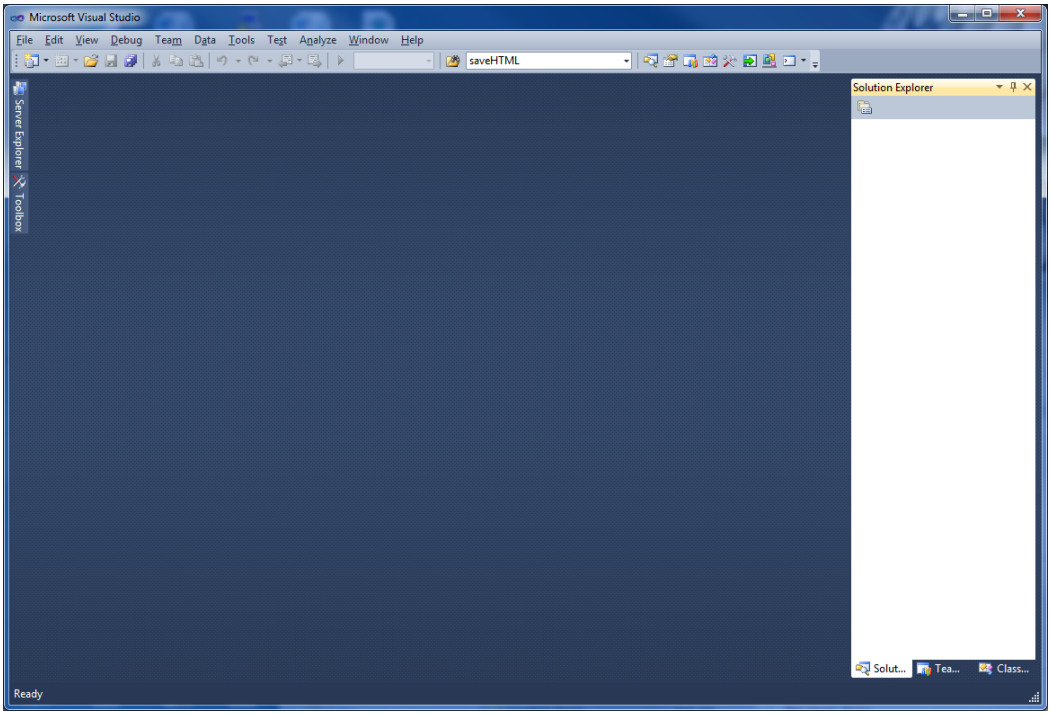
Building a Custom Solution using the GGA Web Services

This tutorial will guide you through creating a piece of software that uses the Web Services provided by the Georgia General Assembly as a back-end for data. To follow along with this guide you will need:

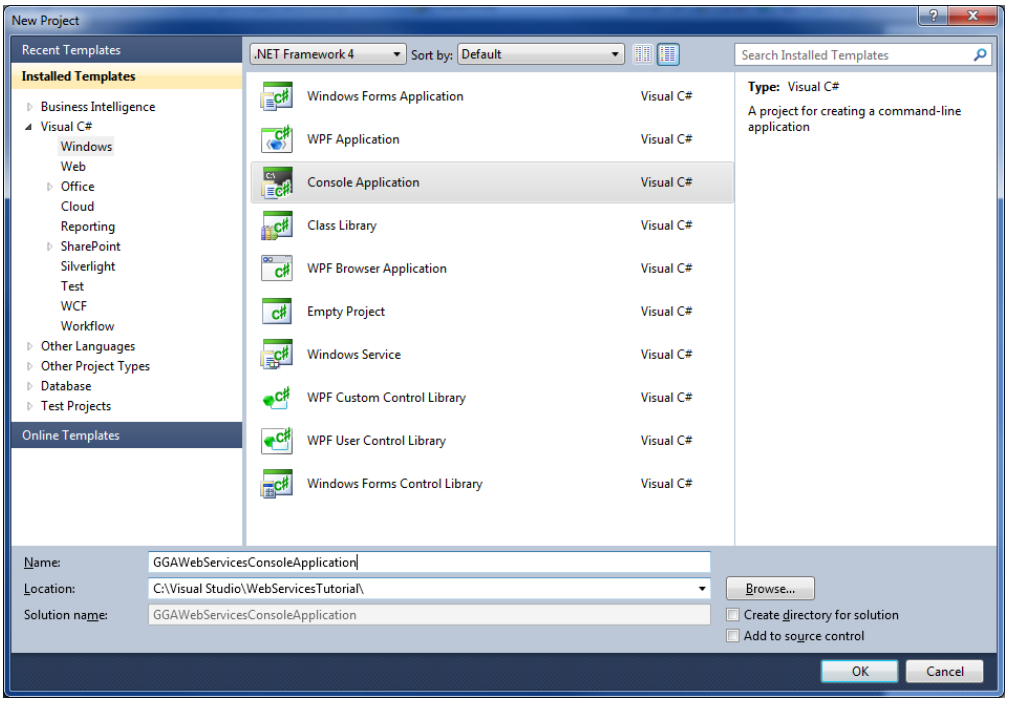
- Microsoft Visual Studio with C# or Microsoft Visual C# Express. Microsoft Visual C# Express is available free of charge at http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-products. Choose the version you are interested in. We will be developing a Console Application, which is a project type that is available in Visual Studio for Windows Desktops. Although, the current version of Visual Studio is Visual Studio 2012, I will be developing this solution using Visual Studio 2010.
- Knowledge of Microsoft Visual C# programming language, or an equivalent programming language, such as Java.

You are not limited to using C#. You can use the basic steps in this tutorial to build your application using Microsoft Visual Basic.NET, which is also available in Visual Studio. The steps will be the same, it would only be the language details that vary. It is also possible to use our Web Services using Java; the Axis library from Apache is useful in creating proxy classes. I also understand that at least one third party is using PHP, but the scope of this article will be limited to Visual Studio and C#.

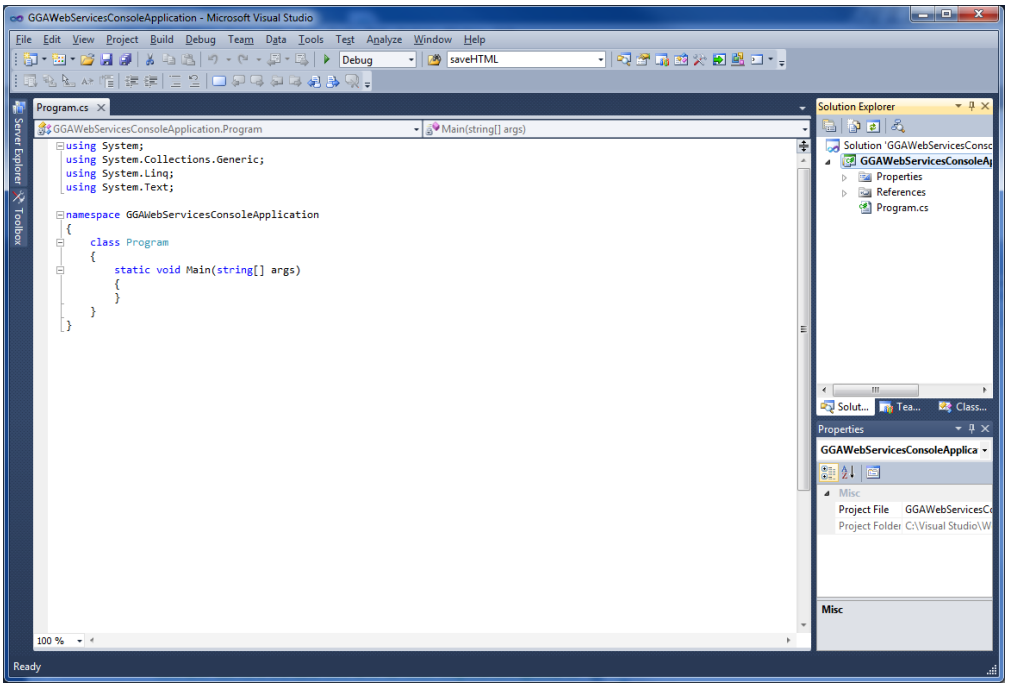
Start Visual Studio



Start Microsoft Visual Studio and click on File… New Project. Choose Console Appication and give your project an appropriate name and location on your file system (I chose not to create a new directory for my solution:



Your solution should look like so:

