SILVERLIGHT

Build Line-Of-Business Enterprise Apps With Silverlight, Part 2

Hanu Kommalapati

THIS ARTICLE DISCUSSES:

- The Silverlight runtime environment
- Silverlight asynchronous programming
- Cross-domain policies
- A sample enterprise application

THIS ARTICLE USES THE FOLLOWING TECHNOLOGIES:

Silverlight 2

CODE DOWNLOAD AVAILABLE FROM THE MSDN CODE GALLERY

Browse the Code Online

Contents

During the first installment of this series, I introduced a call center scenario and showed a screen-population (screen pop) implementation through the connected sockets that utilized the asynchronous TCP sockets supported by Silverlight (please see "Build Line-Of-Business Enterprise Apps With Silverlight, Part 1").

The screen pop was implemented through a simulated call dispatcher that picked up a call from an internal queue and pushed notifications through the previously accepted socket connection cached in a generic list on the server. Here I will conclude by implementing application security, integrating with business services, and implementing cross-domain policies for Web services and application partitioning. The logical architecture of the call center application is shown in **Figure 1**. The authentication service will be implemented in the utility service while the business services, ICallService and IUserProfile, will be implemented inside the business service project, as the name suggests.

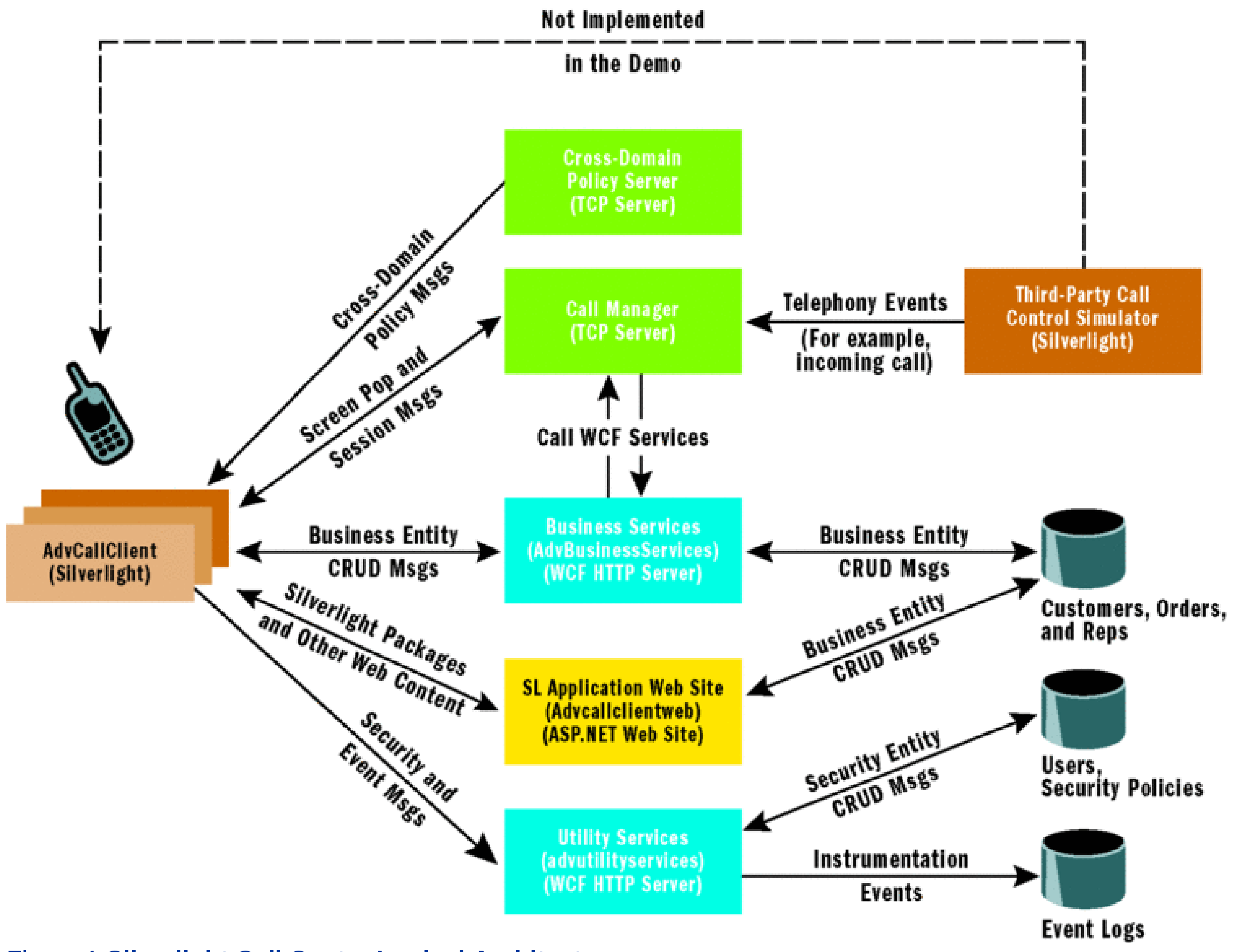

Not Implemented
in the Demo
Cross-Domain Policy Server (TCP Server)
Call Manager (TCP Server)
Third-Party Call Control Simulator (Silverlight)
Telephony Events
(For example, incoming call)
Cross-Domain Policy Msgs
Screen Pop and Session Msgs
Call WCF Services
AdvCallClient (Silverlight)
Business Entity CRUD Msgs
Business Services (AdvBusinessServices) (WCF HTTP Server)
Business Entity CRUD Msgs
Customers, Orders, and Reps
Silverlight Packages and Other Web Content
SL Application Web Site (Advcallclientweb) (ASP.NET Web Site)
Business Entity CRUD Msgs
Security and Event Msgs
Users, Security Policies
Security Entity CRUD Msgs
Utility Services (advutilityservices) (WCF HTTP Server)
Instrumentation Events
Event Logs

Figure 1 **Silverlight Call Center Logical Architecture**

Even though the diagram shows event streaming to utility services, in the interest of time, the downloadable demo does not include this functionality. The implementation of the event capture service feature will be similar to the business services implementation. However, business events that are not critical errors can be cached locally into isolated storage and dumped onto the server in a batch mode. I will begin the discussion with the implementation of business services and conclude with application partitioning.

Integration with Business Services

