WEBORB/AMETHYST IDE PLUG-IN FOR VISUAL STUDIO

Reviewer/Evaluator Guide

Midnight Coders Headquarters
15455 Dallas Parkway, Suite 600
Addison, TX 75001
www.themidnightcoders.com

Contact: Kathleen Erickson
kathleen@themidnightcoders.com

Sapphire Steel Software Headquarters
Bideford, Devon, UK

www.sapphiresteel.com

contact: Huw Collingbourne
huw@sapphiresteel.com

TABLE OF CONTENTS

INTRODUCTION

Thank you for taking the time to review the WebORB/Amethyst IDE plug-in for Visual Studio. We're very excited to tell you why this plug-in makes building Flash/Flex applications very attractive to .NET development teams.

This guide is divided into four sections.

- General Overview
- Installation
- Hands-On Guide
- Product Licensing

Be sure to click on videos where provided to learn more about product features and benefits.

GENERAL OVERVIEW

The Adobe Flash platform has roughly 96% or more of an installed base, which makes developing for the Flash platform attractive on one hand. On the other hand, if a .NET developer wants to develop for the Flash platform, this attractiveness loses its luster once a .NET developer realizes he has to learn how to use two new IDEs (FlashBuilder and Eclipse) that don't adequately address integration. Working across multiple IDEs is not only frustrating and inefficient to .NET developers, but also increases development time and cost – something IT management cringes at, especially when scrutinized on spending. As a result, .NET developers have increasingly shied away from creating Flex/Flash applications only to embrace a still cumbersome process to develop a lesser adopted Silverlight platform that is rumored as having an uncertain future.

Bottom line: Having a single IDE that offers developers the flexibility to develop for multiple client-side technologies and the .NET platform ensures the most efficient workflow and optimal developer productivity. Furthermore, developing on top of a runtime (WebORB in this case)[1] that supports multiple client-side technologies and literally future proofs an application against technology obsolescence. Yet, it is unrealistic to assume a single IDE vendor can support everything, which is why 3rd party plug-ins are so important to developers.

Midnight Coders (WebORB) and Sapphire Steel (Amethyst) have created products that plug into the Visual Studio IDE. These products eliminate workflow inefficiencies and provide the flexibility developers are looking for in terms of technology solution stack, feature set and ability to customize their development environment.

The marrying of WebORB and Amethyst into a single unified interface within Visual Studio, creates a seamless work environment inside of one IDE and this enables .NET developers to take a GIANT STEP forward in terms of productivity. Furthermore, IT management can rest easy knowing they can develop for multiple client types efficiently, as needed and at a lower cost to develop and maintain.

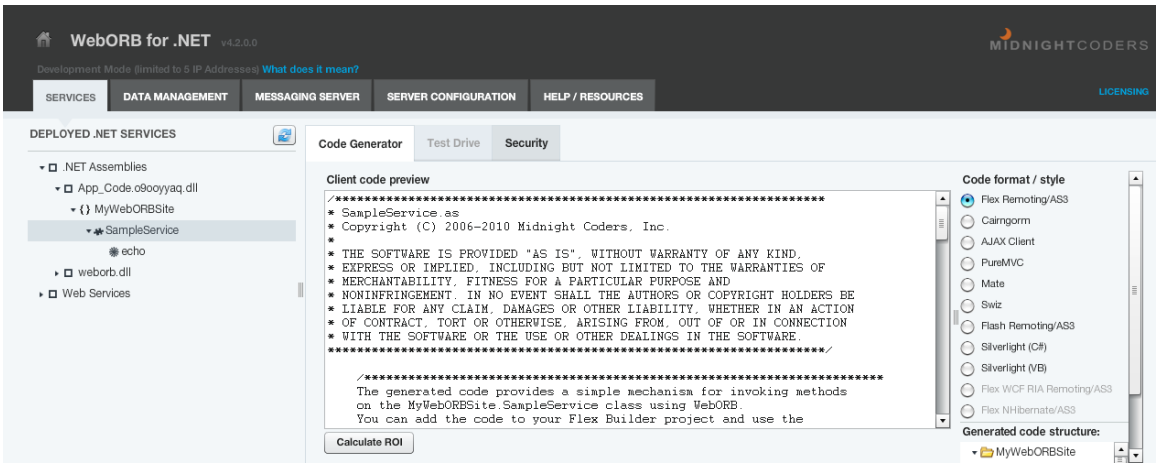
[1] WebORB is an integration/media server that is used not only in development, but in runtime to marshal the communication between client and server applications.

WHAT IS WEBORB?

WebORB is a full-featured integration/media server that is used to rapidly create an integration between various clients (Flex, Flash, JavaScript, and Silverlight) and multiple backends (.NET, Java and PHP). WebORB also plugs into IIS to extend any ASP.NET application with the capabilities that WebORB provides.

