OGSI.NET
UVa Grid Computing Group

OGSI.NET Developer Tutorial

Table of Contents

Introduction

This document is intended to guide developers, new to the OGSI.NET project, through the process of creating grid services. We assume that you have a working understanding and familiarity with the Visual Studio .NET environment and with grid services in general.

Throughout this tutorial, we will follow a "code first" programming model for grid services. What we mean by this is that we will first describe how to write the backend service code (code that is more or less agnostic of grid services and OGSI.NET). After this backend code has been written, we will then go back and annotate the code with OGSI.NET specific C# attributes. These attributes indicate various parameters relevant to the OGSI.NET system and include those for naming, identification, security policy, and serialization (among others). At the core of its programming model, OGSI.NET is an attribute based approach to grid service programming.

Important

Before beginning this tutorial you MUST set two environment variables which will point to the bin directory for your OGSI.NET server install and to the schema directory for OGSI.NET and this must be done prior to starting Visual Studio. Refer to the windows online help for information on setting environment variables.

The first environment variable is called **OGSIDOTNETSCHEMA** and must point to the schema directory for OGSI.NET. This directory is determined by your IIS configuration, but usually the value will be "**C:\inetpub\wwwroot\Schema**".

The second environment variable that you need to set is the **OGSIDOTNETSERVER** variable and this must point to the Bin directory under your OGSI.NET server installation. If you selected the default location when you installed OGSI.NET server, then this value should be
"**C:\Program Files\UVAGCG\OGSIdotNetServer**"

Writing a Simple Service

This chapter will explain in detail how to write a simple OGSI.NET service. The examples given here highlight the basic concepts involved in writing OGSI.NET. Further details and more advanced options can be found in the OGSI.NET Programmer's Reference.

Simple Math Port Type

The example that we will use throughout this tutorial to illustrate the steps necessary to create OGSI.NET services is that of a Math service. For this first chapter, we will implement an *add* and a *subtract* method and following chapters will expand on this concept.

To begin with, we'll implement the bare-bones Math service port type without any attributes that would mark it as a grid service. Recall that a grid service implements some number of port types and each port type implements some number of methods. For our Simple Math Service, we'll start by implementing methods for the SimpleMathPortType. Start by creating a new Visual Studio .Net blank solution. Once the solution has been created, start a new "Class Library" C# Project which I will call **SimpleMathService** (add a new project to your solution). Next, change the name of the Class1.cs file that was created for you to SimpleMathPortType.cs.

Open the SimpleMathPortType.cs file and edit it so that it contains the source given in Figure 1.

```
using System;

namespace SimpleMathService
{
   public class SimpleMathPortType
   {
      public int add(int one, int two)
      {
         return one + two;
      }

      public int subtract(int one, int two)
      {
         return one - two;
      }
   }
}
```

Figure 1: SimpleMathPortType without Annotations

While no client code yet exists with which to call this service, you should still be able to compile the project without any difficulties. Go ahead and do so to ensure that there are no typing errors so far.

Now that you have the simple skeleton code written, its time to start annotating the class and its methods with attributes. These attributes indicate to the OGSI.NET container how the code should be exposed as a grid service (what methods are exposed, under what policy, etc.). For the **SimpleMathPortType**, we'll add some

attributes that are needed to generate wsdl for the port type (see below). Add the following attributes to the **SimpleMathPortType** class (see Figure 2 and Figure 3).

```
[WsdlBaseName("SimpleMath",
   "http://gcg.cs.virginia.edu/simple-math")]
[WebServiceBinding]
```

Figure 2: Initial Class Attributes for SimpleMathPortType

These two attributes (**WsdlBaseName** and **WebServiceBinding**) tell OGSI.NET that this is a new grid service and provides a naming scheme to use when generating the service's wsdl. You also need to derive the class off of OGSI.NET's base grid service class, called **GridServiceSkeleton**. In addition, some of these new types require assemblies that are not currently referenced by your project. Add the following assemblies to your project:

- System.Web.Services.dll
- UVa.Grid.WsdlDocuments.dll
- UVa.Grid.OGSIGridServices.dll

The finished product should look like the code shown in Figure 3.

```
using System;
using System.Web.Services;

using UVa.Grid.WsdlDocuments;
using UVa.Grid.OGSIGridServices;

namespace SimpleMathService
{
   [WsdlBaseName("SimpleMath",
      "http://gcg.cs.virginia.edu/simple-math")]
   [WebServiceBinding]
   public class SimpleMathPortType : GridServiceSkeleton
   {
      public SimpleMathPortType()
      {
      }

      public int add(int one, int two)
      {
         return one + two;
      }

      public int subtract(int one, int two)
      {
         return one - two;
      }
   }
}
```

Figure 3: SimpleMathPortType with Initial Attribute Annotations

In Figure 3 we added the WsdlBaseName attribute with two parameters – "SimpleMath" and "http://gcg.cs.virginia.edu/simple-math". The first of these parameters gives a "base name" from which a number of wsdl elements and generated

source code names are derived. The second parameter is a namespace which is also used in the generated wsdl.

