (TLS) and provides:

- Authentication — communication is established between two trusted parties. This is already applied in the Sitecore web service.
- Confidentiality — the data exchanged is encrypted.
- Message integrity — the data is checked for possible corruption.
- Secure key exchange between client and server.

Application-Level Security

Application-level security complements transport-level security. Application-level security is based on XML frameworks that define confidentiality, integrity, authenticity; message structure; trust management; and identity propagation.

Security Considerations for the Sitecore Web Service

The following list describes the threats in the Sitecore web service and how to overcome them:

Unauthorized Access

The user name and password of the consumer is passed in plaintext in the SOAP envelope.

You can use the following countermeasures to prevent unauthorized access:

- Use password digests in SOAP headers for authentication.
- Use Kerberos tickets in SOAP headers for authentication.
- Use X.509 certificates in SOAP headers for authentication.
- Use role-based authorization to restrict access to web services. This can be done by using URL authorization to control access to the web service file (.asmx) or at the web method level by using principal-permission demands.

Parameter Manipulation

Parameter manipulation refers to the unauthorized modification of data that is sent between the web service consumer and the web service. For example, an attacker can intercept a web service message, as it passes through an intermediate node en route to its destination; and can then modify it before sending it on to its intended endpoint.

You can use the following countermeasures to prevent parameter manipulation:

- Digitally sign the message. The digital signature is used by the recipient to verify that the message has not been tampered with while it was in transit.
- Encrypt the message payload to provide privacy.

Network Eavesdropping

With network eavesdropping, an attacker is able to view web service messages as they flow across the network. For example, an attacker can use network monitoring software to retrieve sensitive data contained in a SOAP message. This might include sensitive application level data or credential information.

You can use the following countermeasures to protect sensitive SOAP messages as they flow across the network:

- Use transport level encryption such as SSL or IPSec. This is only applicable if you control both endpoints.
- Encrypt the message payload to provide privacy. This approach works in scenarios where your message travels through intermediary nodes route to the final destination.

Disclosure of Configuration Data

There are two main ways in which a web service can disclose configuration data.

First, the web service can support the dynamic generation of the WSDL file or it can provide the WSDL information in downloadable files that are available on the web server. This may not be desirable depending on your scenario.

Note
WSDL describes the characteristics of a web service, for example, its method signatures and supported protocols.

Second, with inadequate exception handling, the web service may disclose sensitive internal implementation details that might be useful to an attacker.

You can use the following countermeasures to prevent the unwanted disclosure of configuration data:

- Authorize access to the WSDL files using the NTFS permissions.
- Remove the WSDL files from the web server.
- Disable the documentation protocols to prevent the dynamic generation of the WSDL file.
- Capture exceptions and throw a `SoapException` or `SoapHeaderException` — that returns only minimal and harmless information — back to the client.

Message Replay

Web service messages can potentially travel through multiple intermediate servers. With a message replay attack, an attacker captures and copies a message and replays it to the web service impersonating the client. The message may or may not be modified.

The most common types of message replay attacks include:

- Basic replay attack

The attacker captures a message, copies it, and then replays the same message and impersonates the client. This replay attack does not require the malicious user to know the contents of the message.

- Man in the middle attack

  The attacker captures the message and then changes some of its contents, for example, a shipping address, and then replays it to the web service.

You can use the following countermeasures to address the threat of message replay:

- Use an encrypted communication channel, for example, SSL.
- Encrypt the message payload. Although this does not prevent basic replay attacks, it does prevent man in the middle attacks where the message contents are modified before being replayed.
- Use a unique message ID or nonce with each request to detect duplicates, and digitally sign the message for *tamperproofing*.

Note
A nonce is a cryptographically unique value used for the request.

When the server responds to the client it sends a unique ID and signs the message, including the ID. When the client makes another request, the client includes the ID with the message. The server ensures that the ID sent to the client in the previous message is included in the new request from the client. If it is different, the server rejects the request and assumes it is subject to a replay attack.

The attacker cannot spoof the message ID, because the message is signed. this only protects the server from client-initiated replay attacks that use the message request, and offers the client no protection against replayed responses.

Anonymous Access

You can use the IIS server to block anonymous access to the `/sitecore/shell/WebService` folder. For more information about limiting anonymous access to folders in IIS, see the *Sitecore Security Hardening Guide*.

Brute Force Attack

Brute Force attacks rely on computational power to crack hashed passwords or other secrets secured with hashing and encryption.

To mitigate this risk:

- Use strong passwords.
- Set a limit to the amount of times a user can enter his password incorrectly.

For more information on security threats and countermeasures, see http://msdn.microsoft.com/en-us/library/ff649874.aspx