

Note sui Web Services in ambiente C#

da <http://escher07.altervista.org>

Generalità

Che cos'è un Web Service? In due parole è un programma che riceve delle richieste strutturate come XML via HTTP e fornisce dei dati di ritorno utilizzando gli stessi protocolli. Direi che l'importanza dei WS sia dovuta al fatto che costituiscono un po' il tramite naturale fra una pagina Web ed un ambiente server side come ad esempio un database.

I dati che arrivano ad un WS in realtà non sono XML generico, quanto piuttosto un XML opportunamente standardizzato, secondo la cosiddetta specifica SOAP (Simple Object Access Protocol) definita dal W3C.

Un WS, per la sua natura di servizio esposto via Web (in contesto intranet o internet) sarà interrogato da una serie di client la cui tecnologia è solo limitatamente stabilibile a priori. Per questo non è importante solo che un WS fornisca i dati secondo determinati standard ma anche che descriva se stesso, ovvero i propri metodi esposti e i relativi argomenti o valori di ritorno in modo standard. A questo "pensa" un opportuno metodo presente in tutti i WS, il cosiddetto WSDL il cui output (ovvero l'elenco delle informazioni di cui sopra) deve rispettare l'omonima specifica "Web Service Description Language".

Un servizio "esposto" deve in qualche modo poter essere trovato. Pensiamo all'inutilità di un sito web senza un qualche motore di ricerca che lo indicizza. Per questo ci sono anche altre specifiche (ad esempio UDDI e WS-Discovery) che regolano il modo in cui un WS rende visibile alla rete di cui fa parte la sua presenza.

I WS possono essere sviluppati utilizzando varie tecnologie (es. Java, .NET etc...) : la standardizzazione in ambito WS non va tanto nella direzione delle tecnologie ma delle interfacce e nel formato dei dati che tali servizi forniscono.

Personalmente ho sviluppato dei WS solo in ambiente .NET utilizzando C# ed è a questo tipo di implementazione che mi riferirò nel resto del documento che è da intendersi come introduttivo all'argomento e dunque di livello assolutamente base.

Sicurezza

Il motivo per cui in generale i WS sono considerati una tecnologia "sicura" risiede nel fatto che non costringono ad aprire particolari "porte" : le richieste http girano di norma sulla porta 80 che deve essere obbligatoriamente aperta dal server web in DMZ verso l'esterno. Installando su questo un WS si possono in pratica fare delle elaborazioni con la potenza di un linguaggio server side, rendere disponibili i risultati a delle pagine Web senza, praticamente esporre alcun "punto debole" verso l'esterno.

Mi sembra ovvio che il grado di sicurezza dei WS sia anche legato a cosa fanno. In altre parole un WS che espone un unico metodo con cui si dropa un DB magari sembrerà sicuro dal punto di vista dell'amministratore del firewall ma di certo non lo può essere definito in pratica.

I WS tipicamente vengono invocati da pagine ASP o DHTML o anche da programmi di tipo client/server (es. Visual Basic) e mai direttamente dagli utenti. Per questo spesso succede di veder trascurati gli aspetti di autenticazione trovando WS con metodi esposti anche "potenti" raggiungibili digitandone un semplice URL (mi sono capitati anche in aziende grandi WS raggiungibili dalla rete che espongono anche metodi di Insert o Update...).