Next, we need to identify which methods in our **SimpleMathPortType** are to be exported by the grid service. This is done by placing the **WebMethod** attribute on each method that we wish to export (see Figure 4).

```
[WebMethod]
public int add(int one, int two)
…

[WebMethod]
public int subtract(int one, int two)
…
```

Figure 4: WebMethod Annotations on our Business Methods

Once again, try to compile the project and make sure there are no errors. At this point, you have very nearly completed the **SimpleMathPortType**. The only remaining task is to annotate the web methods with the names of the return values[1]. This tells OGSI.NET that when this function returns a value, that value should be wrapped in an XML element with the indicated name.

On each of the *add* and *subtract* methods, add the **XmlElement** attribute to the return values as shown in Figure 5. You will also need to add the System.Xml.dll assembly to your project and should add a statement indicating that you will be using the **System.Xml.Serialization** namespace.

[1] This step will not be necessary in version 2.1 of the OGSI.NET system.

```
using System;
using System.Web.Services;
using System.Xml.Serialization;

using UVa.Grid.WsdlDocuments;
using UVa.Grid.OGSIGridServices;

namespace SimpleMathService
{
   [WsdlBaseName("SimpleMath",
      "http://gcg.cs.virginia.edu/simple-math")]
   [WebServiceBinding]
   public class SimpleMathPortType : GridServiceSkeleton
   {
      public SimpleMathPortType()
      {
      }

      [WebMethod]
      [return: XmlElement("sumResult")]
      public int add(int one, int two)
      {
         return one + two;
      }

      [WebMethod]
      [return: XmlElement("differenceResult")]
      public int subtract(int one, int two)
      {
         return one - two;
      }
   }
}
```

Figure 5: Fully Annotated Version of the SimpleMathPortType

You should once again be able to compile the port type at this point. You have now completed all of the tasks necessary to create the port type. In the next section, we will write the service class that exposes this port type and generate the wsdl file that goes along with your new service.

Simple Math Service and Bindings

The next step in this process is to write a service class that exposes the **SimpleMathPortType** class. In the OGSI specification, port types provide for a logical grouping of related methods. The **add** and **subtract** methods that you wrote are part of a new port type that you created called the **SimpleMathPortType**. However, a port type is not in and of itself a grid service. A grid service is a logical grouping of port types in a single container. At a minimum, any new service must contain at least the **GridServicePortType**, but usually you will also include other port types as well (for example, a port type for the new grid service methods that you wish to expose). For your Simple Math Service, you need to create a new class which will group the **SimpleMathPortType** port type with the **GridServicePortType** port type. In your **SimpleMathService** project, create a new C# class called **SimpleMathService** (call the new file SimpleMathService.cs). Open this file in the editor and modify the code as shown below:

```
using System;

namespace SimpleMathService
{
   public class SimpleMathService
   {
      public SimpleMathService()
      {
      }
   }
}
```

Figure 6: Initial Implementation of the SimpleMathService Class

Once you ensure that this new code compiles, you will need to add some attributes to:

1. Identify this service as a web service with the **WebService** attribute. Recall that all grid services are web services.
2. Add naming information for the **WsdlGenerator** with the **WsdlBaseName** attribute. As before, the name and namespace parameters given here are used in generating a services wsdl (see below).

```
using System;
using System.Web.Services;

using UVa.Grid.WsdlDocuments;
using UVa.Grid.OGSIGridServices;

namespace SimpleMathService
{
   [WsdlBaseName("SimpleMath",
      "http://gcg.cs.virginia.edu/simple-math")]
   [WebService]
   public class SimpleMathService : GridServiceSkeleton
   {
      public SimpleMathService()
      {
      }
   }
}
```

Figure 7: WsdlBaseName and WebService Attributes on SimpleMathService

Next, you need to identify which port types this service implements. Obviously, the **SimpleMathPortType** is implemented here, but our service also implements the **GridServicePortType** because all OGSI compliant grid services must implement the **GridServicePortType**. We also derive the **SimpleMathService** class off of the **GridServiceSkeleton** class which contains much of the **GridServicePortType** code.

To specify the port types that this service is implementing, we use the **OGSIPortType** attribute contained in the **UVa.Grid.WsdlDocuments** namespace. The **GridServicePortType** class (which we will use in our code as well) is contained

in the **UVa.Grid.OGSICoreServices.GridService** namespace and in order to access that namespace, you will have to add the UVa.Grid.OGSICoreServices.dll assembly to your project. The new code is given in Figure 8. Once you have those changes implemented, compile your project.

```
using System;
using System.Web.Services;

using UVa.Grid.WsdlDocuments;
using UVa.Grid.OGSIGridServices;
using UVa.Grid.OGSICoreServices.GridService;

namespace SimpleMathService
{
   [WsdlBaseName("SimpleMath",
      "http://gcg.cs.virginia.edu/simple-math")]
   [WebService]
   [OGSIPortType(typeof(SimpleMathPortType))]
   [OGSIPortType(typeof(GridServicePortTYpe))]
   public class SimpleMathService : GridServiceSkeleton
   {
      public SimpleMathService()
      {
      }
   }
}
```

Figure 8: Fully Annotated Version of SimpleMathService

Now that we have a working, compiled service class (and assembly), it is time to put that assembly and service into the OGSI.NET Container. To do this, the generated DLL will have to be copied to the %OGSIDOTNETSERVER%\Bin directory. We will also have to generate the wsdl that goes along with the new service and place that wsdl into the appropriate schema directory in OGSI.NET.