WebORB's integration capabilities include providing access from various clients to data and rich media (video and audio content) located on the server side. WebORB includes a runtime engine that is deployed with an application into production to marshall the communication between client applications and server-side resources. This marshalling includes real-time client-to-server, server-to-client, client-to-client and server-to-server communication.

While WebORB comes in three flavors (WebORB for .NET, WebORB for Java and WebORB for PHP) the WebORB/Amethyst IDE plug-in uses a the powerful Service Browser, Code Generation and Invocation Test Drive features of WebORB for .NET. Other features, like data management, graphical security and server configuration are available only in the full WebORB product download. Below is a view of the WebORB for .NET Service Browser module that includes code generation and invocation test drive..



KEY WEBORB FEATURES

Take a Guided Tour>

- WebORB for .NET is the only integration server that provides Flash, Flex, JavaScript and Silverlight connectivity to .NET servers. (Support for additional client-side technologies and platforms, such as HTML5, iOS and Android are being added to a future release.)
- WebORB is a runtime server that supports remoting, real-time messaging, data management and media streaming. It can be deployed in both premise and Cloud (Microsoft Azure, Amazon EC2 and Google AppEngine) environments.
- WebORB includes many developer productivity tools:
    - **Service Browser** – lists all deployed services, including .NET classes, Spring.NET objects, WCF Services, WCF RIA Service and Web Services. WebORB supports custom pluggable browsers. Watch This Video> http://www.youtube.com/watch?v=NJ_ifGK7AC4

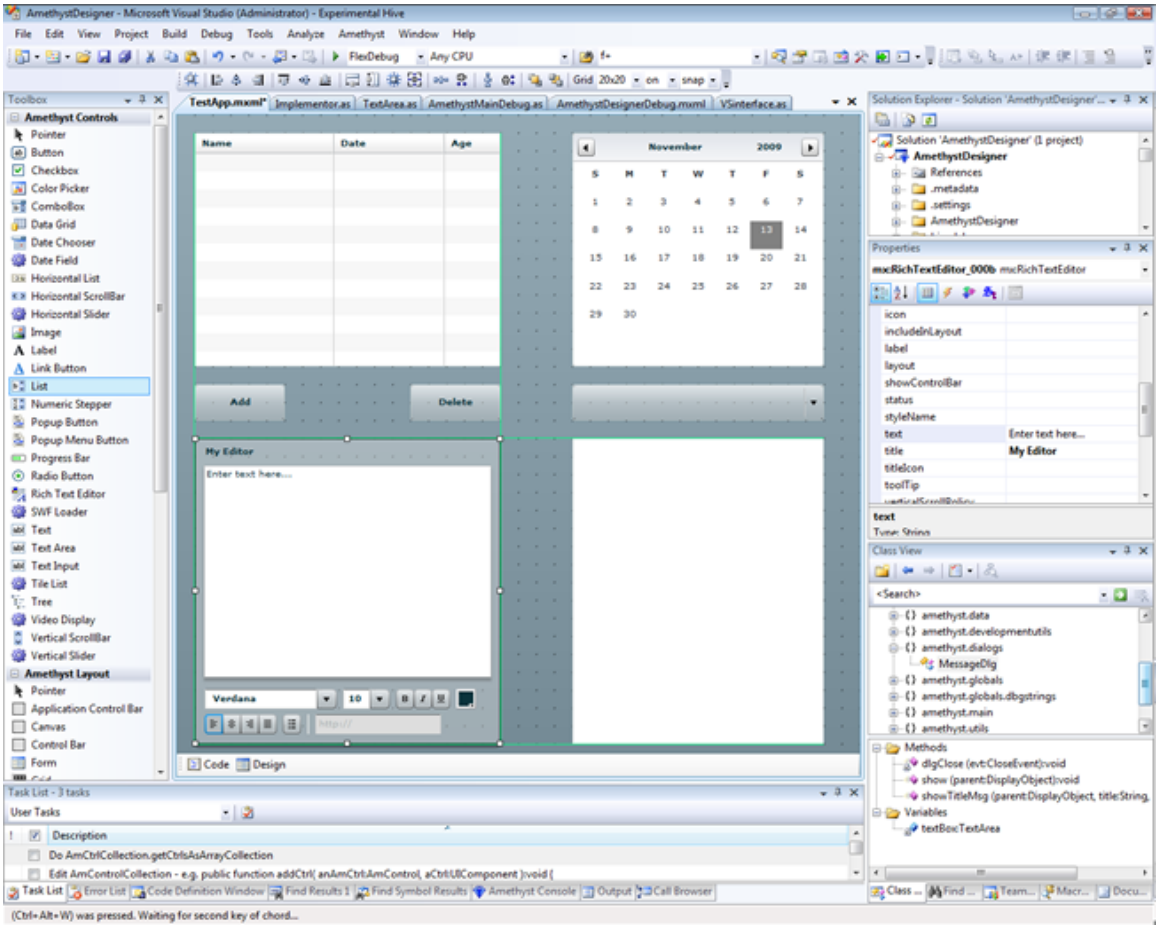
    - **Code Generators** – generates all of the client-side ActionScript, server-side C# and VB.NET, code documentation and project files necessary for client-to-server integration. There are code generators for Flex/Flash remoting (AS3), Cairngorm, PureMVC, Mate, Swiz, AJAX, Silverlight and NHibernate frameworks. Developers can easily add their own custom code generators as well. Watch This Video> http://www.youtube.com/watch?v=N0sWhDdsE0g
    - **Invocation Test Drive** – enables developers to invoke their methods to ensure proper returned results before deploying the integration code to the client-side project. Full client-to-server debugging is available in the WebORB/Amethyst plug-in.
    - **Data Management** – is a powerful framework that simplifies the creation of data driven applications. The framework includes a data modeler, code generators, sample test drive application, runtime engine and CRUD (create, retrieve, update and delete) API, as well as, mechanisms to process requests in transactions, data paging, sorting, client synchronization and many other data management operations. Watch This Video> http://www.youtube.com/user/MidnightCodersVideos#p/u/1/5yKQuycqXZ8
    - **Graphical Security Configuration** –enables production systems to be protected against malicious attacks and service outages through multiple, graphically configurable security settings.
    - **Graphical Server Configuration** – offers an alternative to command line configuration to ease the process of configuring the server for optimal performance.
- WebORB is extensible, which means it has multiple features which may be customized to meet the needs of the development environment.
- WebORB integrates with multiple IDEs (Visual Studio, Eclipse and Flash/Flex Builder).