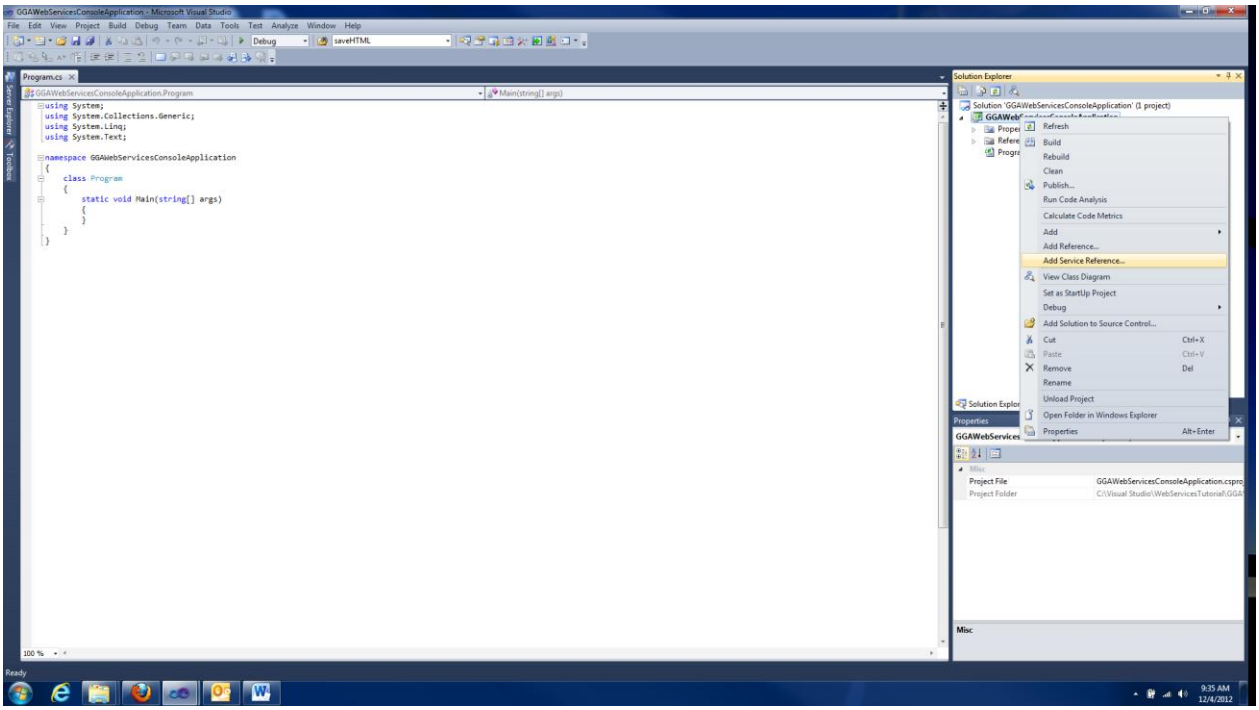


Create Service References to the Georgia General Assembly Web References

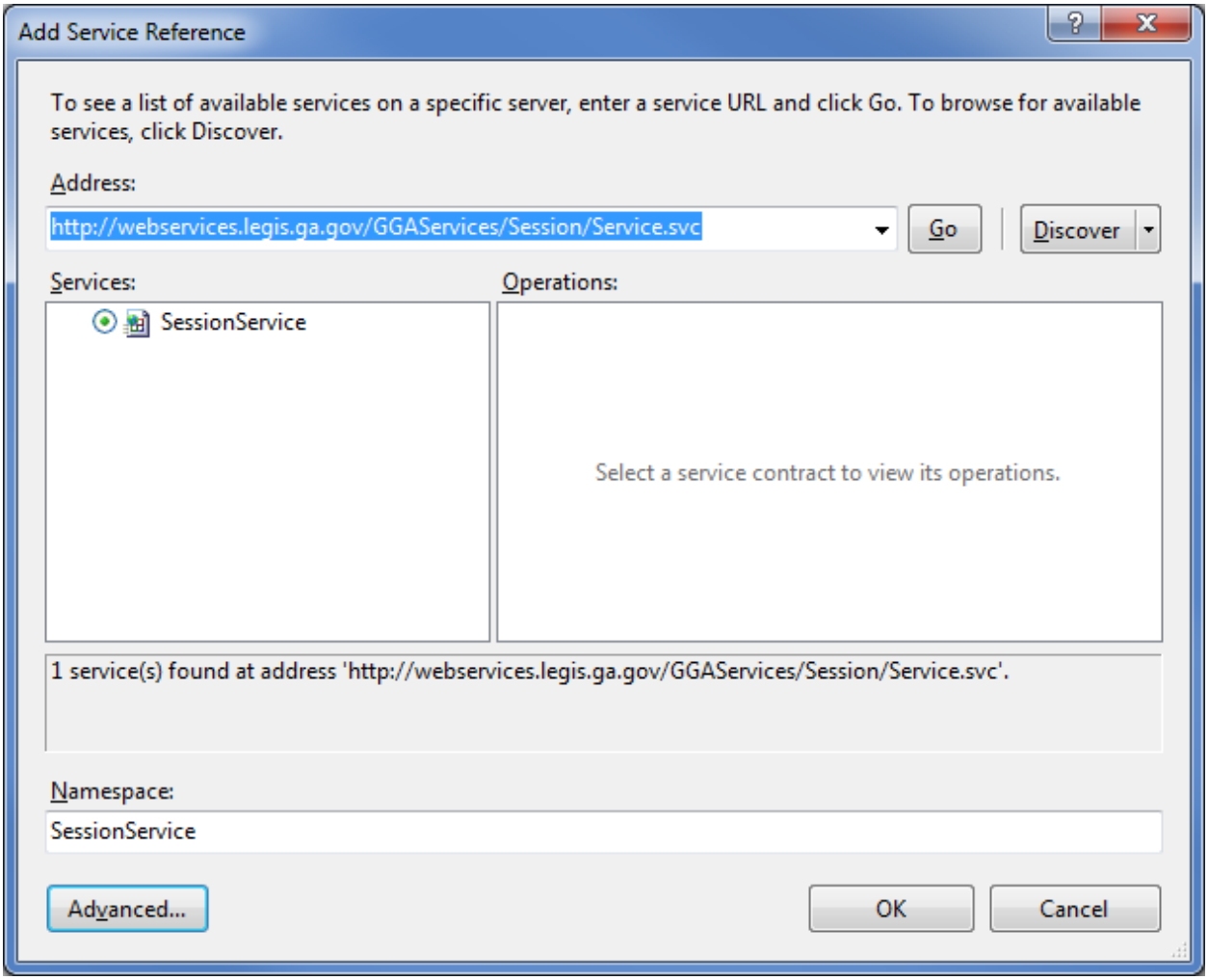
For the purposes of this tutorial, we will simply create all of our code and components in the same project; however, in a real-world solution you would be well advised to create separate projects to allow for a separation of concerns. For example, you may want to create a project of type Class Library to contain your Data Access Components; then, whenever you need these Data Access Components in a different project (a Web Site, for example), you can simply create a reference to your Class Library.

The first step you will need to take regardless of whichever programming environment you are using will be to create a set of proxy classes that will interact with our services. In Visual Studio, this is as simple as adding a Service Reference.

Right-Click on your Console Application in the Solution Explorer and select Add Service Reference:



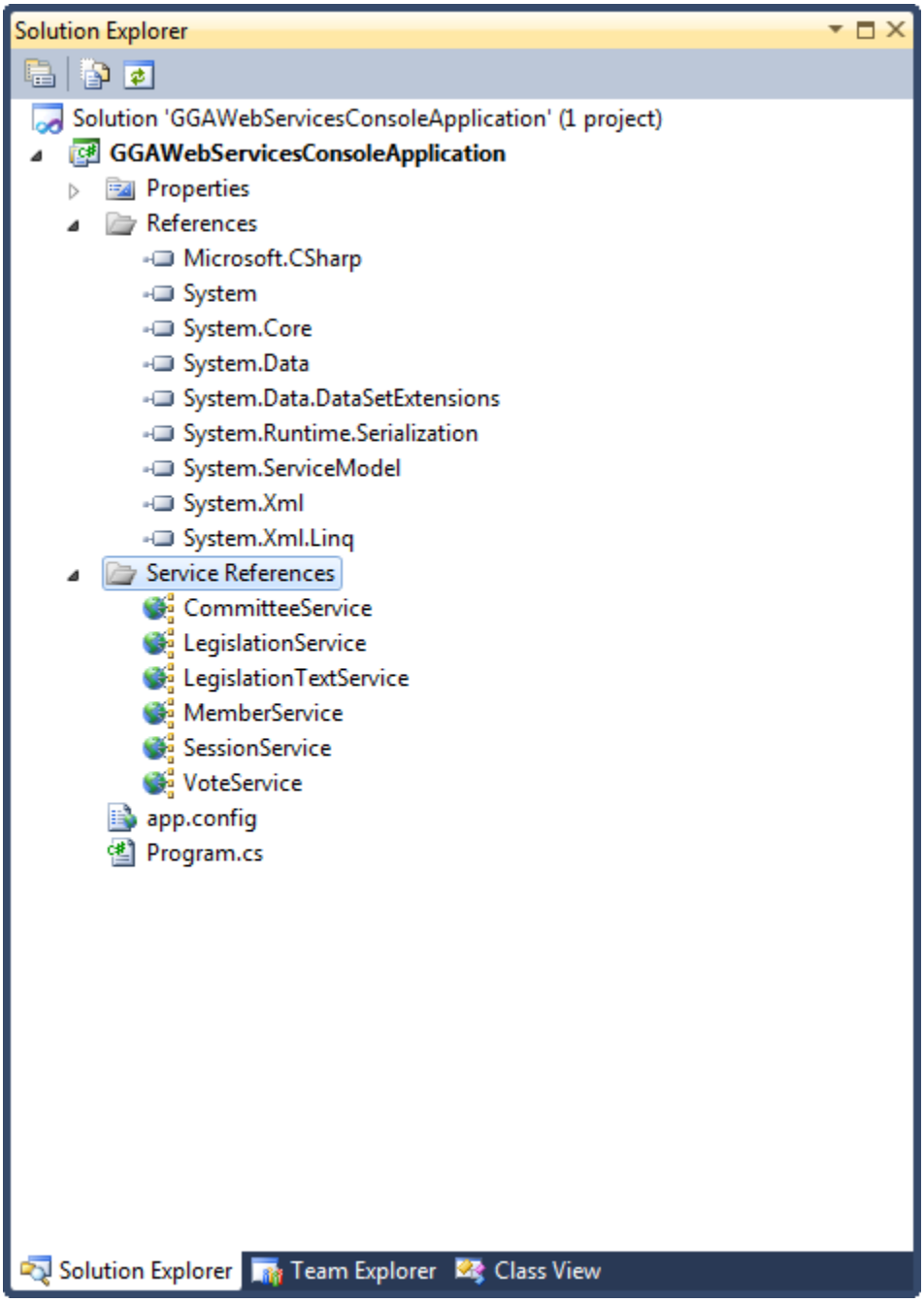
The first service we will add is the Session Service. Type its address in the Address Combo Box and click Go. Give it the Namespace SessionService:



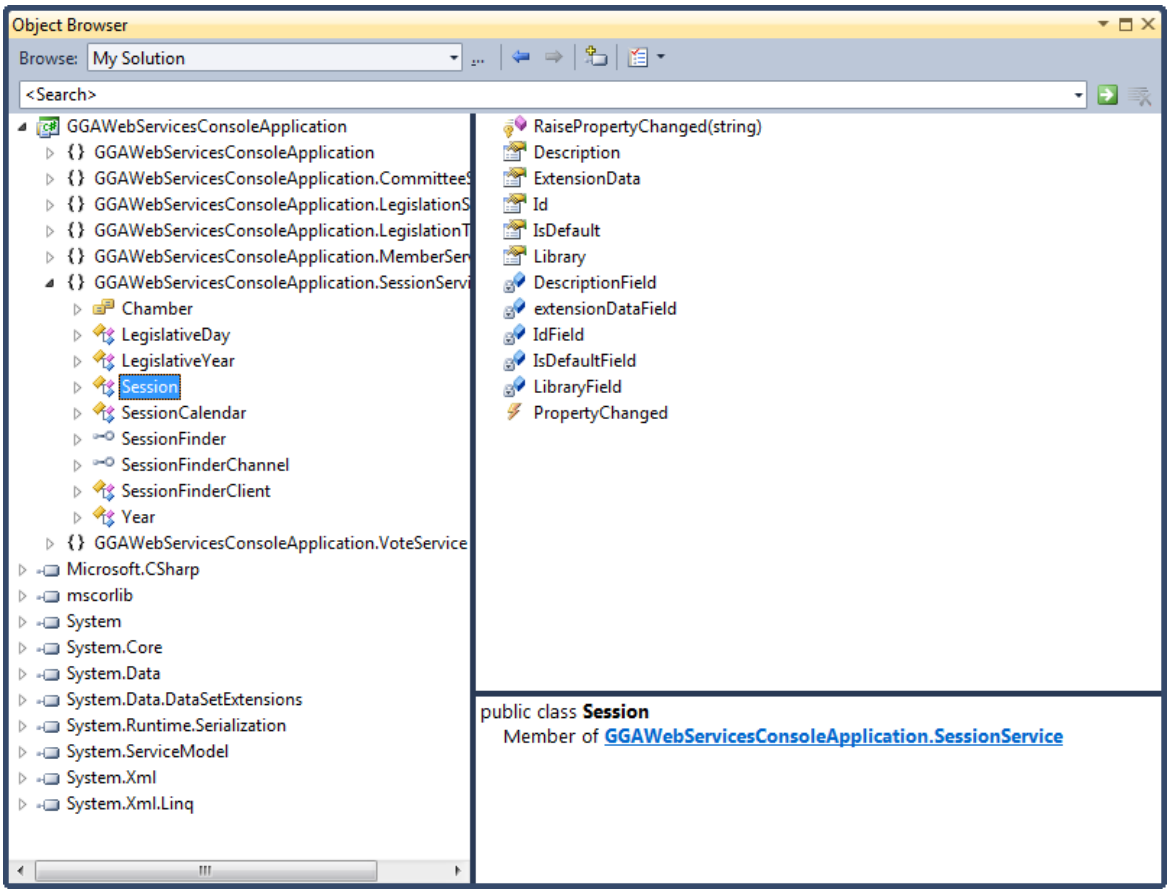
Repeat for remaining services, giving each an appropriate Namespace:

- http://webservices.legis.ga.gov/GGAServices/Legislation/Service.svc
- http://webservices.legis.ga.gov/GGAServices/LegislationText/Service.svc
- http://webservices.legis.ga.gov/GGAServices/Members/Service.svc
- http://webservices.legis.ga.gov/GGAServices/Committees/Service.svc
- http://webservices.legis.ga.gov/GGAServices/Votes/Service.svc

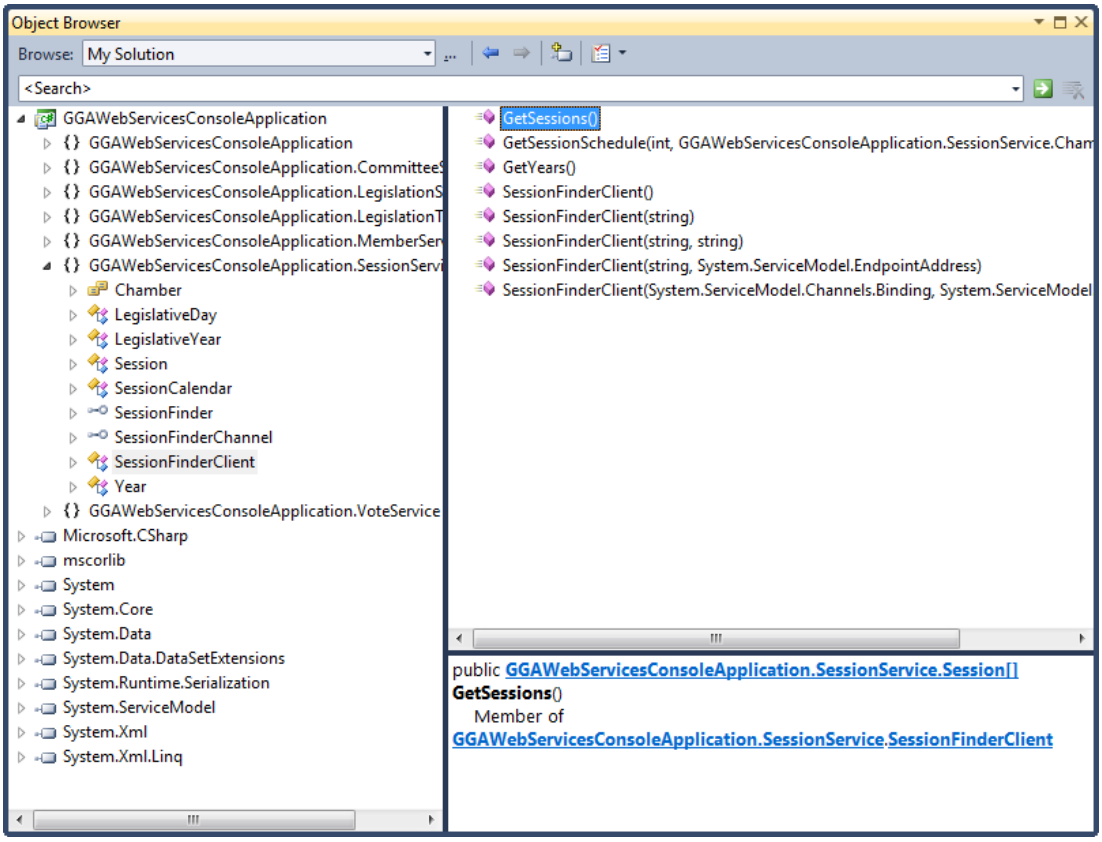
Your Solution Explorer should now appear as follows:



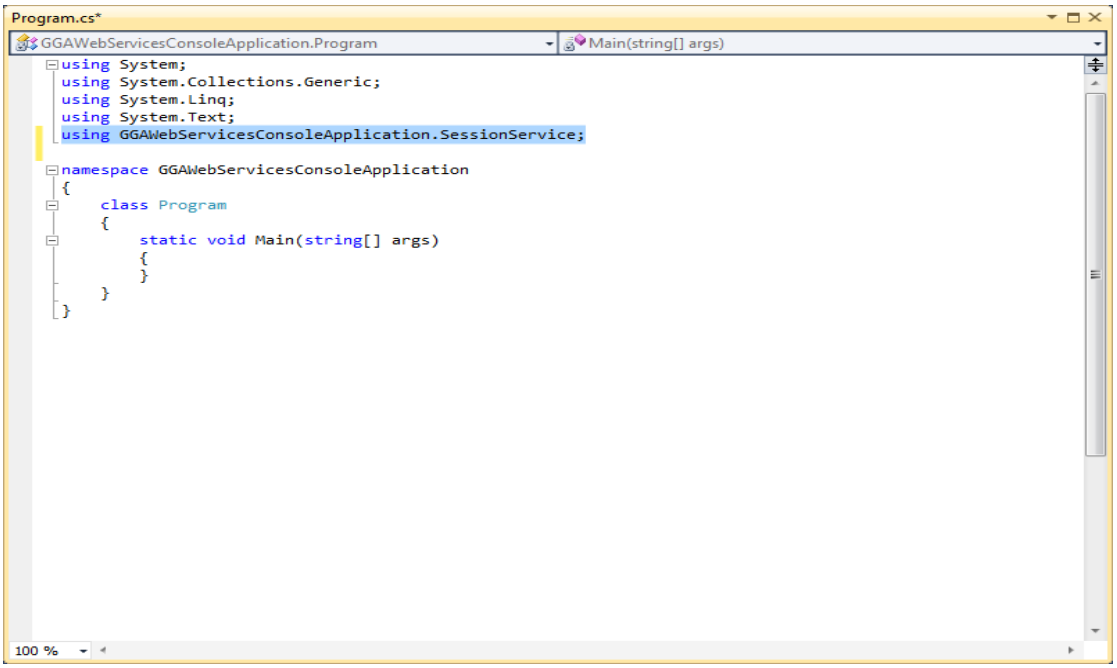
Double-Click on SessionService. The Object Browser will appear. If you expand the GGAWebServicesConsoleApplication.SessionService Node and select the Session Node, your Object Browser will look as follows:



These are the properties of the Session objects that the Session Service will return. But how do we get these objects? Click on the SessionFinderClient node and then click on the GetSessions() method. That's how!



SessionFinderClient is the entry level class for the SessionService proxy. It has a method called GetSessions() that returns an Array of Session. So let's get the available Sessions! First, we need to add a using statement to our Console Application's code file:



Now we can access the Session Service in this code file. Modify void Main by inserting the following code:

```
using (SessionFinderClient sessionService = new SessionFinderClient())
{
        var sessions = sessionService.GetSessions();
        foreach (var session in sessions)
        {
                Console.WriteLine(session.Description + " can be located by it's Id: " +
        session.Id);
        }
        Console.ReadLine();

}
```

Your code file will now look like this:

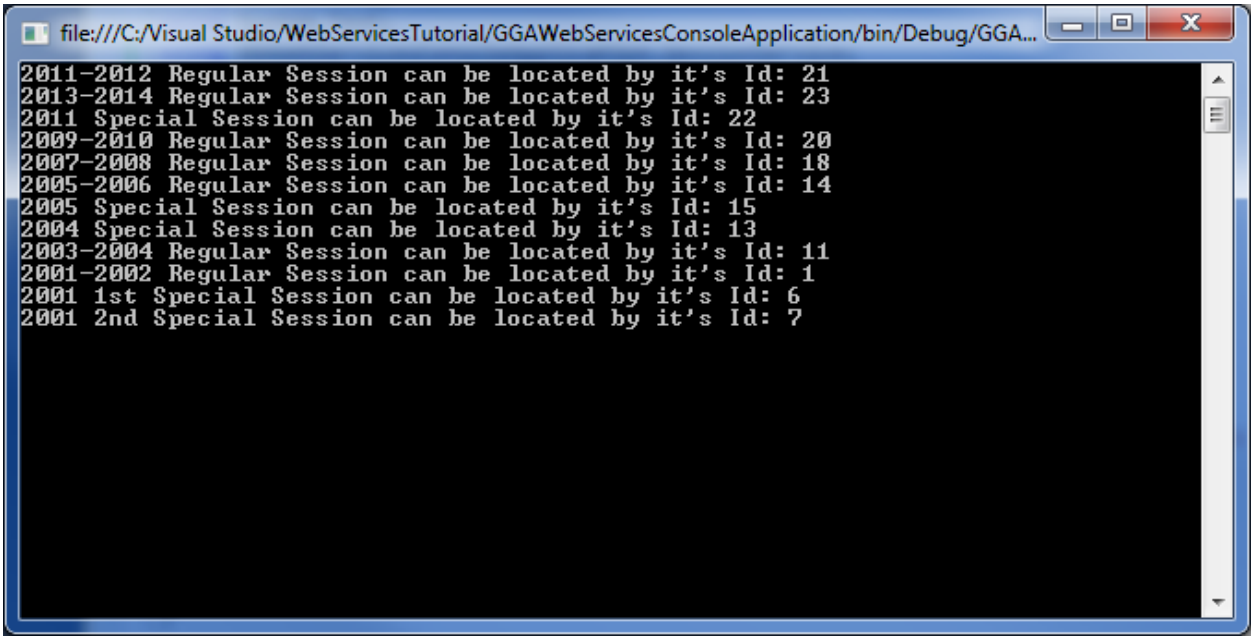
Program.cs*

GGAWebServicesConsoleApplication.Program | Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using GGAWebServicesConsoleApplication.SessionService;

namespace GGAWebServicesConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            using (SessionFinderClient sessionService = new SessionFinderClient())
            {
                var sessions = sessionService.GetSessions();
                foreach (var session in sessions)
                {
                    Console.WriteLine(session.Description + " can be located by it's Id: " + session.Id);
                }
            }
            Console.ReadLine();
        }
    }
}
```

100 %

Press F5 to run the program. The output will display in a Console Window:



The first Session in the list is the *Default* Session. Currently, this is the **2011-2012 Regular Session**, although this will soon change to the **2013-2014 Regular Session**. When it *does* change, the **2013-2014 Regular Session** will be the first item in the list. The Session objects have an IsDefault Property which will tell you whether or not the Session object you have a reference to is currently the Default Session.

Next, we will query for a LegislativeDay in the 2011-2012 Regular Session. Insert the following code after the end brace that follows the Console.WriteLine:

```
Session thisSession = sessions.Single(c => c.Id == 21);
var schedule = sessionService.GetSessionSchedule(thisSession.Id, Chamber.House);
foreach (var year in schedule.Years)
{
        var days = year.Days;
        foreach (var day in days)
        {
                Console.WriteLine("The " + day.Branch + " met in " + year.Year + " for
        Legislative Day " + day.Number + " on " + day.Date);
        }
}
```

The c => c.Id == 21 is a *Lambda Expression*. It should be read as *c* **goes to** *c,* of which the **Id** property *should equal* **21.** The use of the value ***c*** is not of special importance, it is merely a variable used to construct the Lambda; we could have said ***s=>s.Id==21*** and gotten the same result. This type of expression allows us to pass a filter to the Single() method to find the Session we want.

We could also have written this expression like this:

```
Session thisSession = sessions.Single(c => c.Description.Contains("2011-2012"));
```

This Lambda Expression will instruct the Single method to find the One and Only Session that contains the characters **2011-2012** in its Description Property. Note: Single() will throw an exception if the item does not exist. You can use SingleOrDefault() instead if you like: this method will return Null if the item does not exist. Also, Single() will throw an exception if more than one match satisfies the Predicate that we constructed in our Lambda Expression. If we are not certain that our object exists in the collection, we could use the Where() extension method instead.

Your code window should now look as follows:

Program.cs*

GGAWebServicesConsoleApplication.Program | Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using GGAWebServicesConsoleApplication.SessionService;

namespace GGAWebServicesConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            using (SessionFinderClient sessionService = new SessionFinderClient())
            {
                var sessions = sessionService.GetSessions();
                foreach (var session in sessions)
                {
                    Console.WriteLine(session.Description + " can be located by it's Id: " + session.Id);
                }
                Session thisSession = sessions.Single(c => c.Id == 21);
                var schedule = sessionService.GetSessionSchedule(thisSession.Id, Chamber.House);
                foreach (var year in schedule.Years)
                {
                    var days = year.Days;
                    foreach (var day in days)
                    {
                        Console.WriteLine("The " + day.Branch + " met in " + year.Year + " for Legislative Day " + day.Number + " on " + day.Date);
                    }
                }
            }
            Console.ReadLine();
        }
    }
}
```

100 %

Run your program. Your output should look similar to mine:

```
file:///C:/Visual Studio/WebServicesTutorial/GGAWebServicesConsoleApplication/bin/Debug/GGA...
The House met in 2011 for Legislative Day 17 on 2/22/2011 12:00:00 AM
The House met in 2011 for Legislative Day 18 on 2/23/2011 12:00:00 AM
The House met in 2011 for Legislative Day 19 on 2/24/2011 12:00:00 AM
The House met in 2011 for Legislative Day 20 on 2/28/2011 12:00:00 AM
The House met in 2011 for Legislative Day 21 on 3/1/2011 12:00:00 AM
The House met in 2011 for Legislative Day 22 on 3/2/2011 12:00:00 AM
The House met in 2011 for Legislative Day 23 on 3/3/2011 12:00:00 AM
The House met in 2011 for Legislative Day 24 on 3/4/2011 12:00:00 AM
The House met in 2011 for Legislative Day 25 on 3/7/2011 12:00:00 AM
The House met in 2011 for Legislative Day 26 on 3/8/2011 12:00:00 AM
The House met in 2011 for Legislative Day 27 on 3/10/2011 12:00:00 AM
The House met in 2011 for Legislative Day 28 on 3/11/2011 12:00:00 AM
The House met in 2011 for Legislative Day 29 on 3/14/2011 12:00:00 AM
The House met in 2011 for Legislative Day 30 on 3/16/2011 12:00:00 AM
The House met in 2011 for Legislative Day 31 on 3/21/2011 12:00:00 AM
The House met in 2011 for Legislative Day 32 on 3/22/2011 12:00:00 AM
The House met in 2011 for Legislative Day 33 on 3/23/2011 12:00:00 AM
The House met in 2011 for Legislative Day 34 on 3/28/2011 12:00:00 AM
The House met in 2011 for Legislative Day 35 on 3/29/2011 12:00:00 AM
The House met in 2011 for Legislative Day 36 on 3/30/2011 12:00:00 AM
The House met in 2011 for Legislative Day 37 on 3/31/2011 12:00:00 AM
The House met in 2011 for Legislative Day 38 on 4/11/2011 12:00:00 AM
The House met in 2011 for Legislative Day 39 on 4/12/2011 12:00:00 AM
The House met in 2011 for Legislative Day 40 on 4/14/2011 12:00:00 AM
```