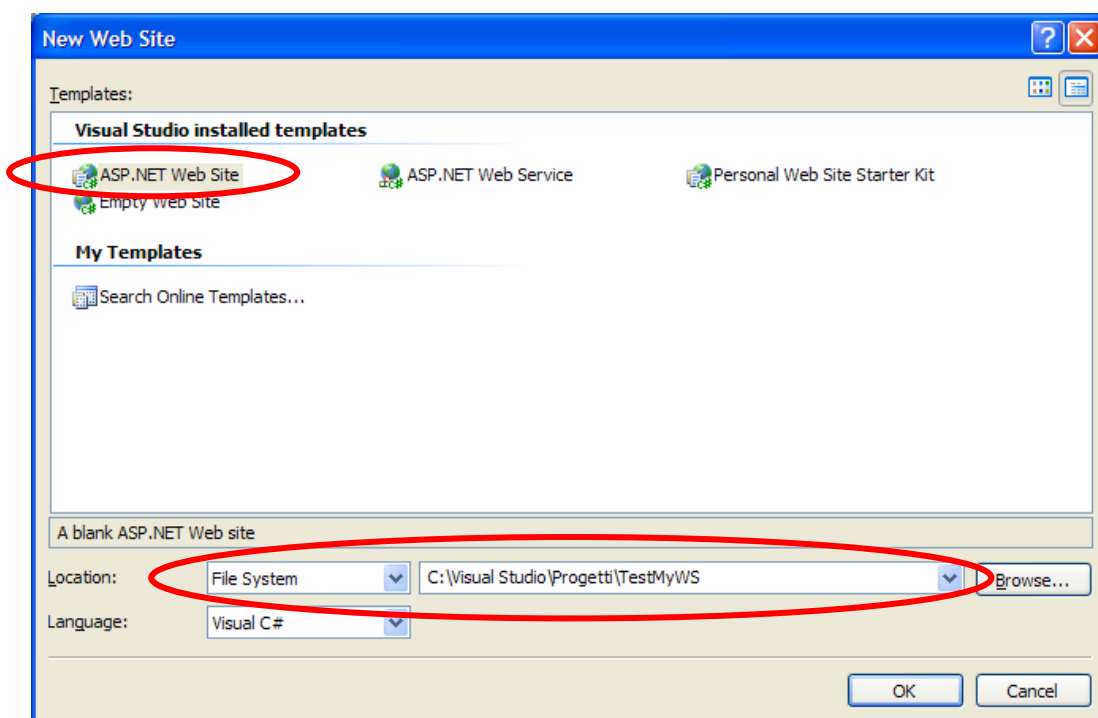
A livello di base una certa sicurezza si ottiene già mettendo fra gli argomenti dei metodi esposti un Username ed una Password o in alternativa esponendo un metodo Autenticate(User,Password) che sia un prerequisito all'esecuzione degli altri.

Soluzioni più raffinate prevedono l'utilizzo di certificati digitali sul client e sul server o l'inclusione di Username e Password negli header del SOAP.

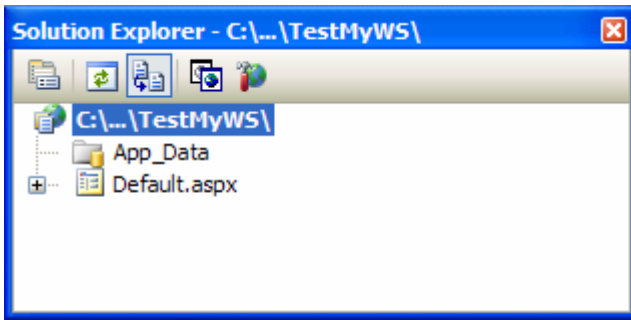
Come si scrive un WS in .NET

La cosa più semplice è cominciare dall'applicazione contenitore ovvero quella in cui in genere si mette un bottone "Test Service" per testare il nostro servizio.