Any schema directory can be used (provided that it shows up under the %OGSIDOTNETSCHEMA% directory), but for the purposes of this tutorial, we'll assume that we are placing our service in %OGSIDOTNETSCHEMA%\tutorial\math-service.

Important Note

Make sure that the schema directory for the Simple Math Service (%OGSIDOTNETSCHEMA%\tutorial\math-service) exists and is writeable on your machine.

Now you need to create a post-build step for the **SimpleMathService** project that will move the new assembly to the %OGSIDOTNETSERVER%\Bin directory and generate the service's wsdl files. If you are unfamiliar with creating post-build steps in Visual Studio .Net for projects, please refer to Appendix A.

The post build step that we wish to have for this project should contain the following text:

```
move /Y "$(TargetPath)" "%OGSIDOTNETSERVER%\Bin\$(TargetFileName)"
cd "%OGSIDOTNETSCHEMA%\tutorial\math-service\"
del /F /Q "*.wsdl"
"%OGSIDOTNETSERVER%\Bin\WsdlGenerator.exe" -a ↺
   "%OGSIDOTNETSERVER%\Bin\$(TargetFileName)" -c ↺
   SimpleMathService.SimpleMathService
```

Figure 9: Post-build Step for the SimpleMathService Project

Note

The use of the ↺ character in the code block above is meant to indicate a line continuation. Lines ending with this symbol should not in-fact have a carriage return at the end but rather the text following should continue without a newline.

Once again, build your **SimpleMathService** project[2]. This time, your post-build step should be executed at the end of the compilation phase. You may get an error message displayed indicated that the build process was unable to delete *.wsdl, but this error message is non-fatal and can be ignored. It only occurs when you have not yet generated any wsdl files in the target schema directory.

At this point you should be able to browse to the %OGSIDOTNETSERVER%\Bin directory and verify that the SimpleMathService.dll assembly has shown up there. Likewise, the %OGSIDOTNETSCHEMA%\tutorial\math-service directory should now contain wsdl files that have been generated for your service.

Starting the SimpleMathService OGSI.NET Service

Next, we'll configure the OGSI.NET to contain the Simple Math Service and finally we'll start the **OGSIContainerService** process. The configuration information for OGSI.NET is contained in the %OGSIDOTNETSERVER%\Configuration\OGSI.config. Open this file in any editor you choose. Inside, you will find an XML document containing a root XML document called **deployment** with a number of contained elements called **service**. You need to add a new service element which describes the Simple Math Service (shown in Figure 10). The format and meaning of the XML elements contained in this file is described in more detail in the OGSI.NET Programmer's Reference.

[2] Because this build step now involves over writing DLL's in the OGSI.NET Bin directory, it's possible at various points in the future that this step may fail if you do not stop the OGSIContainerService windows service. See Appendix B for a description of how to do this.

```
<service name="tutorial/math-service/SimpleMathService">
   <parameter name="schemaPath"
      value="schema/tutorial/math-service/SimpleMathService.wsdl"/>
   <parameter name="class" value=
      "SimpleMathService.SimpleMathService, SimpleMathService.dll"/>
   <parameter name="messageHandlers" value=
      "UVa.Grid.SoapMessageHandler.SoapMessageHandler, ↺
       UVa.Grid.SoapMessageHandler.dll; ↺
       UVa.Grid.RemotingMessageHandler.RemotingMessageHandler, ↺
       UVa.Grid.RemotingMessageHandler.dll"/>
</service>
```

Figure 10: New Service Element in the OGSI.config File

After you have finished modifying the configuration file, save it and open your Windows Services program (Refer to Appendix B if you need help with this) and start/restart the *OGSIContainerService*. The final activity for this chapter is to create a client driver which can talk to this service and test its functionality.

Creating the SimpleMathClient

Inside your Visual Studio Solution browser, add a new "Console Application" project to your solution and call this project **SimpleMathClient**. As before, change the name of the default Class1.cs file, this time to SimpleMathClient.cs.

Assuming that you have installed the AddGridReference tool provided with OGSI.NET, creating and adding a stub file for your service is relatively easy. Right-click on the **SimpleMathClient** project inside the Solution Browser and select "Add Grid Reference" from the pop-up menu which appears. You will be prompted for the name of a grid service for which you wish to add a reference. In the space provided, you need to indicate the name of the service which you wish to communicate with (see Figure 11). Based on the configuration information that you wrote into the OGSI.config file in the previous section, the name of the Simple Math Service should be "**http://localhost/OGSA/Services/tutorial/math-service/SimpleMathService**". After a few seconds, you should see that your project has been updated to contain a new source file called SimpleMathServiceProxy.cs as well as several new assembly references. The SimpleMathServiceProxy.cs file contains client stub classes which you can use to communicate with the Simple Math Service. However, before you can compile this new code, you will need to add assembly references to the following files:

- Microsoft.Web.Services.dll
- System.Web.Services.dll
- UVa.Grid.OGSICoreServices.Bindings.dll
- UVa.Grid.WsdlDocuments.dll

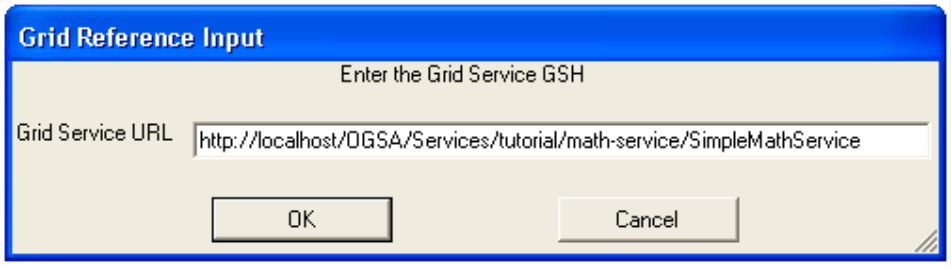

Figure 11: Adding a Grid Reference

You should now compile the client project and make sure that everything is correct so far (in particular, verify that the generated binding code compiles).