Now that we have demonstrated a couple of methods for working with the Session Service, we will refactor the code a little to prepare ourselves for the next step of the tutorial. Replace *all* of the code inside of void Main() with the following:

```
using (SessionFinderClient sessionService = new SessionFinderClient())
{
	Session thisSession = sessionService.GetSessions().Single(c => c.Id == 21);
	var year = sessionService.GetSessionSchedule(thisSession.Id,
	Chamber.House).Years.Single(c=>c.Year==2012);
	var sineDieYear2012 = year.Days.SingleOrDefault(c=>c.Number==40);
	if (sineDieYear2012 != null)
	{
		Console.WriteLine("Sine Die for year " + year.Year + " of the " +
		thisSession.Description + " occured on " + sineDieYear2012.Date);
	}
	else
	{
		Console.WriteLine("Sine Die for year " + year.Year + " of the " +
		thisSession.Description + " has not yet occured.");
	}
}
Console.ReadLine();
```

We will run our program to obtain the following output:

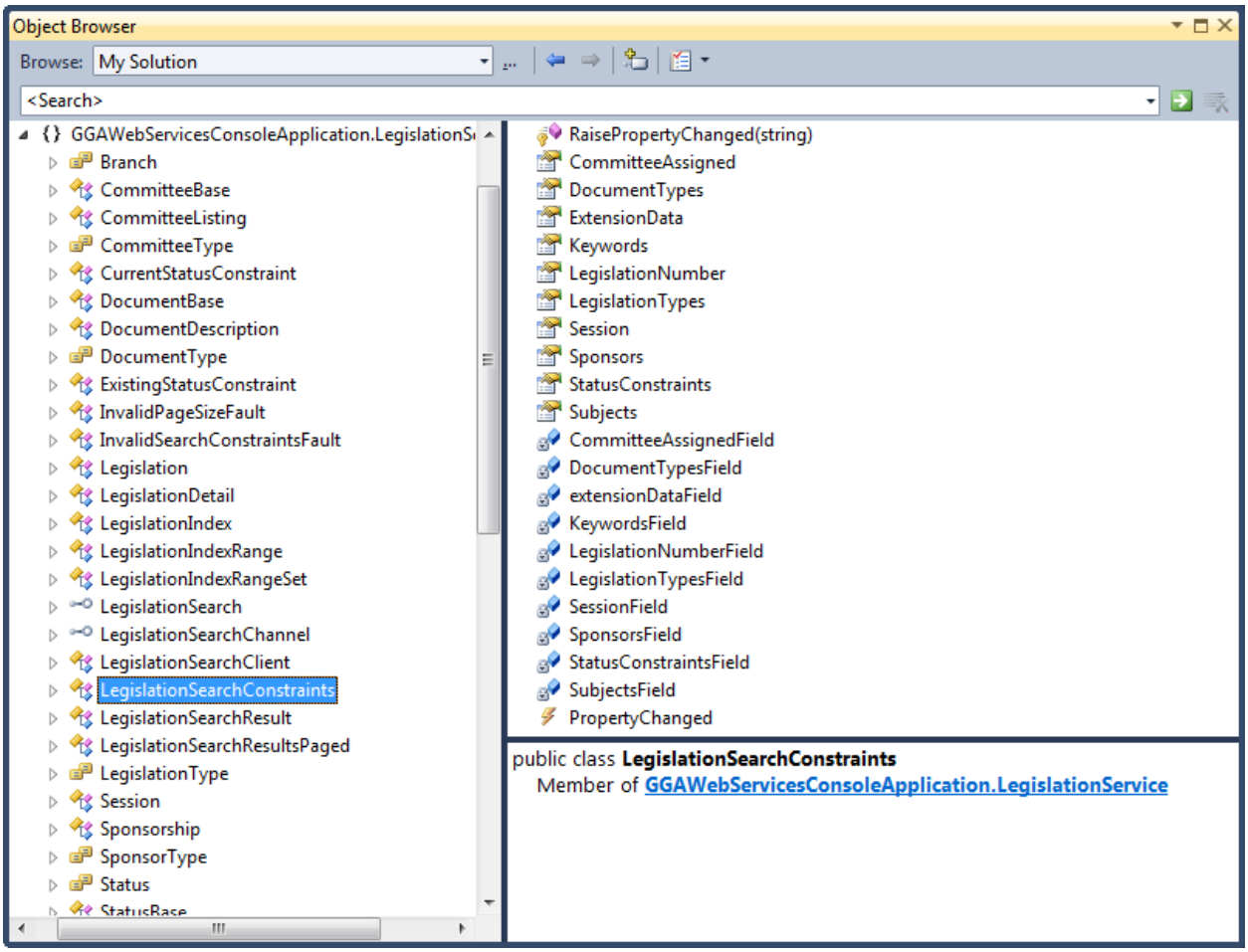
```
file:///C:/Visual Studio/WebServicesTutorial/GGAWebServicesConsoleApplication/bin/Debug/GGA...
Sine Die for year 2012 of the 2011-2012 Regular Session occured on 3/29/2012 12:
00:00 AM
```

Now we will remove the Console.WriteLine code, because all we need for the next step is the Session and the Date of Sine Die. Remove these lines and your code window will look as follows:

```
Program.cs*
GGAWebServicesConsoleApplication.Program        Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using GGAWebServicesConsoleApplication.SessionService;

namespace GGAWebServicesConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            using (SessionFinderClient sessionService = new SessionFinderClient())
            {
                Session thisSession = sessionService.GetSessions().Single(c => c.Id == 21);
                var year = sessionService.GetSessionSchedule(thisSession.Id, Chamber.House).Years.Single(c=>c.Year==2012);
                var sineDieYear2012 = year.Days.SingleOrDefault(c=>c.Number==40);
            }
            Console.ReadLine();
        }
    }
}
100 %
```

So, this Session stuff is great, now what about the actual **Legislation?!?!?**

The workhorse of the Legislation Service is an object called LegislationSearchConstraints. Let's look at this object in the Object Browser. Double-Click on the LegislationService in the Solution Explorer and when the Object Browser is displayed Double-Click on the GGAWebServicesConsoleApplication.LegislationService Node to expand it. Select the LegislationSearchConstraints node to look at its Properties:



This object is the Primary argument for the GetLegislationSearchResultsPaged method of the LegislationSearchClient object (which is our main proxy class to the Legislation Service.) This method allows you the most flexibility when trying to search for Legislation – it is the method used by *our* Search Page on the General Assembly Website, , as well as many other pages on our site, such as House Prefiled Legislation, Senate First Readers, and Signed By Governor, among others. Each of the properties, when set, will represent an *AND* Predicate, restricting the legislation data returned by the service. Many of these properties are Arrays; this means that you pass in an Array of the appropriate type, causing the Search Engine to create an *OR* Predicate for the values contained in the Array, and then an *AND* Predicate that will join this *OR* Predicate to the base query. For example, the Sponsors property accepts an Array of int. If we pass an Array of Member Ids to this property, the Search Engine

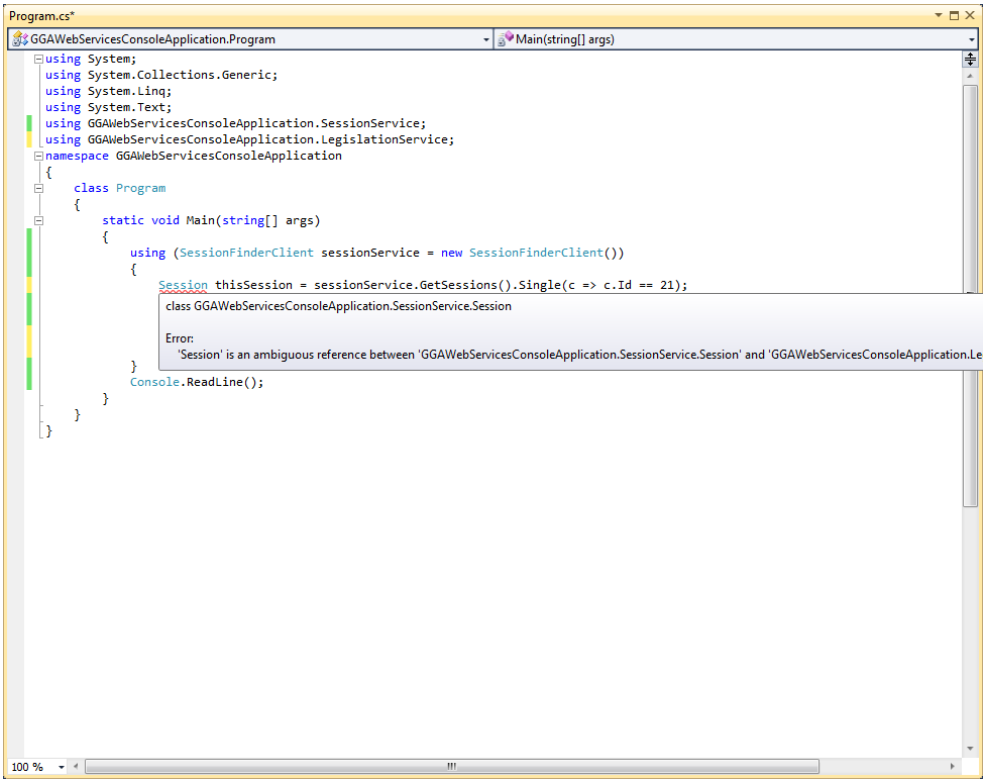
will restrict our search to legislation that was sponsored by one of the members whose Id matches one of the int values passed in the Array. (Although we will not cover it in this tutorial, you can obtain a List of members for a Session by using the Member Service. The objects returned that represent Members will have an Id property – this is the value that should be passed as an Array item to the Sponsors property)

The only property on the LegislationSearchConstraints that you are *required* to set is the Session. Unfortunately the service requires you to construct a Session object (instead of just passing the Id as an int) and it must be in the same namespace as the Legislation Service, which means we *cannot* use the Session object that we obtained from the Session Service – because *that* Session object is in the *Session Service* namespace. However, we can easily construct a new Session in the Legislation Service namespace with the only property that LegislationSearchConstraints is concerned with – the Session.Id property.

We need to modify our code file a bit. First, we need to add a using statement for the LegislationService. Add the following statement under the using statement for the SessionService:

```
using GGAWebServicesConsoleApplication.LegislationService;
```

Visual Studio immediately complains about our Session object:



This is easily resolved by specifying the appropriate namespace. Change

```
Session thisSession = sessionService.GetSessions().Single(c => c.Id == 21);
```

To

```
SessionService.Session thisSession = sessionService.GetSessions().Single(c => c.Id ==
21);
```

Studio is happy once again. Now we will write some code that uses the LegislationSearchClient. Enter the following after `var sineDieYear2012 = year.Days.SingleOrDefault(c=>c.Number==40);`

```
int pageSize = 10;
int startIndex = 0;

// work with Legislation Service
LegislationSearchClient legislationSearch = new
LegislationService.LegislationSearchClient();
LegislationSearchConstraints constraints = new LegislationSearchConstraints();

// we need to create the Session in the LegislationService namespace.  The only property
that is important
// is the Id Property
constraints.Session = new LegislationService.Session { Id = thisSession.Id };

// get the first page of 10 results
var results = legislationSearch.GetLegislationSearchResultsPaged(constraints, pageSize,
startIndex);
Console.WriteLine("The first Page of results for our search, which had " + results.Total
+ " hits:");
var page = results.Page;
foreach (var result in page)
{
Console.WriteLine(string.Format("{0} {1}{2} - {3}", result.DocumentType,
result.Number,result.Suffix, result.Caption));
}
legislationSearch.Close();
```

Your code window will look like this:

Program.cs*

GGAWebServicesConsoleApplication.Program | Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using GGAWebServicesConsoleApplication.SessionService;
using GGAWebServicesConsoleApplication.LegislationService;
namespace GGAWebServicesConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            using (SessionFinderClient sessionService = new SessionFinderClient())
            {
                SessionService.Session thisSession = sessionService.GetSessions().Single(c => c.Id == 21);
                var year = sessionService.GetSessionSchedule(thisSession.Id, Chamber.House).Years.Single(c=>c.Year==2012);
                var sineDieYear2012 = year.Days.SingleOrDefault(c=>c.Number==40);

                int pageSize = 10;
                int startIndex = 0;

                // work with Legislation Service
                LegislationSearchClient legislationSearch = new LegislationService.LegislationSearchClient();
                LegislationSearchConstraints constraints = new LegislationSearchConstraints();

                // we need to create the Session in the LegislationService namespace.  The only property that is important
                // is the Id Property
                constraints.Session = new LegislationService.Session { Id = thisSession.Id };

                // get the first page of 10 results
                var results = legislationSearch.GetLegislationSearchResultsPaged(constraints, pageSize, startIndex);
                Console.WriteLine("The first Page of results for our search, which had " + results.Total + " hits:");
                var page = results.Page;
                foreach (var result in page)
                {
                    Console.WriteLine(string.Format("{0} {1}{2} - {3}", result.DocumentType, result.Number,result.Suffix, result.Caption));
                }
                legislationSearch.Close();
            }
            Console.ReadLine();
        }
    }
}
```

100 %

Run your program to achieve the following output:

file:///C:/Visual Studio/WebServicesTutorial/GGAWebServicesConsoleApplication/bin/Debug/GGA...