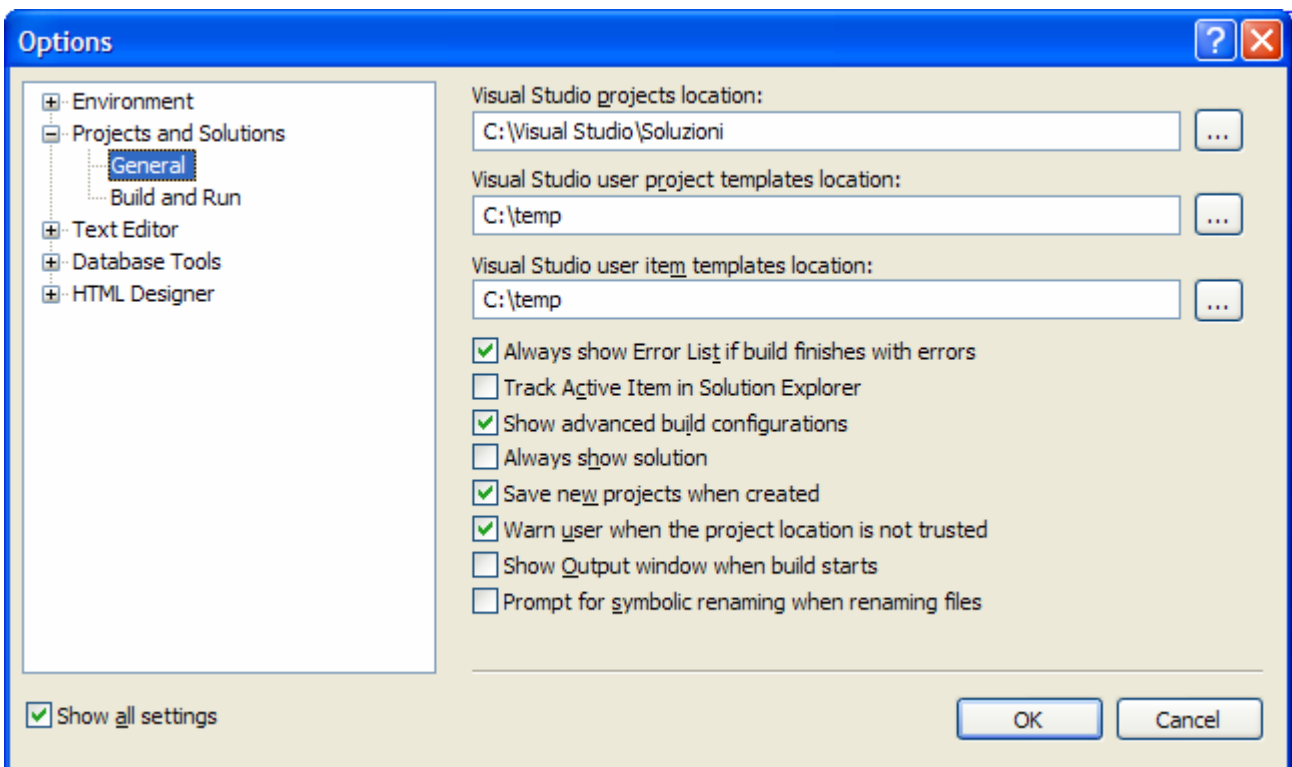
Dunque, si apre Visual Studio Express e da File si sceglie New Web Site, dando ad esempio questi parametri:



... abbiamo ottenuto una soluzione vuota come questa:



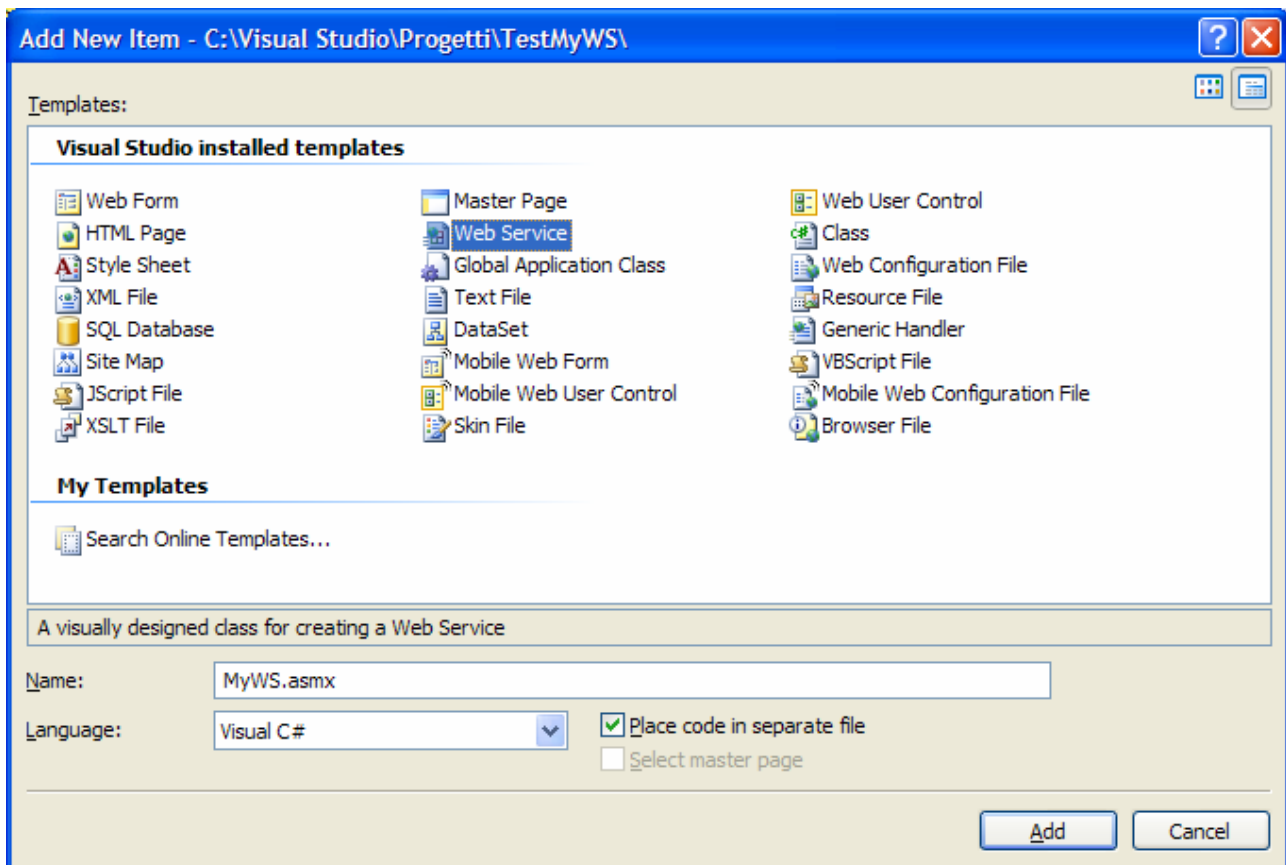
Fisicamente questi file saranno stati messi nella locazione impostata in precedenza (C:\Visual Studio\Progetti\TestMyWS nel nostro caso). La corrispondente soluzione (ovvero il file .sln che costituisce un po' l'«indice» degli stessi si troverà invece nella locazione impostata in Tools/Options, ovvero:



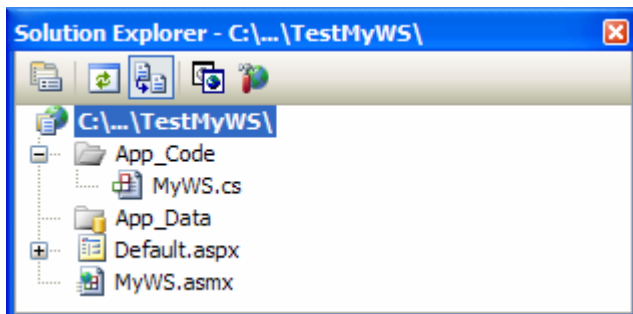
Avrò in altri termini due files con estensione .sln e .suo in una cartella di path C:\Visual Studio\Soluzioni\TestMyWS che VS avrà creato automaticamente.

Per inciso se una gestione di questo tipo vi sembra contorta sono d'accordo con voi e credo sia dovuta alle limitazioni che Microsoft mette nella Express Edition (che non dovrebbe gestire le soluzioni ma solo i progetti, tanto è vero che non esiste una funzione File/New/Solution come nelle edizioni non free di VS, ma che in pratica non ne può fare a meno).