Next, you have to fill in the client code for the **SimpleMathClient**. Because this code is fairly self explanatory, it is provided in its entirety below. Please open the SimpleMathClient.cs file and modify it to match the code in Figure 12.

```
using System;
using SimpleMathService;

namespace SimpleMathClient
{
   class SimpleMathClient
   {
      [STAThread]
      static void Main(string []args)
      {
         if (args.Length < 3)
         {
            Console.WriteLine(
               "USAGE:  SimpleMathClient "
               + "<service-url> <first-num> <second-num>");
            return;
         }

         int one = Int32.Parse(args[1]);
         int two = Int32.Parse(args[2]);
         int result;

         SimpleMathBinding binding =
            new SimpleMathBinding(args[0]);

         result = binding.add(one, two);
         Console.WriteLine("{0} + {1} = {2}", one, two, result);

         result = binding.subtract(one, two);
         Console.WriteLine("{0} - {1} = {2}", one, two, result);
      }
   }
}
```

Figure 12: Entire SimpleMathClient Source Code

The client code provided above parses the command line arguments, creates the client stub, and then calls the two business methods on our service and displays the results. For this example, the program will take exactly 3 arguments. The first is

the name of the OGSI service that the client connects to ("http://localhost/OGSA/Services/tutorial/math-service/SimpleMathService" for your service) and the last two provide integers which the driver uses to test the add and subtract methods. Notice that we create a **SimpleMathBinding** instance and pass the OGSI service name in to its constructor. This step creates a proxy client which your driver uses to communicate with your service. The constructor parameter indicates which service that proxy will communicate with. Under the covers, the binding stores the grid service handle that you constructed the object with and stores it until the first out call is made from the binding. When that happens, the binding will take care of resolving that handle to a grid service reference and then caching the reference for future communication. This is the main way in which "Add Grid Reference" differs from "Add Web Reference".

Running the Math Service Example

To test the client and service code, compile the SimpleMathClient project and bring up a command line prompt. Change directories to the location of your SimpleMathClient binary (this should be somewhere under the bin sub-directory inside the location you gave for your project when you created it). Finally, run the driver with reasonable command line arguments (see Figure 13).

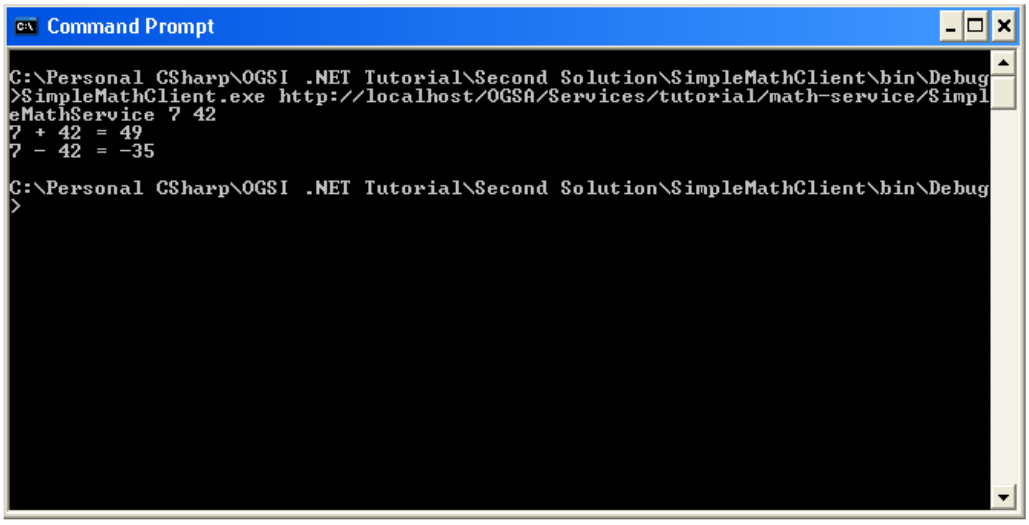

Figure 13: Simple Math Client Running

OGSI.NET Services as One Class

While writing separate port type and service wrapper classes can be useful when many different services need to access your service, sometimes it is desirable to collapse these two distinct code bases into a single C# service class. By annotating the service class with appropriate attributes, OGSI.NET will create the correct port type for the methods indicated.