KEY WEBORB BENEFITS

- .NET developers have the option to develop for both .NET and non-.NET client applications without sacrificing productivity, because their workflow is more efficient.
- A more efficient workflow process shortens the development cycle, enabling applications to be ready for production earlier, potentially enabling businesses to derive revenue earlier.
- A shortened development cycle reduces overall cost to develop.
- WebORB's full list of features and extensibility options literally future-proofs a company's investment in technology. WebORB can handle changes in client technologies, server environments and application feature-sets without requiring a complete rewrite.

WHAT IS AMETHYST?

Amethyst is the only non-Adobe IDE that provides a complete visual design, editing and debugging environment inside of Visual Studio, while simultaneously providing access to the graphics and animation features of the Flash IDE. Below is a screenshot of Amethyst's design view running inside of Visual Studio.



KEY AMETHYST FEATURES

Take a Guided Tour>

- Refactoring (SmartTags, Encapsulate Field, Rename, etc.)
- Visual drag-and-drop Designer
- Advanced Debugging with conditional breakpoints, Immediate Window, etc.
- Debugger IntelliSense
- Ability to debug multiple SWF files simultaneously
- Customizable code reformatting
- Organize import statements (remove, sort)
- Code Snippets and Snippet Editor
- Adobe Flash CS4 and CS5 IDE integration
- Advanced IntelliSense (Find All References, Go to Definition)
- Library projects
- Support for large team-based projects

KEY AMETHYST BENEFITS

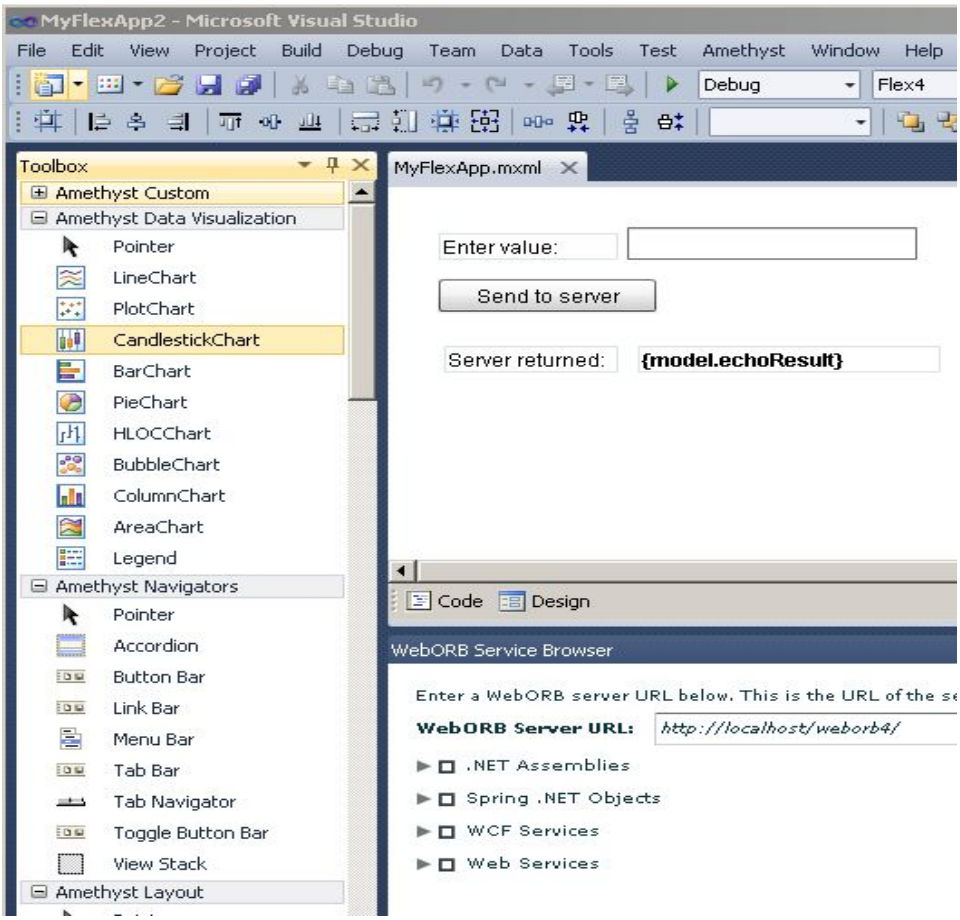
- Produces a more efficient workflow by enabling teams to work on projects using a mix of development tools, all within a single IDE.
- Reduces the debugging cycle by enabling developers to conduct full Flash/Flex to .NET debugging in a single view.