```
The first Page of results for our search, which had 5424 hits:
HB 1 - Crimes and Offenses; prenatal murder unlawful; provide
HB 2 - Georgia Right to Grow Act; enact
HB 3 - Constitutional Tender Act; enact
HB 4 - Life, Liberty, and Property Restoration Act; enact
HB 5 - Freedom of Choice and Security Act; enact
HB 6 - Emergency Defense of the Home Act; enact
HB 7 - Right to Travel Act; enact
HB 8 - Due Process Restoration Act; enact
HB 9 - Kathryn Johnston's Law; enact
HB 10 - Child Protection Act; enact
```

Great! But what if we only want Senate Bills and Senate Resolutions?

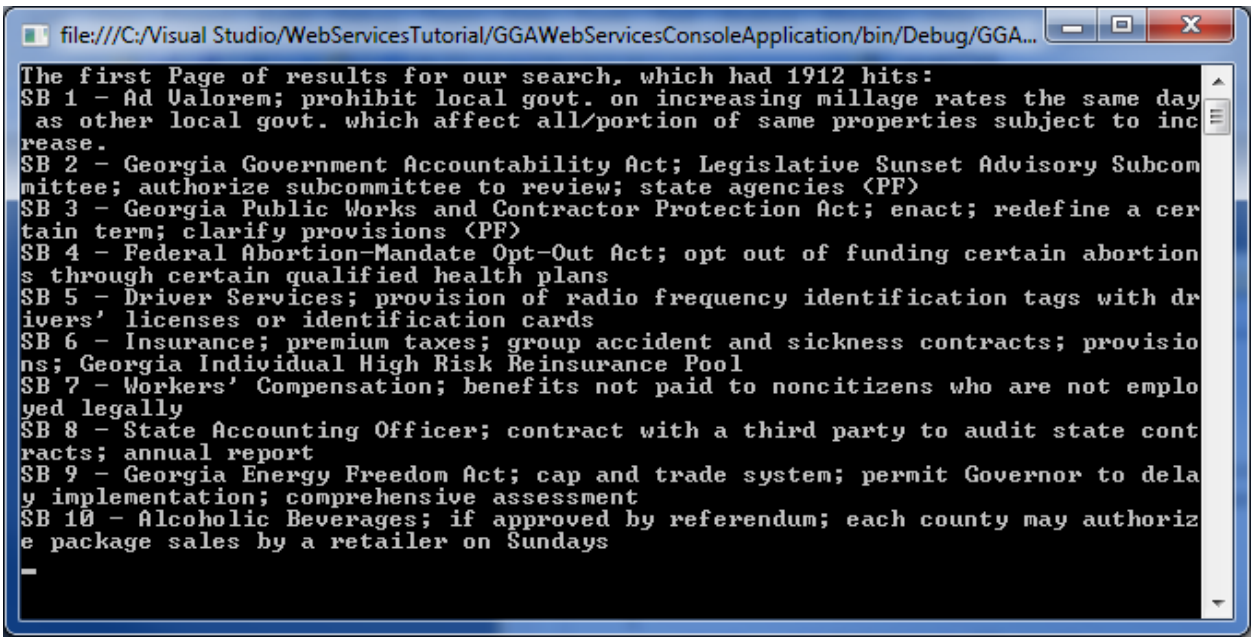
Add

```
constraints.DocumentTypes = new DocumentType[] { DocumentType.SB, DocumentType.SR };
```

under

```
constraints.Session = new LegislationService.Session { Id = thisSession.Id };
```

Run your program again to achieve the following output:



But I only want Senate Bills and Resolutions that have some form of the word ***tax***!

(Note: for instructions on how to phrase keywords, please see http://www.legis.ga.gov/legislation/en-US/SearchHelp.aspx, specifically the area marked by an asterisk *)

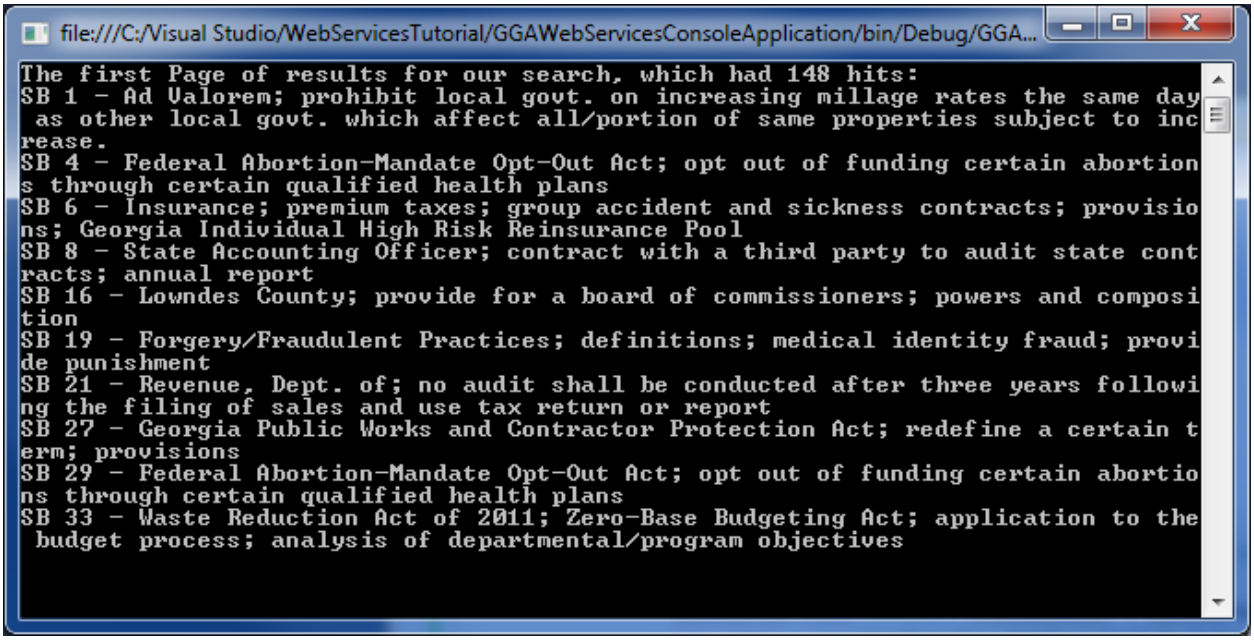
Add

```
constraints.Keywords = "tax*";
```

under

```
constraints.DocumentTypes = new DocumentType[] { DocumentType.SB, DocumentType.SR };
```

Run again:



Nice! But I don't *really* want the Senate Bills and Resolutions with tax in at least one of their text versions – I *actually* want ALL of the bills that had action on the Last Day of Session. After all, why else did I go through all of the trouble to create a variable called sineDie?

Remove

```
constraints.DocumentTypes = new DocumentType[] { DocumentType.SB, DocumentType.SR };
```

and

```
constraints.Keywords = "tax*";
```

and add

```
StatusConstraint statusConstraint = new ExistingStatusConstraint();
statusConstraint.DateStart = sineDieYear2012.Date;
statusConstraint.DateEnd = sineDieYear2012.Date;
constraints.StatusConstraints = new StatusConstraint[] { statusConstraint };
```

Running now will give us:

```
file:///C:/Visual Studio/WebServicesTutorial/GGAWebServicesConsoleApplication/bin/Debug/GGA...
The first Page of results for our search, which had 357 hits:
HB 39 - Education; unexcused absence notices made by United States mail; provide

HB 100 - Georgia Tax Court; create
HB 129 - Conveyances; future conveyance of real property; prohibit fee
HB 133 - Income tax payment; de minimis overpayments and insufficiencies; provis
ions
HB 175 - Online Clearinghouse Act; enact
HB 181 - Special needs scholarship program; waiver one requirement; provide
HB 198 - Superior court clerks; real estate or personal property filing fees; ex
tend sunset dates
HB 237 - Residential mortgage fraud; mortgage lending process; revise definition

HB 247 - Community Health, Department of; fingerprint and investigate emergency
medical services personnel; require
HB 318 - Sales and use tax exemptions; donated food for hunger relief; extend
```

Note: We did not set the Status property of the StatusConstraint. This property is an enum defined below:

```
Object Browser
Browse: My Solution
<Search>
{} GGAWebServicesConsoleApplication.LegislationS
    Branch
    CommitteeBase
    CommitteeListing
    CommitteeType
    CurrentStatusConstraint
    DocumentBase
    DocumentDescription
    DocumentType
    ExistingStatusConstraint
    InvalidPageSizeFault
    InvalidSearchConstraintsFault
    Legislation
    LegislationDetail
    LegislationIndex
    LegislationIndexRange
    LegislationIndexRangeSet
    LegislationSearch
    LegislationSearchChannel
    LegislationSearchClient
    LegislationSearchConstraints
    LegislationSearchResult
    LegislationSearchResultsPaged
    LegislationType
    Session
    Sponsorship
    SponsorType
    Status
    StatusBase

    Any
    FirstReadInHouse
    FirstReadInSenate
    HouseAction
    HouseOrSenateAction
    Prefiled
    SenateAction
    SignedByGovernor

public enum Status
  Member of GGAWebServicesConsoleApplication.LegislationService
```

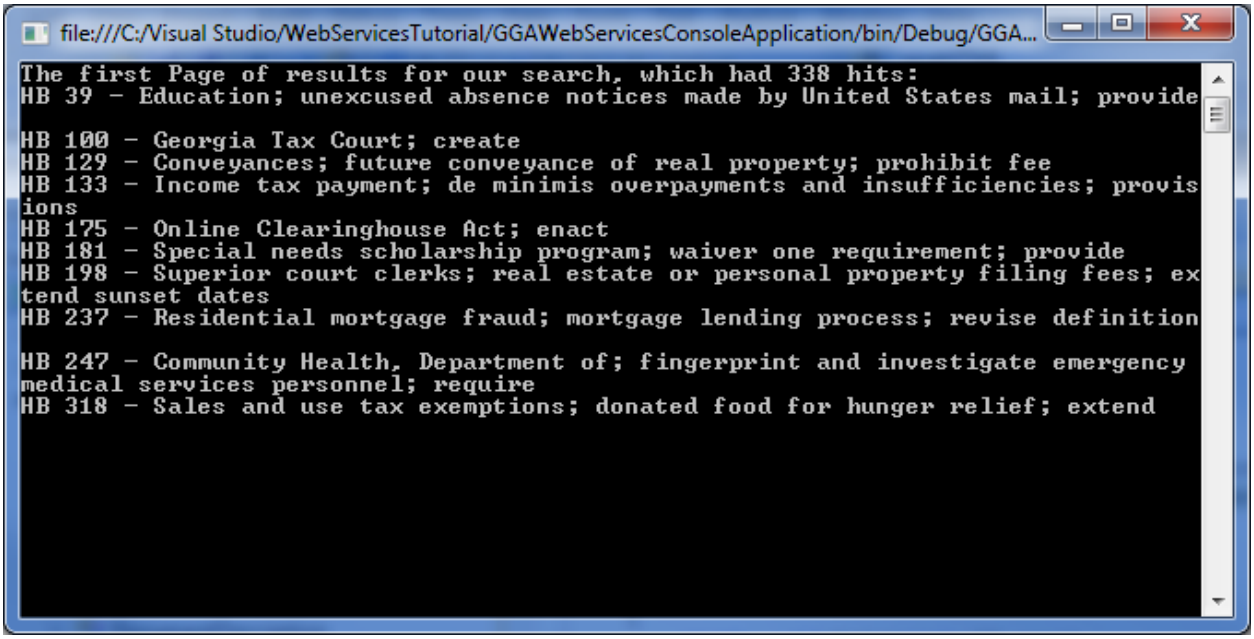
The *default* is Status.Any. This enum will include actions such as House Hopper, Senate Hopper, House Prefiled, and Senate Prefiled. It is quite possible that for your purposes that you wish to exclude these statuses from the result set. This can be useful when trying to find items that had House or Senate Actons on a specific day (Hopper and Prefile are not Floor Actions, they merely note that a bill has been filed (or prefiled) with the Clerk of the House or the Secretary of the Senate.) This can easily be accomplished by setting the Status property to Status.HouseOrSenateAction.

Add