Inseriamo ora nel progetto il WS vero e proprio : Dal menu contestuale del Solution Explorer aggiungiamo ora un "Existing Item" di tipo Web Service, dando ad esempio queste impostazioni:



Senza aver ancora scritto niente, ci troviamo in Solution Explorer questi files:



MyWS.asmx è in pratica la semplice chiamata al codice contenuto in MyWS.cs, il cui contenuto sarà siffatto:

```
using System;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;

/// <summary>
/// Summary description for MyWS
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class MyWS : System.Web.Services.WebService {

    public MyWS () {
```

```

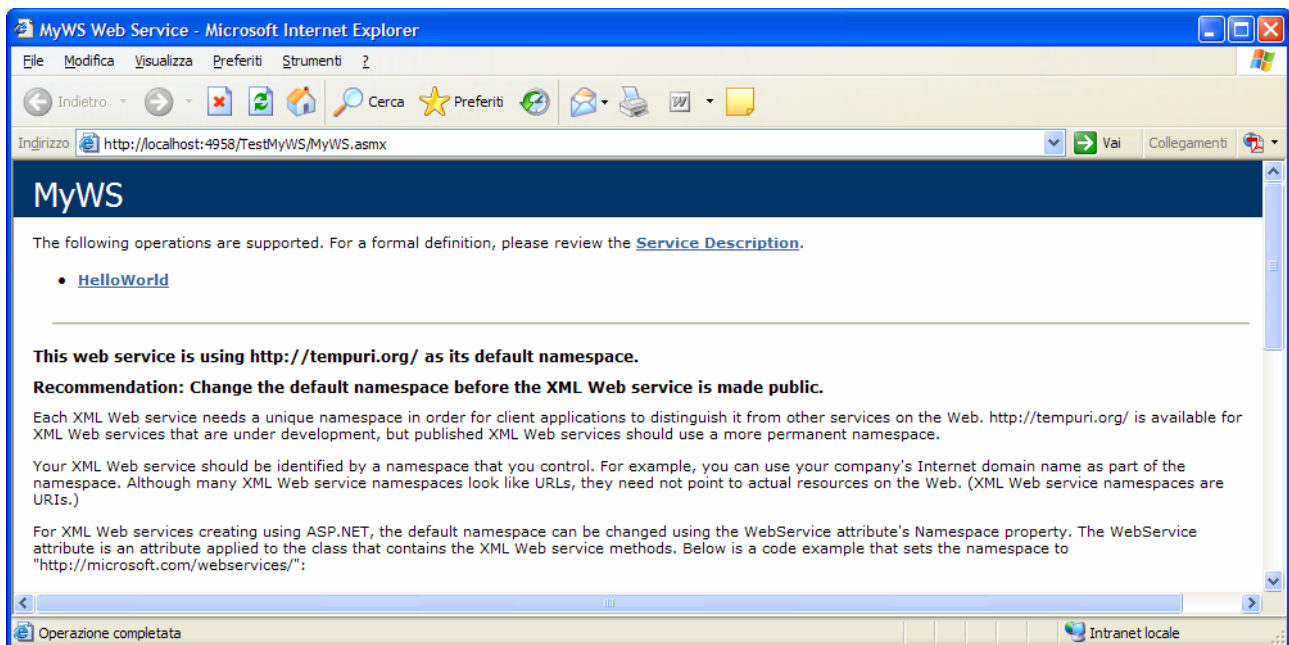
        //Uncomment the following line if using designed components
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
}

```

Abbiamo già un Web Method (ovvero metodo esposto del WS) di esempio che al di là della dichiarazione ([WebMethod]) è una normalissima function di C# che posso “complicare” come voglio esattamente come se la scrivessi in ambiente client/server.