WHAT IS THE WEBORB/AMETHYST PLUG-IN FOR VISUAL STUDIO?

With WebORB and Amethyst now seamlessly integrated and capable of running inside of Visual Studio, developers can, for the first time, use the Visual Studio IDE to do complete design, development, debugging and integration within a single IDE. This enables developers to reap the full-features and benefits offered by Amethyst, WebORB and Visual Studio for a more efficient and less costly development effort. This is very exciting to see in action, so take a moment to view this video. It will walk you through the Hands-On Example included in this guide.

http://www.youtube.com/watch?v=Hy6gZUQz0O0&feature=uploademail

Below is a screenshot of Amethyst and WebORB running side-by-side inside of Visual Studio.



INSTALLATION

Now that we have given you an overview of both WebORB and Amethyst's various features and benefits and how they can be utilized inside of Visual Studio, this next section will help you experience the features and benefits first hand through designing, developing, testing and running your own Visual Studio created client- and server-side applications. Before we get started, you will need to ensure you have met the following prerequisites:

PREREQUISITES

Before installing the WebORB/Amethyst IDE plug-in, you should be sure that you have the prerequisites necessary to run Visual Studio and the WebORB/Amethyst plug-in. These prerequisites include:

- Microsoft .NET Framework 2.0 or later (including IIS)
- Visual Studio 2008 or 2010 (Shell edition can be used, with some feature limitations)
- Flash Player 10 Debug Version for Internet Explorer
  http://www.adobe.com/support/flashplayer/downloads.html
- Flex SDK (normally optional, but required for the example in this guide)
  http://opensource.adobe.com/flex
- Java 2 Runtime Environment (needed for Flex compiler)
  http://www.java.com/en/download

Optional

- Adobe Flash IDE (CS3, CS4 or CS5) may optionally share projects with Amethyst. (Not needed for the example in this guide.)

MINIMUM SYSTEM REQUIREMENTS

The minimum requirements for running the WebORB/Amethyst IDE Plug-in follow those requirements for Visual Studio. The minimum requirements for Visual Studio 2010 include:

- Windows XP (x86) with Service Pack 3 - all editions except Starter
- Windows Vista (x86 & x64) with Service Pack 2 - all editions except Starter
- Windows 7 (x86 & x64)
- Windows Server 2003 (x86 & x64) with Service Pack 2
- Windows Server 2003 R2 (x86 & x64)
- Windows Server 2008 (x86 & x64) with Service Pack 2
- Windows Server 2008 R2 (x64)

Supported Architectures:

- 32-Bit (x86)
- 64-Bit (x64)

Hardware Requirements:

- Computer that has a 1.6GHz or faster processor
- 1 GB (32 Bit) or 2 GB (64 Bit) RAM (Add 512 MB if running in a virtual machine)
- 3GB of available hard disk space
- 5400 RPM hard disk drive
- DirectX 9 capable video card running at 1024 x 768 or higher-resolution display
- DVD-ROM Drive

Once you have installed the prerequisites, go ahead and download and install the Amethyst/WebORB IDE plug-in for this location:

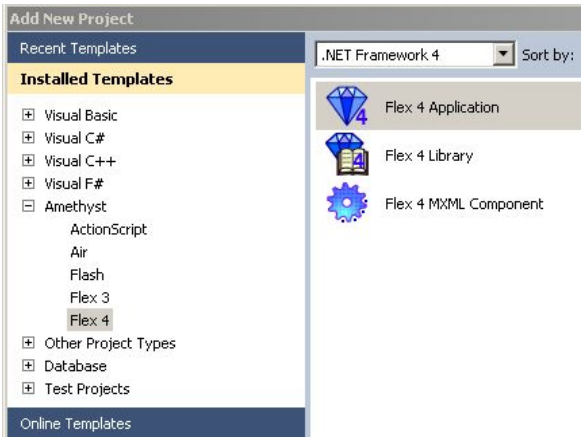
http://www.sapphiresteel.com/Products/amethyst-ide/Download-Amethyst-Adobe-Flash

This download will include the full Professional Edition of Amethyst running as a 60 day trial.