```
statusConstraint.Status = Status.HouseOrSenateAction;
```

before

```
statusConstraint.DateStart = sineDieYear2012.Date;
```



Note that *this* time we have a total of 338 hits, whereas *last* time we had a total of 357 hits.

But wait! I need them *all*, not just the first ten! Let's correct that. We'll start by increasing the Page Size to 250, which is the maximum allowed.

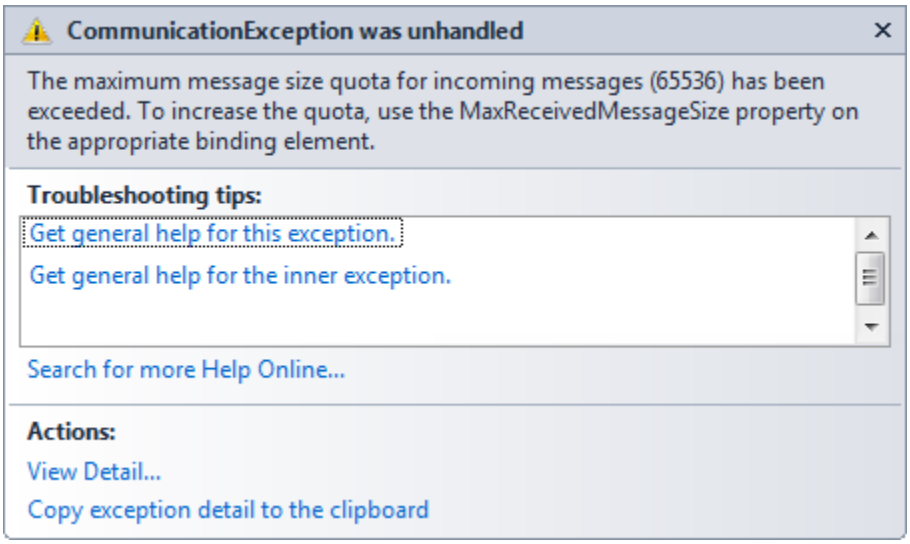
Change

```
int pageSize = 10;
```

to

```
int pageSize = 250;
```

Running the program now produces:



Ouch! This is a security feature provided by Visual Studio when we added our Service Reference. The service returned the data to the client, but the size of the message was bigger than what the client has specified as an acceptable message size. Fortunately, this is easily resolved, although we will have manually edit the XML that was created for us in our app.config file when we added our Service References.

Open the app.config file and find the the <binding> tag that was created for LegislationSearch. Change the values of maxBufferSize and MaxReceivedMessageSize from 65536 to 2147483647 (the max value for int):

```
app.config*
        <bindings>
            <basicHttpBinding>
                <binding name="BasicHttpBinding_SessionFinder" closeTimeout="00:01:00"
                    openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
                    allowCookies="false" bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard"
                    maxBufferSize="65536" maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
                    messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
                    useDefaultWebProxy="true">
                    <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
                        maxBytesPerRead="4096" maxNameTableCharCount="16384" />
                    <security mode="None">
                        <transport clientCredentialType="None" proxyCredentialType="None"
                            realm="" />
                        <message clientCredentialType="UserName" algorithmSuite="Default" />
                    </security>
                </binding>
                <binding name="BasicHttpBinding_LegislationSearch" closeTimeout="00:01:00"
                    openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
                    allowCookies="false" bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard"
                    maxBufferSize="2147483647" maxBufferPoolSize="524288" maxReceivedMessageSize="2147483647"
                    messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
                    useDefaultWebProxy="true">
                    <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
                        maxBytesPerRead="4096" maxNameTableCharCount="16384" />
                    <security mode="None">
                        <transport clientCredentialType="None" proxyCredentialType="None"
                            realm="" />
                        <message clientCredentialType="UserName" algorithmSuite="Default" />
                    </security>
                </binding>
                <binding name="BasicHttpBinding_ILegislationTextService" closeTimeout="00:01:00"
                    openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
                    allowCookies="false" bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard"
                    maxBufferSize="65536" maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
                    messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
                    useDefaultWebProxy="true">
                    <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
                        maxBytesPerRead="4096" maxNameTableCharCount="16384" />
                    <security mode="None">
                        <transport clientCredentialType="None" proxyCredentialType="None"
                            realm="" />
                        <message clientCredentialType="UserName" algorithmSuite="Default" />
                    </security>
                </binding>
                <binding name="BasicHttpBinding_MemberFinder" closeTimeout="00:01:00"
                    openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
                    allowCookies="false" bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard"
                    maxBufferSize="65536" maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
                    messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
                    useDefaultWebProxy="true">
                    <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
                        maxBytesPerRead="4096" maxNameTableCharCount="16384" />
100 %
```

Run the program again to see the following:

```
file:///C:/Visual Studio/WebServicesTutorial/GGAWebServicesConsoleApplication/bin/Debug/GGA...
SB 464 - Fish and Fishing; limit the number of commercial crabbing licenses issu
ed
SB 469 - Labor; provide provisions prohibiting mass picketing shall apply to cer
tain private residences
SB 470 - Patient Self-Referral; revise the definition of "rural area"
SB 480 - Chatham County; change the description of the commissioner districts; d
efinitions and inclusions
SB 481 - Savannah, City of; Chatham County; change the description of the educat
ion districts
SB 483 - Public Service Commission; chairperson; change the term/manner of elect
ion
SB 492 - State Purchasing; require state contract awards for heavy equipment fol
low certain specific procedures
SB 516 - Magistrate Court of Chatham County; provide for the collection of addit
ional costs as law library fees
SB 527 - Turner County; board of education; change the description of the educat
ion districts
SB 528 - Turner County; board of commissioners; change the description of the co
mmissioner districts
SB 529 - Cherokee County Development Authority; provide for expanded powers and
duties
SB 530 - Mansfield, City of; provide a new charter
SB 532 - Doraville, City of; change the corporate limits
SB 533 - Floyd County; provide for nonpartisan elections
```

But we still don't have them all! It said the total was 338, and we only got the first 250. We will fix that now. Find that part of the code that starts with `var results = legislationSearch.GetLegislationSearchResultsPaged(constraints, pageSize, startIndex);` And ends with `legislationSearch.Close();`.

Replace with

```
// get all of the results
List<LegislationSearchResult> allResults = new List<LegislationSearchResult>();
int total = 0;  //assume zero
do
{
	var results = legislationSearch.GetLegislationSearchResultsPaged(constraints,
	pageSize, startIndex);
	total = results.Total;  //this is the actual total for our search
	startIndex += pageSize; //and for the next page we will want to start on the
	record following the last record on this page
	var page = results.Page;
	allResults.AddRange(page);
} while (startIndex < total);
foreach (var result in allResults)
{
	Console.WriteLine(string.Format("{0} {1}{2} - {3}", result.DocumentType,
	result.Number, result.Suffix, result.Caption));
}
Console.WriteLine("Our search had " + total + " hits:");
legislationSearch.Close();
```

Your code window will look like this:

Program.cs

GGAWebServicesConsoleApplication.Program | Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using GGAWebServicesConsoleApplication.SessionService;
using GGAWebServicesConsoleApplication.LegislationService;
namespace GGAWebServicesConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            using (SessionFinderClient sessionService = new SessionFinderClient())
            {
                SessionService.Session thisSession = sessionService.GetSessions().Single(c => c.Id == 21);
                var year = sessionService.GetSessionSchedule(thisSession.Id, Chamber.House).Years.Single(c=>c.Year==2012);
                var sineDieYear2012 = year.Days.SingleOrDefault(c=>c.Number==40);

                int pageSize = 250;
                int startIndex = 0;

                // work with Legislation Service
                LegislationSearchClient legislationSearch = new LegislationService.LegislationSearchClient();
                LegislationSearchConstraints constraints = new LegislationSearchConstraints();

                // we need to create the Session in the LegislationService namespace.  The only property that is important
                // is the Id Property
                constraints.Session = new LegislationService.Session { Id = thisSession.Id };
                StatusConstraint statusConstraint = new ExistingStatusConstraint();
                statusConstraint.Status = Status.HouseOrSenateAction;
                statusConstraint.DateStart = sineDieYear2012.Date;
                statusConstraint.DateEnd = sineDieYear2012.Date;
                constraints.StatusConstraints = new StatusConstraint[] { statusConstraint };

                // get all of the results
                List<LegislationSearchResult> allResults = new List<LegislationSearchResult>();
                int total = 0;  //assume zero
                do
                {
                    var results = legislationSearch.GetLegislationSearchResultsPaged(constraints, pageSize, startIndex);
                    total = results.Total;  //this is the actual total for our search
                    startIndex += pageSize; //and for the next page we will want to start on the record following the last record on this page
                    var page = results.Page;
                    allResults.AddRange(page);
                } while (startIndex < total);
                foreach (var result in allResults)
                {
                    Console.WriteLine(string.Format("{0} {1}{2} - {3}", result.DocumentType, result.Number, result.Suffix, result.Caption));
                }
                Console.WriteLine("Our search had " + total + " hits:");
                legislationSearch.Close();
            }
            Console.ReadLine();
        }
    }
}
```

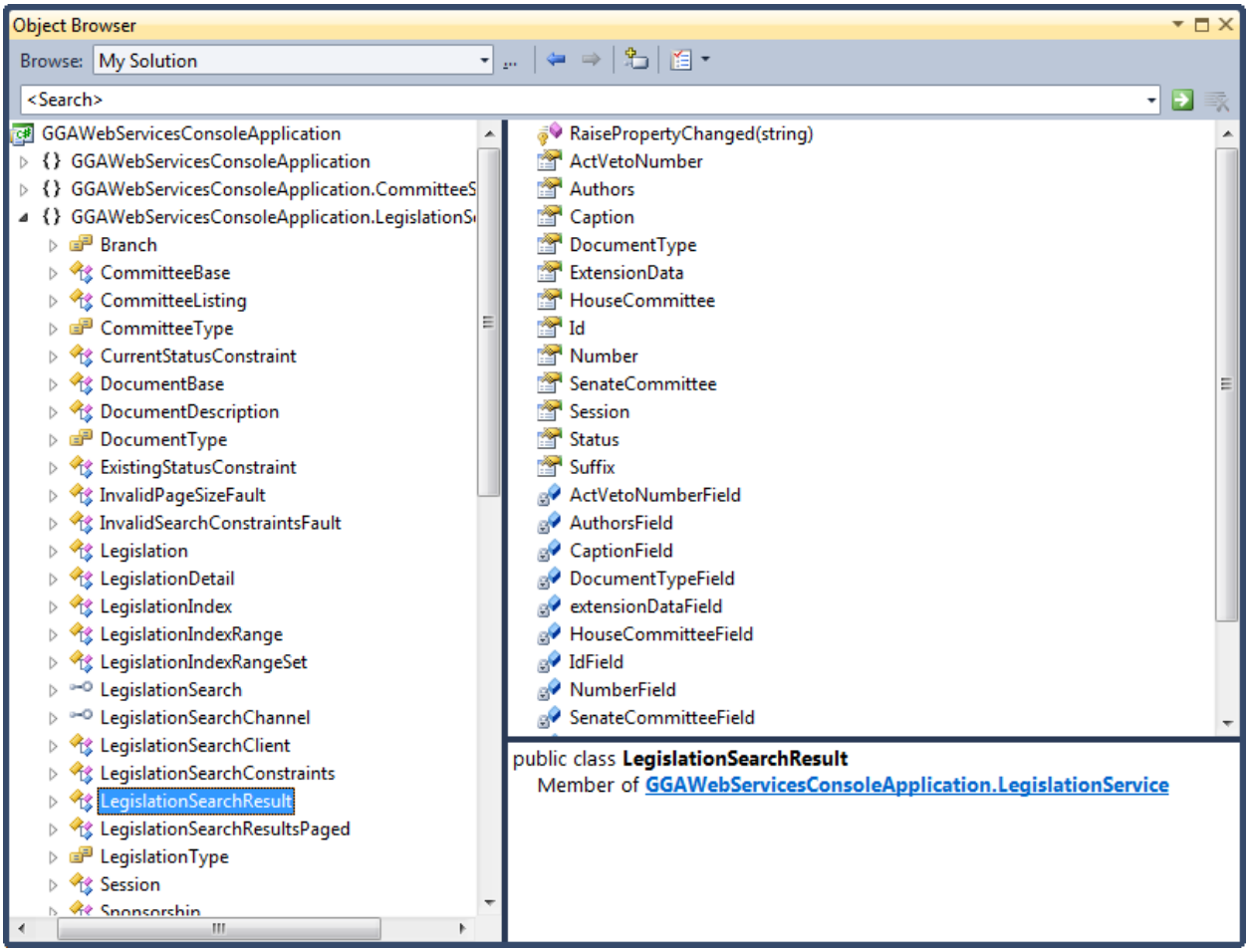
100 %

Running the program returns

file:///C:/Visual Studio/WebServicesTutorial/GGAWebServicesConsoleApplication/bin/Debug/GGA...

```
SR 1349 - Harrison, Timothy Zachary; recognize
SR 1350 - Gunner, Cameron David; recognize
SR 1351 - Green, Ryan Kristopher; recognize
SR 1352 - Gilstrap, David Sanders; recognize
SR 1353 - Farrar, Anthony Todd; recognize
SR 1354 - Cox II, Jerome; recognize
SR 1355 - Carruthers, Kyle Christopher; recognize
SR 1356 - Gunner, III, Larry David; recognize
SR 1357 - Forgay, Mr. Ryan; recognize
SR 1358 - Gary, Jr., William Rodney; recognize
SR 1359 - Clay, Olivia Braxton; recognize
SR 1360 - Carson, Bishop Kenneth Pierre; recognize
SR 1361 - Atlanta Chapter of World Wings International, Inc.; recognize
SR 1362 - Howell, Vikki; recognize
SR 1363 - Johnson, Lil Willie; recognize
SR 1364 - District 35 Student Volunteers and Interns; recognize
SR 1365 - Harmon, Dr. W. Ken; recognize
SR 1366 - Papp, Dr. Daniel S.; recognize
SR 1367 - Grace Lutheran Church; recognize
SR 1368 - Veterans and Community Outreach Foundation; recognize
SR 1369 - Borum, Mr. Bradford R. and Ms. Shannon Leigh Wallace; recognize
SR 1370 - Wallace, Ms. Marcia D.; recognize
SR 1372 - Hooks, Senator George; recognize
Our search had 338 hits:
```