Lanciando il Run VS aggiunge in automatico in Web.config (dopo averci chiesto se farlo oppure no) e viene visualizzata la “home page” del WS:



..in cui scegliendo il metodo HelloWorld (l'unico...) abbiamo il nostro SOAP di risposta:

```

<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Hello World</string>

```

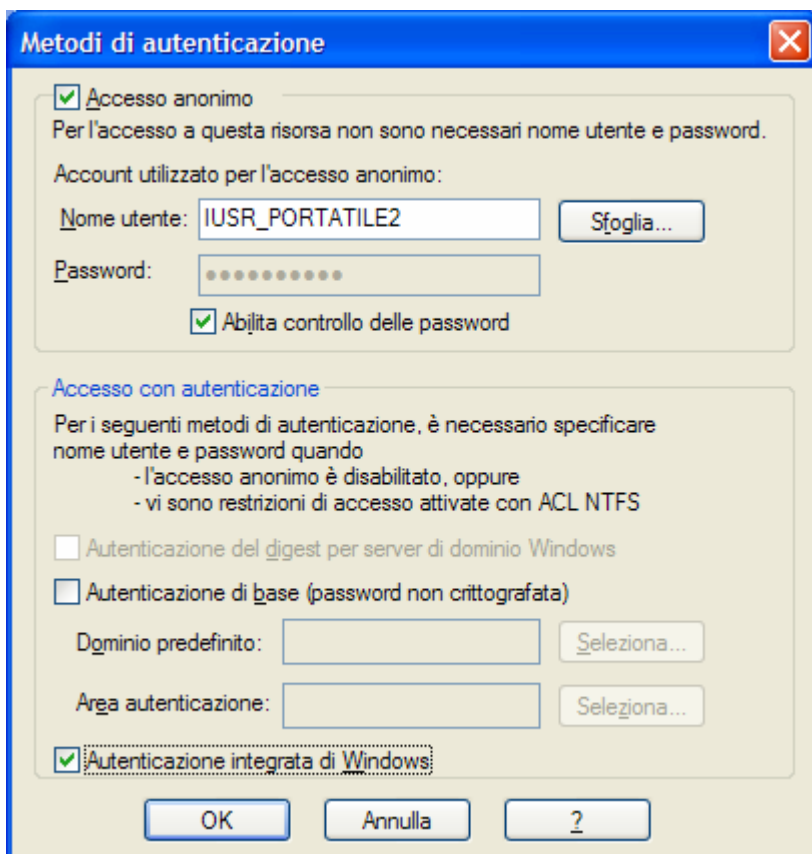
Come si può vedere la pagina di accesso al WS non ci è mostrata attraverso il vero Web Server che andremo ad usare, ma attraverso il Web Server di test incluso il VS, che risponde in questo caso sulla porta 4958.

Per fare in modo che il WS sia visibile dalla rete e/o per testare in modo “più serio” il medesimo occorre mapparne la corrispondente cartella su Web Server di produzione, in questa ipotesi un IIS sulla medesima macchina.

Per farlo utilizziamo poi lo snap-in di IIS e aggiungiamo una "Nuova directory virtuale" puntando alla cartella dove si trova il nostro progetto ovvero in questo caso:

C:\Visual Studio\Progetti\TestMyWS

Diamo poi i permessi di lettura ed esecuzione, indichiamo Default.aspx come pagina iniziale ed infine dalla sottoscheda "Protezione Directory/Modifica" delle proprietà della nostra Virtual Directory abilitiamo l'autenticazione integrata:



Fatto questo possiamo verificare che digitando nel browser il seguente link:

<http://localhost/TestMyWS/MyWS.aspx>

ovvero quello che, localhost a parte sarà quello utilizzato dai client si riottiene la pagina di accesso/test al MyWS di cui sopra.

Nota : è ovviamente possibile definire un WS anche senza una applicazione contenitore da File/New Web Sites/ASP Web Service. Personalmente preferisco la soluzione con l'applicazione contenitore, magari con pagina Default.aspx vuota come in questo caso, in modo da avere già pronta la struttura per eventuali prove un po' più complesse di quelle consentite dalle sole maschere standard. Chiaramente nel caso di WS "stand alone" il test sarà fatto direttamente nella applicazione che utilizzerà il WS che lo avrà fra le Web References.