HANDS-ON EXAMPLE

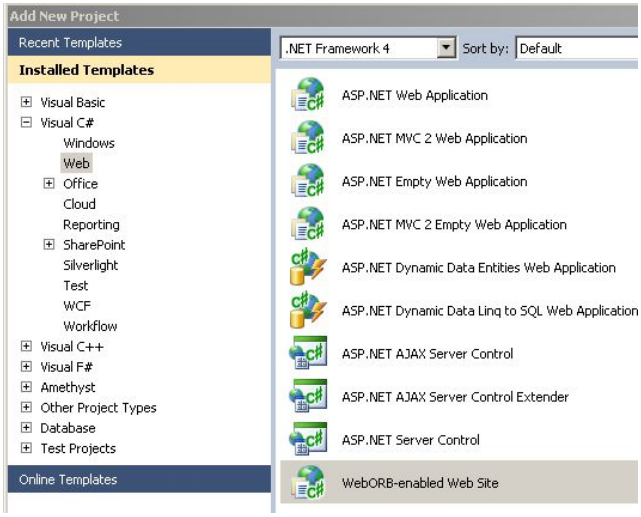
This guide will walk you through the process of creating, debugging and running a remoting-enabled Flex application connecting to a .NET service using Amethyst and WebORB running inside of Visual Studio.

Step 1 Run Visual Studio and select File>NewProject>Amethyst>Flex (You can select either Flex 3 or Flex 4.) For this example, choose Flex 4 (assuming you downloaded the Flex 4 SDK). Name your project MyFlexApp and select OK. You will be prompted to select the location of the Flex SDK since this will be the first time you've built a Flex application using Amethyst running in Visual Studio.



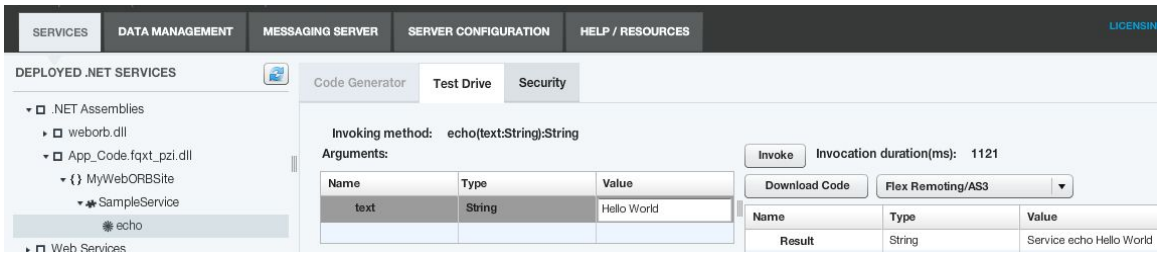
Step 2 Amethyst creates a default Flex application. Go ahead and compile the project by right clicking on MyFlexApp in the Solution Explorer and selecting Build.

Step 3 Create your server-side application using WebORB's built-in Visual Studio project templates. To do this, select the solution node called MyFlexApp again and select Add>New Project. Then select the Visual C# node from the Installed Templates and drill down to Web>WebORB-enabled Web Site. (WebORB includes Visual Studio templates.) Next name your website MyWebORBSite. Visual Studio imports the project into the Solution Explorer. Right click on MyWebORBSite and select Set as StartUp Project. Also right click on weborbconsole.html and select it as the startup page. The WebORB template automatically includes a template service under the App_Code node called SampleService.cs