Modifying SimpleMathService

We will now modify the **SimpleMathService** project so that it contains only a single C# class rather then both a service class and a port type class. The first step is to remove the SimpleMathService.cs file from the **SimpleMathService** project. Next, rename the SimpleMathPortType.cs file to SimpleMathService.cs and open it in the editor. Inside this file, replace each instance of the word **SimpleMathPortType** with the text **SimpleMathService**. Next, add the **OGSIPortType** attribute for **GridServicePortType** as well as the **WebService** attribute (as shown below) to the class attributes. These attributes serve the same function that they did before on your service class. You will also need to add a using statement for the **UVa.Grid.OGSICoreServices.GridServices** namespace. Once you have made these changes, recompile your service project (once again, you need to stop the OGSIContainerService during this step).

```
using System;

using System.Web.Services;
using System.Xml.Serialization;

using UVa.Grid.WsdlDocuments;
using UVa.Grid.OGSIGridServices;
using UVa.Grid.OGSICoreServices.GridService;

namespace SimpleMathService
{
   [WsdlBaseName("SimpleMath",
      "http://gcg.cs.virginia.edu/simple-math")]
   [WebServiceBinding]
   [WebService]
   [OGSIPortType(typeof(GridServicePortType))]
   public class SimpleMathService : GridServiceSkeleton
   {
      public SimpleMathService()
      {
      }

      [WebMethod]
      [return: XmlElement("sumResult")]
      public int add(int one, int two)
      {
         return one + two;
      }

      [WebMethod]
      [return: XmlElement("differenceResult")]
      public int subtract(int one, int two)
      {
         return one - two;
      }
   }
}
```

Figure 14: Modified SimpleMathService File

The SimpleMathClient code doesn't need to be changed to match the changes that you made to the Simple Math Service. However, because the changes affect the wsdl that gets generated for your service, you will have to recreate the proxy stubs. You can do so by re-adding the grid reference for SimpleMathService to your project. Once again, right-click on your **SimpleMathClient** project and select "**Add Grid Reference**" from the resulting pop-up menu. As before, when prompted for the name of the service to which you wish to add a grid reference, enter the name of your **SimpleMathService** service ("*http://localhost/OGSA/Services/tutorial/math-service/SimpleMathService")*. You will be prompted whether or not it is OK to replace the existing proxy client file and can safely accept this dialog. Finally, recompile your client project and once again run the client to verify that your service and client can communicate.

SDEs and Grid Services

Sometimes it's desirable to have your grid services export public data that clients can access and manipulate. In the OGSI specification, the interface to do this is through SDEs (Service Data Elements). OGSI.NET supports SDEs and the following chapter will guide you through the steps of specifying SDEs in your Simple Math Service. We will also modify your client driver so that it can retrieve and modify those values.

Adding SDEs to Service Code

First, you need to add an SDE to your existing **SimpleMathService** code. Open the SimpleMathService.cs file in your editor and add a public data element to the service class (of type **int)** called **_numCalls** (see Figure 15). You should also add a constructor to the class that initializes this value to 0. For this example, we will use this public data item to count the number of calls that have been made to the **add** and **subtract** methods during the service's lifetime. Note that this data member is public in the **SimpleMathService** class. This is because all SDEs which show up as data members in an OGSI.NET service class are required to have public access.

```
…
   public class SimpleMathService : GridServiceSkeleton
   {
      public int _numCalls;

      public SimpleMathService()
      {
         _numCalls = 0;
      }
…
```

Figure 15: Starting to Count Method Calls in SimpleMathService

Next, we need to add code to the **add** and **subtract** methods to increment this data member whenever they are called. The code fragment below shows how this might be done.

```
…
      [WebMethod]
      [return: XmlElement("sumResult")]
      public int add(int one, int two)
      {
         _numCalls++;
         return one + two;
      }

      [WebMethod]
      [return: XmlElement("differenceResult")]
      public int subtract(int one, int two)
      {
         _numCalls++;
         return one - two;
      }
…
```

Figure 16: Using the _numCalls Data Member to Count Method Calls

Finally, you have to tell the OGSI.NET system that this public data member is in fact an SDE (analogous to annotating the **add** and **subtract** methods with the WebMethod attribute). To do this, add an attribute called **SDE** to the **_numCalls** data member (see Figure 17). Once you have done that, stop the **OGSIContainerService** service, recompile your service code and restart the **OGSIContainerService** service as before.

```
…
   public class SimpleMathService : GridServiceSkeleton
   {
      [SDE]
      public int _numCalls;

      public SimpleMathService()
      {
         _numCalls = 0;
      }
…
```

Figure 17: Annotating _numCalls with the SDE Attribute

Retrieving an SDE from Client Code

Now, we will modify the SimpleMathClient.cs file to retrieve the _numCalls SDE that you just created. SDE's can be retrieved through the **GridServicePort** binding that is included with every client stub file (called **GridServiceSOAPBinding**). When you query for an SDE with this class, the result that you get back is in the form of an XML document so it is necessary to implement some minimal amount of XML parsing when reading SDE values. Fortunately, the **XmlSerializer** class included with C# provides a useful interface for accomplishing this task.