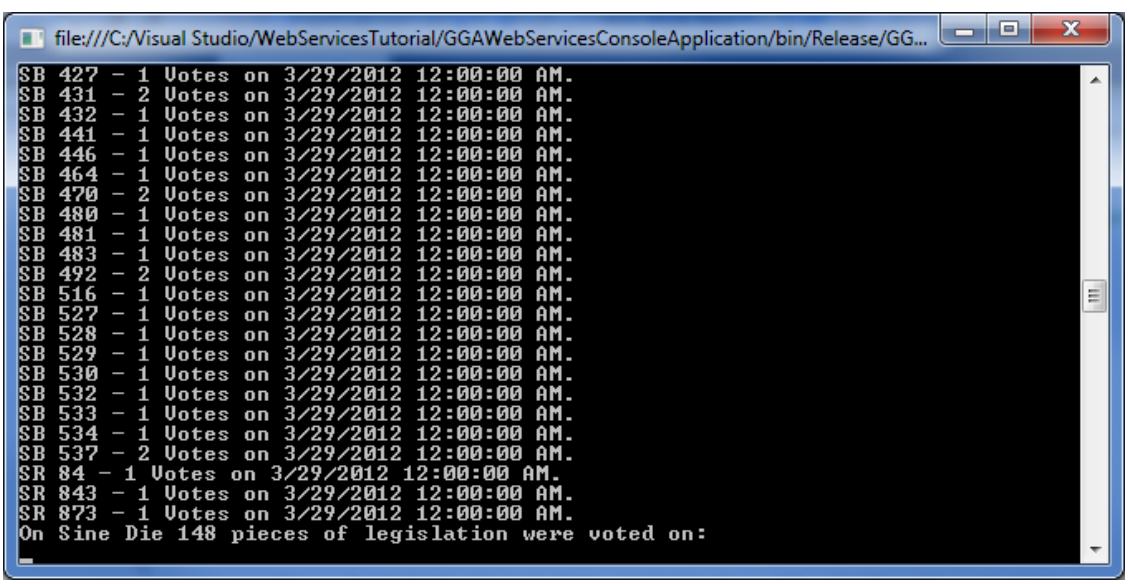
You will notice that LegislationSearchResult contains much of the information about a piece of Legislation, but not all of it.



We have the Authors and the House and Senate Committees, as well as the Current Status, but what about which Versions are available, or the Status History, or what Votes were made? For this we will need the LegislationDetail object. In place of the last block of code that we just wrote, type in the following:

```
// get all of the results
List<LegislationDetail> legislationVotedOn = new List<LegislationDetail>();
int total = 0;  //assume zero
do
{
	var results = legislationSearch.GetLegislationSearchResultsPaged(constraints,
	pageSize, startIndex);
	total = results.Total;  //this is the actual total for our search
	startIndex += pageSize; //and for the next page we will want to start on the
	record following the last record on this page
	foreach (var result in results.Page)
	{
		LegislationDetail details =
		legislationSearch.GetLegislationDetail(result.Id);
		// need to call .Date property of Date to roll back to midnight for
		comparison...
		if (details.Votes != null &&
		details.Votes.Where(c=>c.Date.Date==sineDieYear2012.Date.Date).Count()>0)
		{
			legislationVotedOn.Add(details);
		}
	}
} while (startIndex < total);
foreach (var result in legislationVotedOn)
{
	Console.WriteLine(string.Format("{0} {1}{2} - {3} Votes on {4}.",
	result.DocumentType, result.Number, result.Suffix,
	(result.Votes!=null)?result.Votes.Where(c=>c.Date.Date==sineDieYear2012.Date.Date)
	.Count():0, sineDieYear2012.Date.Date));
}
Console.WriteLine("On Sine Die " + legislationVotedOn.Count + " pieces of legislation
were voted on:");
legislationSearch.Close();
```

The output of our program now looks like:



Ok! Great! Obviously using GetLegislationSearchResultsPaged is a good option:

a) When I want to work with a specific subset of data
b) All of the information that I need is provided by the LegislationSearchResult object.

But, hey this method is pretty slow. What if:

a) I need all of the legislation anyways and
b) Some of the information that I need can only be provided by calling GetLegislationDetail?

Is there a faster way to get the Legislation Ids?

Yes. (and No. There *is* a lightweight method for returning pointers to Legislation, which will be discussed below, but once you have the pointers you will have to call GetLegislationDetail once per item to get the Details. For example, you could create a Form that fills a ListBox with all of the Legislation items, then when the user clicks an item, you could call GetLegislationDetail and display information about the selected item. Getting the Ids will be much faster, but getting *ALL* of the *Details* will be slower than working with GetLegislationSearchResultsPaged and the LegislationSearchResult objects. We recommend that you only call GetLegislationDetail as needed; however, there may be a legitimate need for getting all of the Detail objects for a Session. In this kind of scenario, it is recommended that you restrict your initial search as much as possible by using the LegislationSearchConstraints object. For example, your organization may not be interested in Privileged Resolutions, which make up the bulk of the Resolutions in any given Session. You can pass a Constraint with the LegislationTypes property set like so:

```
constraints.LegislationTypes = new LegislationType[] { LegislationType.GEN,
LegislationType.LOC, LegislationType.CA, LegislationType.NP };
```

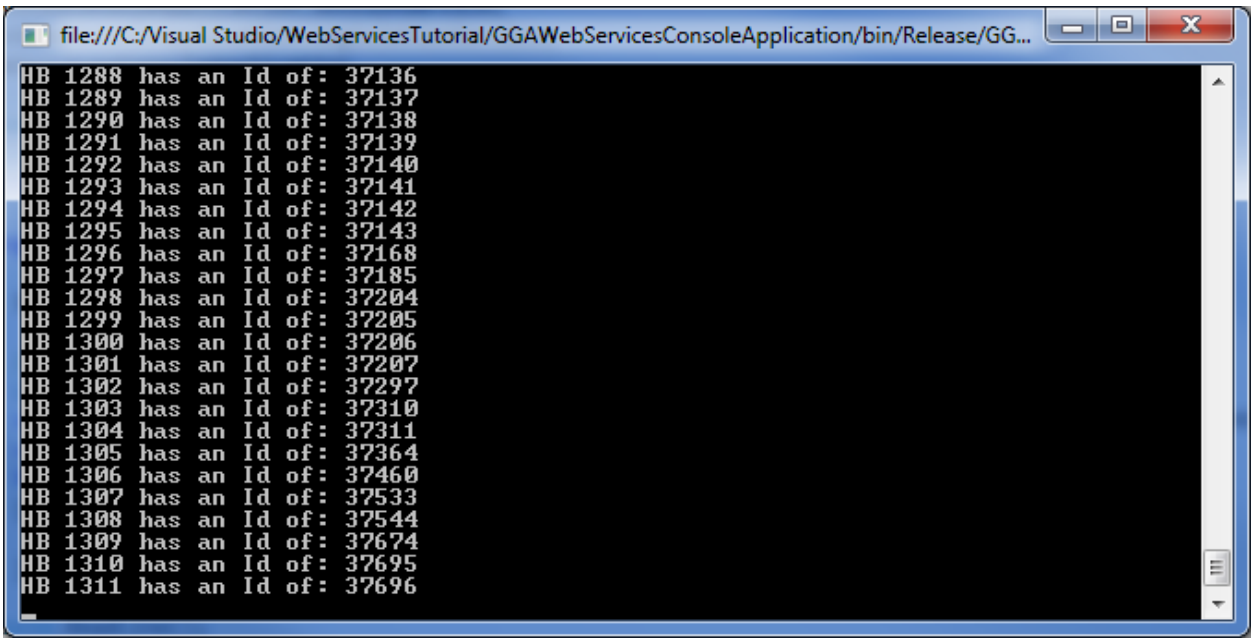
This will return General Bills and Resolutions, Local Bills and Resolutions, Constitutional Amendments, and Non-Privileged Resolutions. Also, you can pass in a StatusConstraint filter as we did previously to limit your results to items that had Status Events within a specific date range. Whenever you *do* need to write a program that loops through a list of Legislation pointers and creates a collection of Detail objects, as we did above and will do again before this tutorial is over, your program will run much faster if you limit the collection to just what you need rather than all that is available.)

So, we were talking about the method to return lightweight objects that have a Legislation's Id, Caption, and Description. Let's do that now. This object is called a LegislationIndex and there are two methods that you can use to obtain a collection of them. One of these methods is GetLegislationForSession() and the other is GetLegislationIndexRange(). We will use GetLegislationForSession().

Let's replace all of code inside of void Main with the code below:

```
using (SessionFinderClient sessionService = new SessionFinderClient())
{
	SessionService.Session thisSession = sessionService.GetSessions().Single(c => c.Id
	== 21);
	// work with Legislation Service
	LegislationSearchClient legislationSearch = new
	LegislationService.LegislationSearchClient();
	var legislationIndex = legislationSearch.GetLegislationForSession(thisSession.Id);
	foreach (var leg in legislationIndex)
	{
		Console.WriteLine(leg.Description + " has an Id of: " + leg.Id);
	}
}
Console.ReadLine();
```

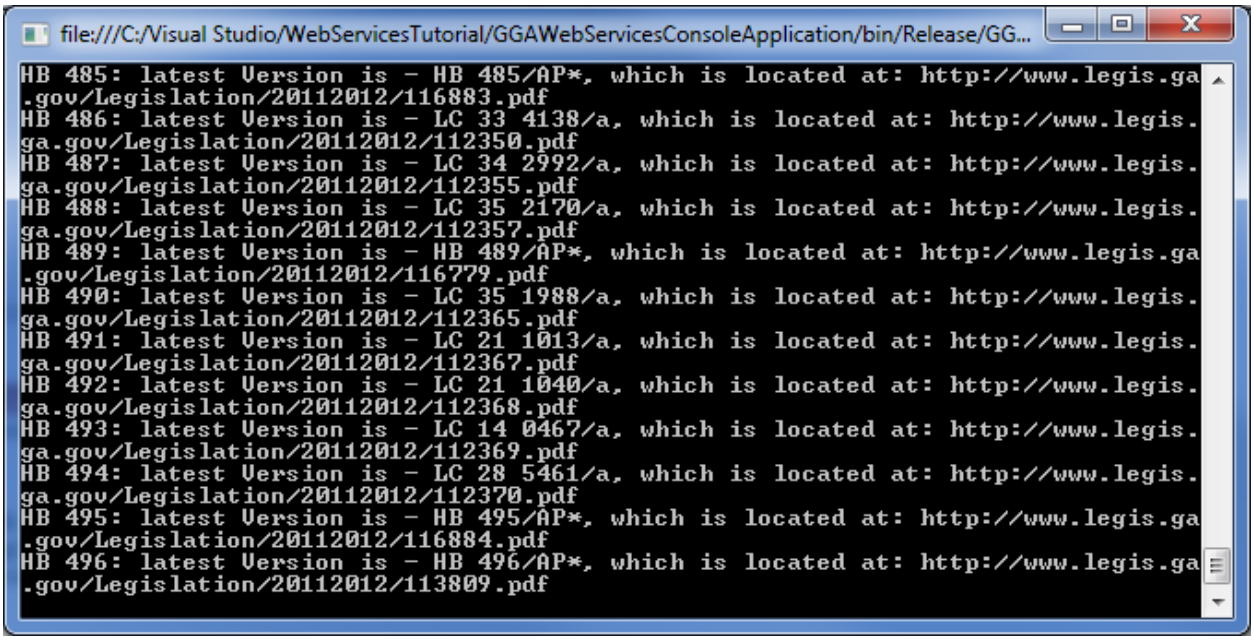
Running the program now produces:



If you need ALL of the information for ALL of the legislation for a Session, call GetLegislationForSession then call GetLegislationDetail inside a loop. We are using the 2011-2012 Regular Session here, so it will take quite some time to complete – there were over 5000 items in this Session. Feel free to change the Session Id to 22 or 23 for the 2011 Special Session or the 2013-2014 Regular Session, respectively, if you want to see the program work but don't want to wait for the results.

```
using (SessionFinderClient sessionService = new SessionFinderClient())
{
        SessionService.Session thisSession = sessionService.GetSessions().Single(c => c.Id
        == 21);
        // work with Legislation Service
        LegislationSearchClient legislationSearch = new
        LegislationService.LegislationSearchClient();
        var legislationIndex = legislationSearch.GetLegislationForSession(thisSession.Id);
        foreach (var leg in legislationIndex)
        {
                LegislationDetail detail = legislationSearch.GetLegislationDetail(leg.Id);
                DocumentDescription latestVersion = detail.Versions.OrderByDescending(c =>
                c.Version).First();
                Console.WriteLine(leg.Description + ": latest Version is - " +
                latestVersion.Description + ", which is located at: " + latestVersion.Url);
        }
}
Console.ReadLine();
```

Running the program now produces:



It will take quite a while for this program to run, as it has to fetch the Detail object for each piece of legislation. As noted earlier, we always recommend that you perform a restrictive search that obtains just the items you need, and then call GetLegislationDetail() as needed.