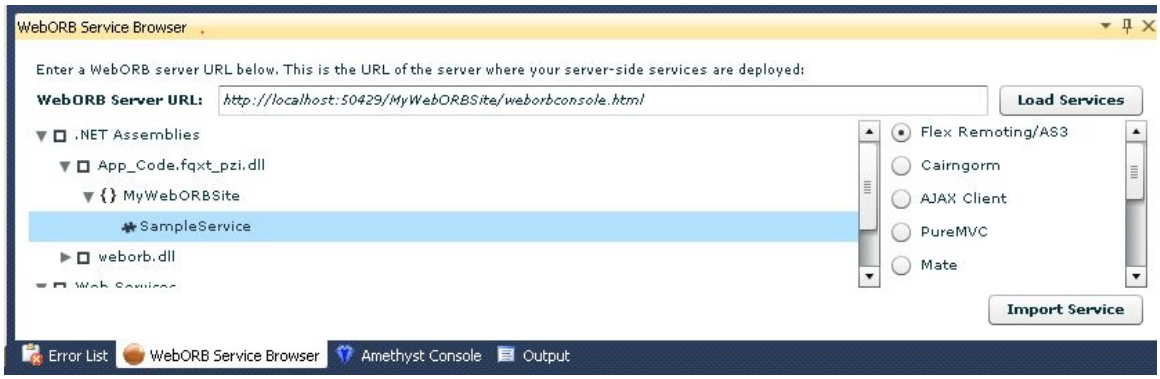
Step 4 Right click on the MyWebORBSite in the Solution Explorer and select Build Web Site. Then click on Debug at the top of the screen to run the application. Since you selected weborbconsole.html as your startup page, Visual Studio will load the WebORB Console in the browser, enabling you to verify that the web site is working properly.

Step 5 Select the Services tab and drill down to into the .NET Assemblies until you get to echo in MyWebORBSite. This brings you to the Test Drive functionality within WebORB. Here you can enter a value; click invoke to return a response; and receive a result returned from the server, which enables you to inspect complete client-to-server side communication for proper operation.



Step 6 Return to Visual Studio, locate and right click on the Services node in the MyFlexApp project and select Add WebORB Services. This is where you will see the integration between Amethyst and WebORB inside of Visual Studio. Amethyst opens up an instance of WebORB Service Browser. Here you will need to enter the location of the WebORB Server. Since you already have a WebORB Server installed as part of your server-side project you can simply go to the URL we opened before, which had the WebORB console running in the browser.

For example:



Click Load Services. What you will see is WebORB Service Browser running in Visual Studio. You'll also note that this Service Browser will return the same listing of services that were displayed in the WebORB console running in the browser.

Step 7 Drill down to the SampleService node in the WebORB Service Browser. This is where it gets really interesting. See the list of frameworks listed on the right hand side of the WebORB Service Browser? That list represents WebORB's built-in code generators. You can select any one of those frameworks and then select Import Service and WebORB automatically generates the documented integration code and deploys this code and project files right into the Flex project – all without ever leaving Visual Studio. For this example, select Flex Remoting/AS3 and then select Import Service. Verify that you now have the generated project files in your Solution Explorer by drilling down to MyFlexApp>MyWebORBSite. You will see two files:

SampleService.as
SampleServiceModel.as

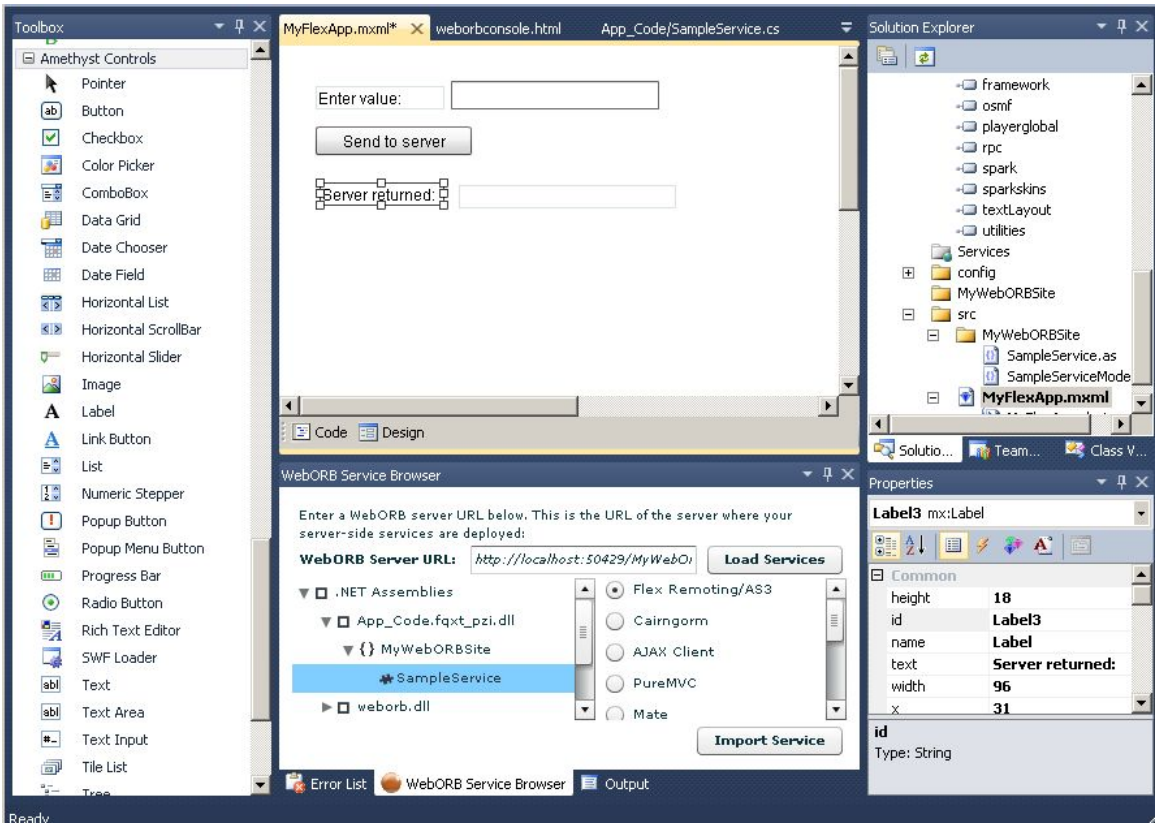
Upon clicking on the first file, a screen that displays the documented integration code that was generated for this service appears. This is all the code needed to enable the Flex client to communicate with WebORB.