Come si invoca un WS in .NET

Facciamo qualche esempio di come si invoca un WS, ipotizzando di avere su un certo Web Server un Web Service con queste caratteristiche:

Nome : MioWs

URL : <http://www.miodominio.local/MioWs>

Metodi Esposti : GetData(string SQLCommand) return value string (dati in XML)

Il modo più semplice per invocare un WS è da browser, ovvero digitando il corrispondente URL. Così facendo viene visualizzata la pagina iniziale (in genere la .asmx) ed essenzialmente delle form coi link ai vari metodi esposti, che così possono essere testati. Quindi:

<http://www.miodominio.local/MioWs>

Chiama la form di test dei metodi del WS

<http://www.miodominio.local/MioWs?WSDL>

Ottiene la descrizione (WSDL) del Web Service

<http://www.miodominio.local/MioWs?op=GetData>

Chiama il metodo GetData del WS. Se il metodo è senza argomenti si ottiene direttamente il valore di ritorno, altrimenti appare una form in cui gli argomenti vengono richiesti, la stessa a cui si arriva cliccando sul link di GetData in <http://www.miodominio.local/MioWs>.

Se al WS devono essere passati dei parametri la cosa più comoda è utilizzare uno degli approcci seguenti.

L'invocazione da un linguaggio server-side come C# viene fatta con queste istruzioni:

```
www.MioWS objWebMethod = new www.MioWS();
string strResult = "";
string strSQLSELECT='SELECT * FROM TUA_TABELLA';
strResult = objWebMethod.GetData(strSQLSelect);
```

Si è fatta l'ipotesi implicita che i WS che sono girano sulla macchina www sono stati mappati nel progetto C# tramite l'aggiunta del riferimento web (in Visual Studio Express dal menù contestuale in Solution Explorer "Add Web Reference").

In alternativa il WS può essere invocato da un linguaggio client-side come Javascript. In questo caso si deve aggiungere al sito web un opportuno file (webservice.htc) di descrizione identico per tutti i WS e scaricabile dal sito Microsoft e poi procedere come segue:

```
<body onload="Init()">
<div id="service" style="BEHAVIOR: url(webservice.htc)"></div>
..
</body>

<script id="ws_connect" language="javascript">
```

```

function Init()
{
    var service;
    var strSELECT;
    strSELECT='SELECT * FROM TUA_TABELLA';
    service = document.getElementById("service");
    service.useService("http://www.miodominio.local/MioWS.asmx?WSDL","MioWS");
    var iCallID = service.MioWS.callService(ElabData, 'GetData', strSELECT);
}

function ElabData(result)
{
    var text = result.value; // result string
    if (!result.error)
    {
        if (text.indexOf("<err>") > -1 )
        {
            alert(text);
        }
        //
        // gestisci il risultato
        //
    }
}
</script>

```

Debug di un WS in .NET

Nella Web.config del progetto WS mettere:

```

<configuration>
<system.web>
<compilation debug="true">
</system.web>
</configuration>

```

Fatto questo si fissano i breakpoint desiderati e lancia il run della applicazione contenitore (quella che per intenderci ha in il codice C# di cui al paragrafo precedente).

Notiamo che la documentazione Microsoft afferma che questo flag abilitato va tolto al momento del rilascio in produzione perché determina una diminuzione delle prestazioni. Non ho motivo di dubitarne ☺

Links

<http://punto-informatico.it/p.aspx?i=186993>

Articolo "Il W3C rifinisce il linguaggio dei Web service" su Punto Informatico, sulle strategie e lo stato dell'arte della standardizzazione dei WS.

<http://msdn2.microsoft.com/en-us/library/ms531032.aspx>

Documentazione microsoft su webservice.htc

<http://www.asptalia.com/articoli/asp.net/wsautenticazione.aspx>

Panoramica sulla sicurezza dei WS.

http://www.dia.unisa.it/professori/auletta/DIDATTICA/PROGRET1_B/seminari/Security_WS.pdf

Seminario dell'Università di Salerno sulla sicurezza dei WS.