First, you need to retrieve a new version of the client stub file. As before, right click on your **SimpleMathClient** project and re-add a grid reference to your service. Next, add a new private, static method to your **SimpleMathClient** class

called **getNumCalls** which you will use to retrieve this value. Figure 18 shows the code for **getNumCalls** needed to retrieve the **_numCalls** SDE and parse the resultant XML. Understanding this XML is beyond the scope of this document and will be covered in greater detail in future manuals.

```
…
   class SimpleMathClient
   {
      static private int getNumCalls(GridServiceSOAPBinding gbinding)
      {
         QueryByServiceDataNames query =
            new QueryByServiceDataNames();
         query.name = new XmlQualifiedName[]
            {
               new XmlQualifiedName("_numCalls", "")
            };
         XmlElement xmlFormattedQuery =
            OgsiInterface.SerializeObject(query,
               typeof(QueryByServiceDataNames), null);
         XmlElement xmlFormattedReturn = gbinding.findServiceData(
            xmlFormattedQuery);
         XmlSerializer serializer = new XmlSerializer(typeof(int));
         return (int)serializer.Deserialize(
            new XmlNodeReader(xmlFormattedReturn.FirstChild));
      }

      [STAThread]
      static void Main(string []args)
      {
         if (args.Length < 3)
         {
…
```

Figure 18: Writing Code to Retrieve SDE Values from a Service

The code given above references various classes from a number of difference namespaces. Before you can compile your client, you will need to add using statements as shown in Figure 19. You will also have to add an assembly reference to your project for UVa.Grid.OGSIInterface.dll which unlike the other assembly references so far, is located in the client install directory rather then the server install directory.

```
using System;

using System.Xml;
using System.Xml.Serialization;
using System.Web.Services.Protocols;

using UVa.Grid.OGSICoreServices.Bindings.GridService;
using UVa.Grid.OGSIInterface;

namespace SimpleMathClient
{
   class SimpleMathClient
   {
…
```

Figure 19: Additional Namespaces Used by the new SimpleMathClient Source

Finally, you need to add calls to this **getNumCalls** method at appropriate and interesting points in your driver code. Recall that the **getNumCalls** method takes a **GridServiceSOAPBinding** instance as its single parameter. This binding is located inside the service proxy file that also contains your **SimpleMathService** proxy code and serves a similar function to that of the **SimpleMathBinding** class. In this case, rather then giving you access to your Simple Math Service methods, it instead allows you to make calls that are part of the Grid Service Port Type. Among these Grid Service Port Type methods are the methods you use to retrieve and modify SDE values. Once you have modified your client code (as shown below) you can recompile your client project and once again test your service by running the resulting binary.

```
…
        int one = Int32.Parse(args[1]);
        int two = Int32.Parse(args[2]);
        int result;

        SimpleMathBinding binding =
           new SimpleMathBinding(args[0]);
        GridServiceSOAPBinding gbinding =
           New GridServiceSOAPBinding(arg[0]);

        Console.WriteLIne("Number of calls made:  "
           + getNumCalls(gbinding));
        result = binding.add(one, two);
        Console.WriteLine("{0} + {1} = {2}", one, two, result);

        Console.WriteLIne("Number of calls made:  "
           + getNumCalls(gbinding));
        result = binding.subtract(one, two);
        Console.WriteLine("{0} - {1} = {2}", one, two, result);
        Console.WriteLIne("Number of calls made:  "
           + getNumCalls(gbinding));
     }
…
```

Figure 20: Displaying the Number of Calls that have been made to the Simple Math Service

Changing SDE Values

Because SDEs declared as we did in the previous section (by labeling a public data member with the **SDE** attribute) are not modifiable, we can't actually set the **_numCalls** SDE from the client. If we want to change this value from the outside (perhaps for the purpose of resetting the value to 0 at some point), then we must either implement a service function to do this (a **setNumCalls** method), or we must declare the SDE in the class attributes rather then as a public data member. In this section, we will modify the **_numCalls** SDE code so that it is declared in this way and will change the **SimpleMathClient** driver to reset the number of calls to 0 at some point during the client's execution.

First take the code for the public data member out from the **SimpleMathServer** code (as well as the **SDE** attribute on it). Instead of declaring the

SDE this way, we will declare that **_numCalls** is an SDE on the class by annotating the **SimpleMathService** class with an **SDE** attribute (as shown in Figure 21).

```
using System;

using System.Web.Services;
using System.Xml.Serialization;
using System.Xml;
using System.Collections;

using UVa.Grid.WsdlDocuments;
using UVa.Grid.OGSIGridServices;
using UVa.Grid.OGSICoreServices.GridService;
using UVa.Grid.WsdlDocuments.GridForum;

namespace SimpleMathService
{
   [WsdlBaseName("SimpleMath",
      "http://gcg.cs.virginia.edu/simple-math")]
   [WebServiceBinding]
   [WebService]
   [OGSIPortType(typeof(GridServicePortType))]
   [SDE("_numCalls", typeof(int), false,
         ServiceDataTypeMutability.mutable, true, "1", "1")]
   public class SimpleMathService : GridServiceSkeleton
   {
```

Figure 21: Declaring an SDE as a Class Annotation Rather then as a Data Member

Notice that in this form, the SDE attribute takes a large number of parameters. The exact meaning of these parameters is given in detail in the OGSI.NET Programmer's Reference. For this example, we are indicating the name of the SDE, its type, that the value cannot be nil, that it is mutable (as defined by the OGSI specification), that it is modifiable, and that it occurs exactly once in any instance of the Simple Math Service.