Step 8 Next, you will create a very basic application that uses the generated code and communicates with WebORB. Open the MyFlexApp.mxml from the Solution Explorer and click on the Design view. This is Amethyst's Designer running inside of Visual Studio. Now if you will remember, we had you create a .NET service that accepts a string and returns a string. What you are going to do in the Amethyst Designer is create the Flex UI components. Open up the Toolbox. If you see a message that says "There are no usable controls in this group…" click on Amethyst at the top of the screen and then click on Component Manager. Then select"Set Default" and "Yes". This will add Amethyst Controls to the Toolbox.

Using the components, you will need to add the following to the design workspace:

- Text input field that represents the client-side Input

- Label that is labeled as Enter Value
- Button that performs the method invocation. Label that button Send to Server
- Label that represents the returned value from the server
- Label that is labeled as Server returned:



Step 9 Next you will need to wire the button to the actual method invocation. For this you will need to switch back to the Code view. (Hide the toolbox.) Click the SampleService.as in the Solution Explorer to bring the generated code back into view. It looks like this:

```
/*****************************************************************
* SampleService.as
* Copyright (C) 2006-2010 Midnight Coders, Inc.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
* EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
* NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
* LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
* WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*****************************************************************/
```

```
/**************************************************************************
The generated code provides a simple mechanism for invoking methods
on the MyWebORBSite.SampleService class using WebORB.
You can add the code to your Flex Builder project and use the
class as shown below:

        import MyWebORBSite.SampleService;
        import MyWebORBSite.SampleServiceModel;

        [Bindable]
        var model:SampleServiceModel = new SampleServiceModel();
        var serviceProxy:SampleService = new SampleService( model );
        // make sure to substitute foo() with a method from the class
        serviceProxy.foo();

Notice the model variable is shown in the example above as Bindable.
You can bind your UI components to the fields in the model object.
**************************************************************************/

package MyWebORBSite
{
import mx.rpc.remoting.RemoteObject;
import mx.controls.Alert;
import mx.rpc.events.ResultEvent;
import mx.rpc.events.FaultEvent;
import mx.rpc.AsyncToken;
import mx.rpc.IResponder;
import mx.collections.ArrayCollection;



public class SampleService
{
  private var remoteObject:RemoteObject;
  private var model:SampleServiceModel;

  public function SampleService( model:SampleServiceModel = null )
  {
    remoteObject  = new RemoteObject("GenericDestination");
    remoteObject.source = "MyWebORBSite.SampleService";

    remoteObject.echo.addEventListener("result",echoHandler);

    remoteObject.addEventListener("fault", onFault);

    if( model == null )
        model = new SampleServiceModel();

    this.model = model;

  }

  public function setCredentials( userid:String, password:String ):void
  {
    remoteObject.setCredentials( userid, password );
  }
```

```
        public function GetModel():SampleServiceModel
        {
          return this.model;
        }


        public function echo(text:String, responder:IResponder = null ):void
        {
          var asyncToken:AsyncToken = remoteObject.echo(text);

          if( responder != null )
              asyncToken.addResponder( responder );

        }

        public virtual function echoHandler(event:ResultEvent):void
        {

            var returnValue:String = event.result as String;
            model.echoResult = returnValue;

        }

        public function onFault (event:FaultEvent):void
        {
          Alert.show(event.fault.faultString, "Error");
        }
      }
    }
```

Next, select MyFlexApp.designer.as from the Solution Explorer and then select line 6 in the Code view, press enter to create line 7. Then copy and paste the highlighted portion of code above starting onto to line 7 . What you have just added is a bindable model, which will contain results from the remote method invocations as well as the actual service proxy. So this SampleService ActionScript class is a proxy for the remote .NET service.