Bonus Section

(See files included with this tutorial, LegacyBillSummary.xsd and LegacyBillSummary.xslt)

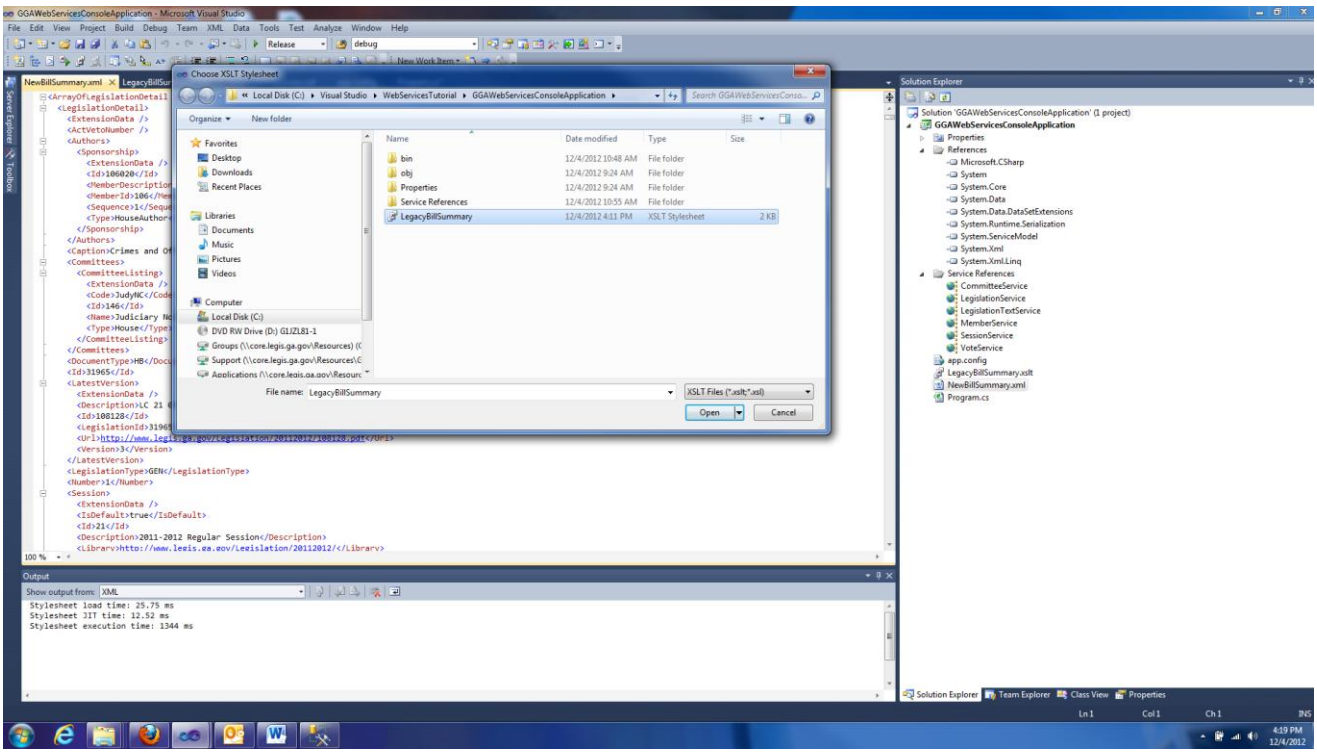
Ok, great! So, using the api that we just used, we could easily construct a Windows Form or an ASP.Net web page that presents a user with short descriptions for each piece of legislation, then when the user selects an item, we can call GetLegislationDetail and display another form or web page that contains the details for that legislation. But what if we already have existing systems centered around the XML schema of the XML File that the General Assembly used to provide *before* Web Services. It might be nice to be able to duplicate this file so that our existing systems can function as is. This is not too awful difficult to do; after all, the objects returned by the Web Services can be serialized to XML, and with a little knowledge of XPath and XSLT, we can write an XSL Stylesheet that converts the *new* XML to the *old* XML schema! We won't provide a complete solution to that problem here, but we will set up the scenario, leaving it to individual organizations to finish if they so choose.

First we will write code that gets *all* of the LegislationDetail Objects for a Session. We are using the 2011-2012 Regular Session here, so it will take quite some time to complete – there were over 5000 items in this Session. Feel free to change the Session Id to 22 or 23 for the 2011 Special Session or the 2013-2014 Regular Session, respectively, if you want to see the program work but don't want to wait for the results.

```
using (SessionFinderClient sessionService = new SessionFinderClient())
{
	SessionService.Session thisSession = sessionService.GetSessions().Single(c => c.Id
	== 21);

	// work with Legislation Service
	LegislationSearchClient legislationSearch = new
	LegislationService.LegislationSearchClient();
	var legislationIndex = legislationSearch.GetLegislationForSession(thisSession.Id);
	List<LegislationDetail> legislationDetails = new List<LegislationDetail>();
	foreach (var leg in legislationIndex)
	{
		LegislationDetail detail = legislationSearch.GetLegislationDetail(leg.Id);
		legislationDetails.Add(detail);
	}
	//work with XML using Linq-to-Xml to serialize our collection of LegislationDetail
	objecet to XML

	System.Xml.Linq.XDocument xml  = new System.Xml.Linq.XDocument();
	using (System.Xml.XmlWriter writer = xml.CreateWriter())
	{
		System.Xml.Serialization.XmlSerializer serializer = new
		System.Xml.Serialization.XmlSerializer(legislationDetails.GetType());
		serializer.Serialize(writer, legislationDetails);
		writer.Close();
	}
	//write the file to the file system.  NOTE: change the path (c:\Visual Studio\) to
	a path on your local file system

	System.IO.FileStream xmlFile = System.IO.File.Create(@"c:\Visual
	Studio\NewBillSummary.xml");
```

```
        byte[] contents = Encoding.UTF8.GetBytes(xml.Root.ToString());
        xmlFile.Write(contents, 0, contents.Length);
        xmlFile.Close();
        Console.WriteLine("Your xml file has been created...");

}
Console.ReadLine();
```

Running this program took ~30 minutes. It produced an XML file with a different schema than the XML file that we used to provide; however, a simple stylesheet can be written to map the data back to the old schema, with some exceptions: **YearID**, **Carryover**, and **CompositeCaption** are not supported by the new Web Services, so we will simply map them to dummy data, such as **-1**. I started an XSLT file to do the transform, and it will be provided with this tutorial. You can use .NET objects to perform the XSL transform programatically, or you can do it manually by opening the file created by the program above (NewBillSummary.xml), clicking on XML menu, Start Xslt (with or without debugging), and selecting LegacyBillSummary.xslt as the stylesheet:



The results of this transform are shown below:

LegacyBillSummary.xml


```xml
<?xml version="1.0" encoding="utf-8"?>
<BillSummary>
  <Bill Id="31965" Type="HB" Num="1" Suffix="" StatusDate="2011-01-24T10:31:46" Carryover="-1" YearID="-1">
    <Number>HB 1</Number>
    <Short_Title>Crimes and Offenses; prenatal murder unlawful; provide</Short_Title>
    <CompositeCaption>This is no longer supported</CompositeCaption>
    <Title>A BILL to be entitled an Act to amend the Official Code of Georgia Annotated so as to provide that prenatal murder shall be unlawful in all events and to remove
numerous references to such procedures; to amend Title 16, relating to crimes and offenses, so as to make certain findings of fact; to define certain terms; to provide that
 any prenatal murder shall be unlawful; to provide a penalty; to repeal certain exceptions to certain offenses; to provide for severability; to provide an effective date;
to repeal conflicting laws; and for other purposes. </Title>
  </Bill>
  <Bill Id="31966" Type="HB" Num="2" Suffix="" StatusDate="2011-01-24T10:31:46" Carryover="-1" YearID="-1">
    <Number>HB 2</Number>
    <Short_Title>Georgia Right to Grow Act; enact</Short_Title>
    <CompositeCaption>This is no longer supported</CompositeCaption>
    <Title>A BILL to be entitled an Act to amend Chapter 1 of Title 2 of the Official Code of Georgia Annotated, relating to general provisions relative to agriculture, so
as to provide a short title; to preempt certain local ordinances relating to production of agricultural or farm products; to protect the right to grow food crops and raise
small animals on private property so long as such crops and animals are used for human consumption by the occupants, gardeners, or raisers and their households and not for
commercial purposes; to define a term; to provide for effect on certain private agreements and causes of action; to provide an effective date; to repeal conflicting laws;
and for other purposes. </Title>
  </Bill>
  <Bill Id="31967" Type="HB" Num="3" Suffix="" StatusDate="2011-01-24T10:31:46" Carryover="-1" YearID="-1">
    <Number>HB 3</Number>
    <Short_Title>Constitutional Tender Act; enact</Short_Title>
    <CompositeCaption>This is no longer supported</CompositeCaption>
    <Title>A BILL to be entitled an Act to amend Title 7 of the Official Code of Georgia Annotated, relating to banking and finance, so as provide a short title; to provide
 legislative findings; to define certain terms; to require any bank or lending institution serving as a depository for the state or any department or  agency of the state
to offer and to accept gold and silver coin for deposit; to amend Title 50 of the Official Code of Georgia Annotated, relating to state government, so as to provide
legislative findings; to define certain terms; to require the exclusive use of gold and silver coin as tender in payment of debts by or to the state; to provide for related
 matters; to provide an effective date; to repeal conflicting laws; and for other purposes. </Title>
  </Bill>
  <Bill Id="31968" Type="HB" Num="4" Suffix="" StatusDate="2011-01-24T10:31:46" Carryover="-1" YearID="-1">
    <Number>HB 4</Number>
    <Short_Title>Life, Liberty, and Property Restoration Act; enact</Short_Title>
    <CompositeCaption>This is no longer supported</CompositeCaption>
    <Title>A BILL to be entitled an Act to amend Title 28 of the Official Code of Georgia Annotated, relating to the General Assembly, so as to create the Joint Committee
on Repeals; to provide a short title; to provide legislative findings; to provide for membership; to provide for duties; to repeal conflicting laws; and for other purposes.
 </Title>
  </Bill>
  <Bill Id="31969" Type="HB" Num="5" Suffix="" StatusDate="2011-01-24T10:31:46" Carryover="-1" YearID="-1">
```


100 %

The old schema will also be attached to this tutorial so you will know what elements and attributes you need to implement in your stylesheet.

Well, we haven't covered everything the Web Services have to offer, but with the knowledge you have gained in this tutorial, you should be able add Service References to your own projects and use the Object Browser to examine the proxy classes that were generated for you. We tried to cover most of the peculiarities and common errors, such as having to construct a Session object in the LegislationService namespace when creating a Constraints object and overcoming the MaxReceivedMessageSize limitation. You may wish to set this value to a number smaller than the one I chose here – I simply chose the Int.MaxValue to insure that we can always accept the return message.

Happy Programming!

Georgia General Assembly I.T.