We also need to modify all accesses to this data from within the service class code. Previously we could modify the **_numCalls** value by directly accessing the associated public data member. Now however, there isn't a physical data member that we can manipulate. Instead, we have to explicitly access the SDEs defined on our class via methods provided by the OGSI.NET code base (see Figure 22).

```
…
      private void incrementCalls()
      {
         OGSIServiceData sde = GetServiceData(
            new XmlQualifiedName("_numCalls"));
         ArrayList values = sde.SDValues;

         values[0] = ((int)(values[0])) + 1;
      }

      public SimpleMathService()
      {
         OGSIServiceData sde = GetServiceData(
            new XmlQualifiedName("_numCalls"));

         sde.Add(0);
      }

      [WebMethod]
      [return: XmlElement("sumResult")]
      public int add(int one, int two)
      {
         incrementCalls();
         return one + two;
      }

      [WebMethod]
      [return: XmlElement("differenceResult")]
      public int subtract(int one, int two)
      {
         incrementCalls();
         return one - two;
      }
…
```

Figure 22: Modifying the _numCalls SDE inside the Simple Math Service Code

As was the case with the SimpleMathClient source code, you should add a static, private method to the service class which performs the operation of incrementing the SDE value. The constructor also needs to be changed to initialize the **_numCalls** SDE. For more information on modifying SDEs from within service code, please refer to the OGSI.NET Programmer's Reference.

At this point, you should be able to stop the *OGSIContainerService* service, rebuild the **SimpleMathService** project, and restart the container. Now we have to add code to the client to change this value. The code to do this is very similar to that for retrieving the SDE value. Essentially, you need to create the XML element that will be embedded in the SOAP message that your client sends to the service. In Figure 23, **setNumCalls** is a static method on the **SimpleMathClient** class which you can use to change the value of the **_numValues** SDE from within your driver code.

```
      static private void setNumCalls(
         GridServiceSOAPBinding gbinding, int val)
      {
         SetByServiceDataNames setData = new SetByServiceDataNames();
         setData.dataNames = new XmlElement[1];

         XmlElement setElement = OgsiInterface.SerializeObject(
            val, typeof(int),
            new XmlQualifiedName("_numCalls", ""));
         setData.dataNames[0] = setElement;
         XmlElement setServiceDataElement =
            OgsiInterface.SerializeObject(setData,
               typeof(SetByServiceDataNames), null);
         XmlElement ret =
            gbinding.setServiceData(setServiceDataElement);
      }

      [STAThread]
      static void Main(string[] args)
      {
...
         result = binding.subtract(one, two);
         Console.WriteLine("{0} - {1} = {2}", one, two, result);
         Console.WriteLIne("Number of calls made:  "
            + getNumCalls(gbinding));

         setNumCalls(gbinding, 0);
         Console.WriteLine("After set to 0:  " +
            getNumCalls(gbinding));
      }
...
```

Figure 23: Modifying the Client Code to Set the _numCalls SDE Value

Once you have made all of these changes, re-add the grid reference to your **SimpleMathService** service and re-compile your client project. Once again, you should now be able to run the client binary and verify that your client and service are running correctly.

Appendix A: Writing Post Build Steps for Visual Studio .Net

Visual Studio makes it relatively painless to create your own post and pre-build scripts to run when building your projects. These scripts run as command line shell scripts (.bat) and any command that would be valid in one of those scripts is valid here. In addition, Visual Studio provides a number of Macros (delimited by ***$(macro-name)***) which can be used inside the script. These macros are listed under the macros button in the post and pre build editors.

To bring up a post or pre build editor for a project, right click on that project in your solution browser and select from the popup menu the properties option (See Figure 24)

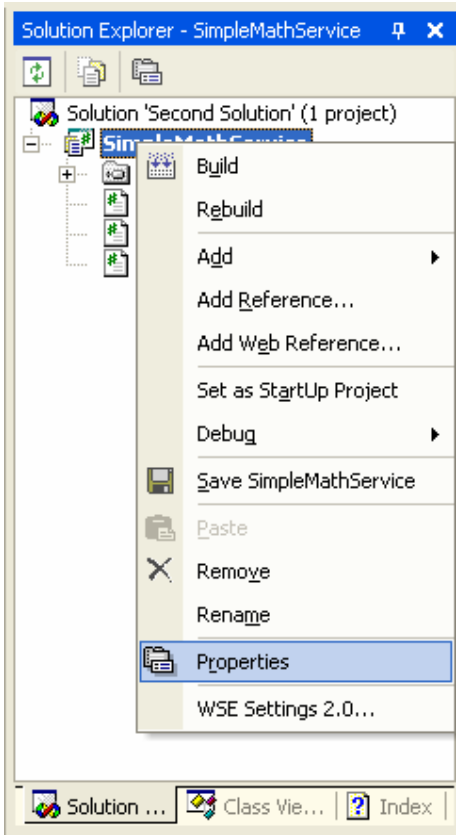

Figure 24: Project Properties Option

The properties dialog pops up on your screen and from here select **Build Events** and then customize the post and pre build event command lines (See Figure 25).