Step 10 Next, add a method by copying and pasting the highlighted section of code below into your MyFlexApp.designer.as code.

```
//

// Amethyst Designer file for MyFlexApp.mxml

//

import mx.events.*;
```

```
import flash.events.*;


import MyWebORBSite.SampleService;

import MyWebORBSite.SampleServiceModel;


[Bindable]

var model:SampleServiceModel= new SampleServiceModel();

var serviceProxy:SampleService= new SampleService(model);


public function sendDataToServer(val:String):void {

                    serviceProxy.echo(val);

}
```

This performs the remote method invocation. When the result is received by Flex, the generated code will automatically update this model object. This means that in the UI you can bind the label that displays the label from the server to the property in this particular model. Click on SampleServiceModel.as to find the name of the property. In this example it is echoResult (this is found in the SampleServiceModel.as.

```
15      package MyWebORBSite
16      {
17        [Bindable]
18        public class SampleServiceModel
19        {
20          public var echoResult:String;
21        }
22      }
23
```

Step 11 To bind the labels and the button in the Flex application to the property values in their respective fields field, go ahead and copy and paste the following into your MyFlexApp.mxml, replacing the content that is already there.

```
<?xml version="1.0" encoding="utf-8"?>

<s:Application height="600" width="800" xmlns:fx="http://ns.adobe.com/mxml/2009"
                    xmlns:mx="library://ns.adobe.com/flex/mx"
                    xmlns:s="library://ns.adobe.com/flex/spark">
```

```
        <fx:Script source="MyFlexApp.designer.as"/>

        <mx:TextInput height="22" id="myInput" name="TextInput" width="160" x="132"
        y="27"/>

        <mx:Label height="18" id="Label1" name="Label" text="Enter value:" width="99"
        x="28" y="31"/>

        <mx:Button height="22" id="Button1" label="Send to server" name="Button"
        width="120" x="28" y="62" click="{sendDataToServer( myInput.text )}"/>

        <mx:Label fontWeight="bold" height="18" id="Label2" name="Label"
        text="{model.echoResult}" width="167" x="138" y="107"/>

        <mx:Label height="18" id="Label3" name="Label" text="Server returned:"
        width="96" x="31" y="107"/>

</s:Application>
```

Step 12 Now you are ready to compile your Flex application. Right click on MyFlexApp in the Solution Explorer and select Build.

Step 13 Once successfully compiled you can debug and run the application. Right click on MyFlexApp again and select Set as Startup Project and the select the arrow next to Debug at the top of the screen. This launches the Flex app in the browser.

Step 14 Enter a value in the input field and the click Send to Server. The server should return a value starting with .NET returned: (whatever you entered in the input field).

What you have just witnessed is the successful round trip communication between a Flex application and a .NET service. Both client- and server-side projects were created, debugged and run inside of Visual Studio using the Amethyst/WebORB IDE plug-in. **This is very powerful because it demonstrates how .NET developers now have access to a streamlined workflow for developing end-to-end Adobe Flex, Flash and AIR applications connecting to .NET services using a single IDE - Visual Studio.**

Next view this video to see how to use Visual Studio for debugging across both client- and server-side projects in a single session.

http://www.youtube.com/watch?v=t1IZvD91Ks0

PRODUCT LICENSING

WebORB and Amethyst have a couple of different licensing models based on deployment needs. The following information describes these licensing models relative to development and production environments. While Amethyst is considered a tool used in development, WebORB is used in both development and production environments.

DEVELOPMENT ENVIRONMENTS

Amethyst offers both Professional and Personal Editions for development.

- **Professional Edition** - the full-featured commercial version of Amethyst that includes Amethyst Designer, extensive code refactoring, IntelliSense, debugger and integration with WebORB running in development mode. Professional Edition is licensed for $398/developer with WebORB or $249/developer without WebORB. Developers can download a free 60-day trial of Professional Edition.
- **Personal Edition** – limits the range of editing and debugging features and does not include the integration with WebORB. Personal Edition is free.

The Amethyst/WebORB free trial can be downloaded here:

http://www.sapphiresteel.com/Products/amethyst-ide/article/amethyst-release-history

PRODUCTION ENVIRONMENTS

Only WebORB Community Edition or WebORB Enterprise Edition is needed to deploy a WebORB-enabled application into production.

- **WebORB Community Edition** is free and is ideal for single-server deployments.
- **WebORB Enterprise Edition** has various licensing plans available depending upon the type of deployment. These plans are described here: http://www.themidnightcoders.com/products/weborb-for-net/licensing.html

WebORB for .NET running in development mode can be downloaded here:
http://www.themidnightcoders.com/products/weborb-for-net/download.html

ADDITIONAL RESOURCES

WebORB for .NET product page

http://www.themidnightcoders.com/products/weborb-for-net/overview.html

Amethyst Product Page

http://www.sapphiresteel.com/Products/amethyst-ide/article/amethyst-product-page

Amethyst Reviewers Guide

http://www.sapphiresteel.com/info/press-room/article/amethyst-media-information

Thank you for reviewing our product. If you have any questions, please contact

Kathleen Erickson
kathleen@themidnightcoders.com