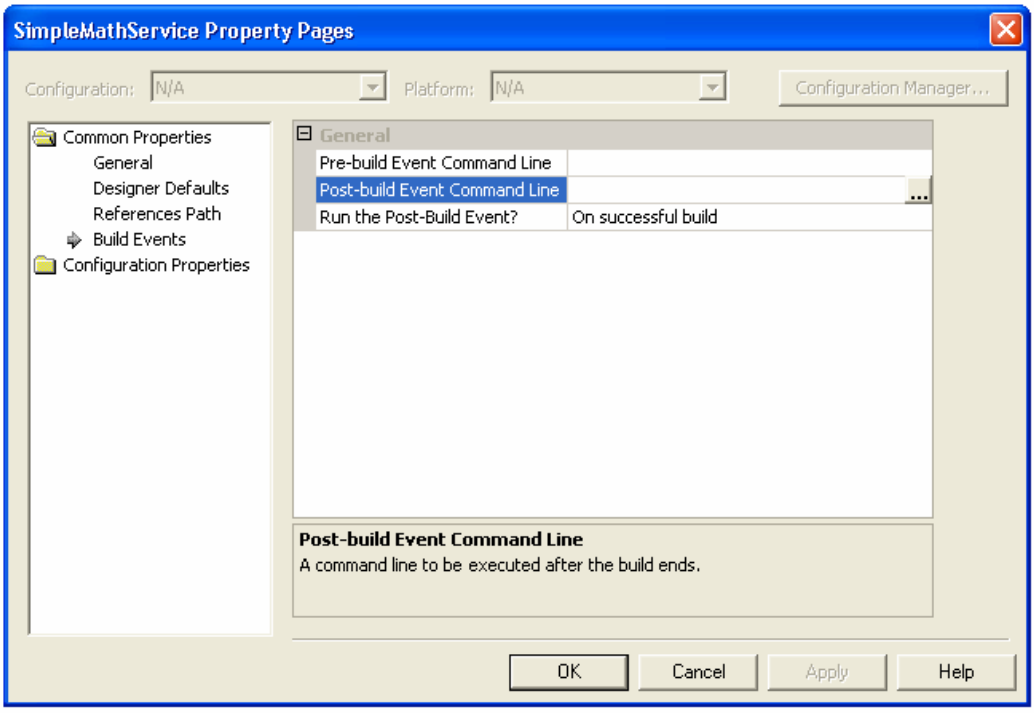

Figure 25: Selecting the Post-build Event Command Line

Appendix B: Controlling Windows Services

In order to start and stop windows services (not to be confused with web or grid services), you need to access the Windows Service Dialog control. The easiest way to get to this control is to select it from the Start Menu. Under Programs in the Start Menu, select Administrative Tools, and from that pop-up menu, select Services (See Figure 26). From here you can start and stop individual services by right-clicking on them.

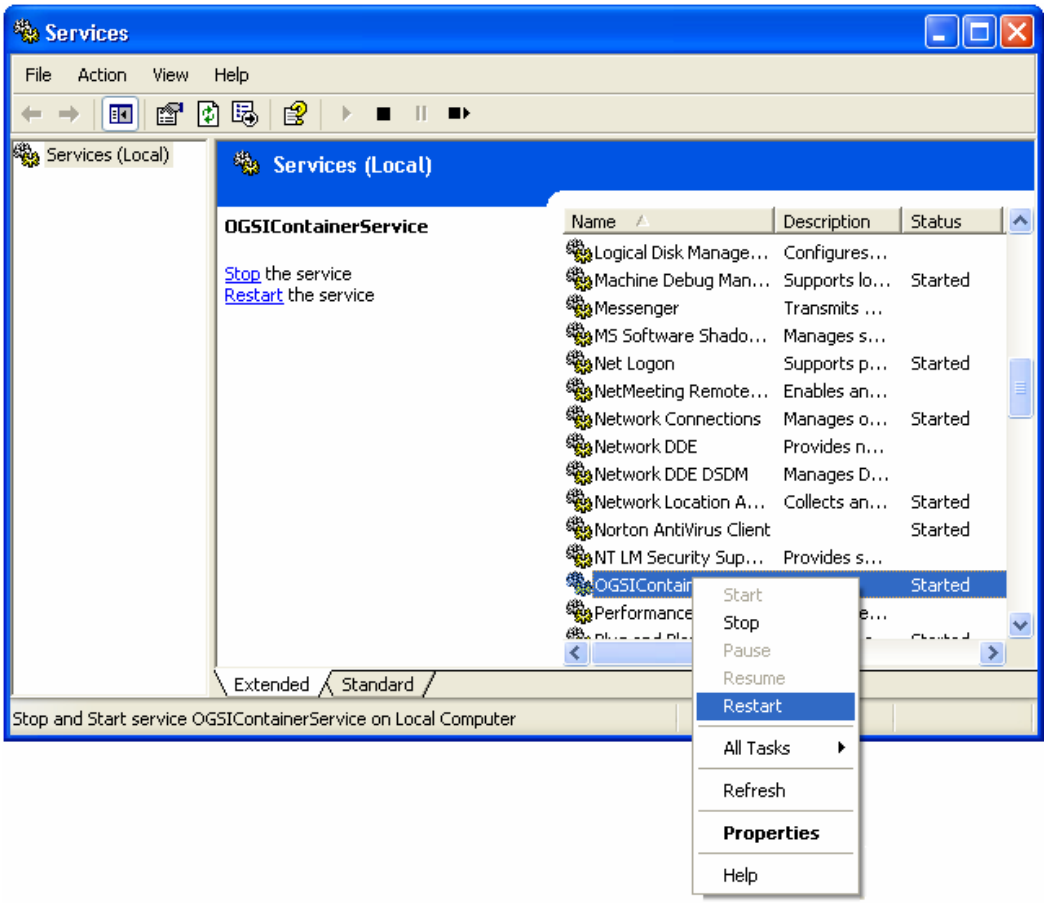

Figure 26: Starting and Stopping Windows Services