Integration with services is one of the important aspects of a line-of-business (LOB) application, and Silverlight provides ample components for accessing Web-based resources and services. HttpWebRequest, WebClient, and Windows Communication Foundation (WCF) proxy infrastructure are a few of the commonly used network components for HTTP-based interaction. In this article, I will use WCF services to integrate with the back-end business processes.

Most of us use Web services for integrating with the back-end data sources during the course of application development; WCF Web service access with Silverlight is not much different than it is with traditional applications such as ASP.NET, Windows Presentation Foundation (WPF) or Windows Forms applications. The differences are the binding support and the asynchronous programming model. Silverlight will only support basicHttpBinding and PollingDuplexHttpBinding. Note that HttpBinding is the most interoperable binding. For this reason, I will use it for all integration in this article.

PollingDuplexHttpBinding allows the use of callback contracts to push notifications over HTTP. My call center could have used this binding for screen-pop notifications. However, the implementation requires the caching

of the HTTP connection on the server, thereby monopolizing one of the two concurrent HTTP connections allowed by browsers such as Internet Explorer 7.0. This can cause performance issues, as all the Web content will have to be serialized through one connection. Internet Explorer 8.0 allows six concurrent connections per domain and will address such performance issues. (Push notifications using PollingDuplexHttpBinding could be a topic for a future article when Internet Explorer 8.0 is widely available.)
Back to the application. When the agent accepts a call, the screen-pop process populates the screen with the caller information—in this case, the order details of the caller. The caller information should contain necessary information to uniquely identify the order in the back-end database. For this demo scenario, I will assume that the order number was spoken into the interactive voice response (IVR) system. The Silverlight application will call WCF Web services with the order number as the unique identifier. The service contract definition and the implementation is shown in **Figure 2**.

Figure 2 Business Service Implementation

ServiceContracts.cs

Copy Code

```
[ServiceContract]
public interface ICallService
{
    [OperationContract]
    AgentScript GetAgentScript(string orderNumber);
    [OperationContract]
    OrderInfo GetOrderDetails(string orderNumber);
}

[ServiceContract]
public interface IUserProfile
{
    [OperationContract]
    User GetUser(string userID);
}
```

CallService.svc.cs

Copy Code

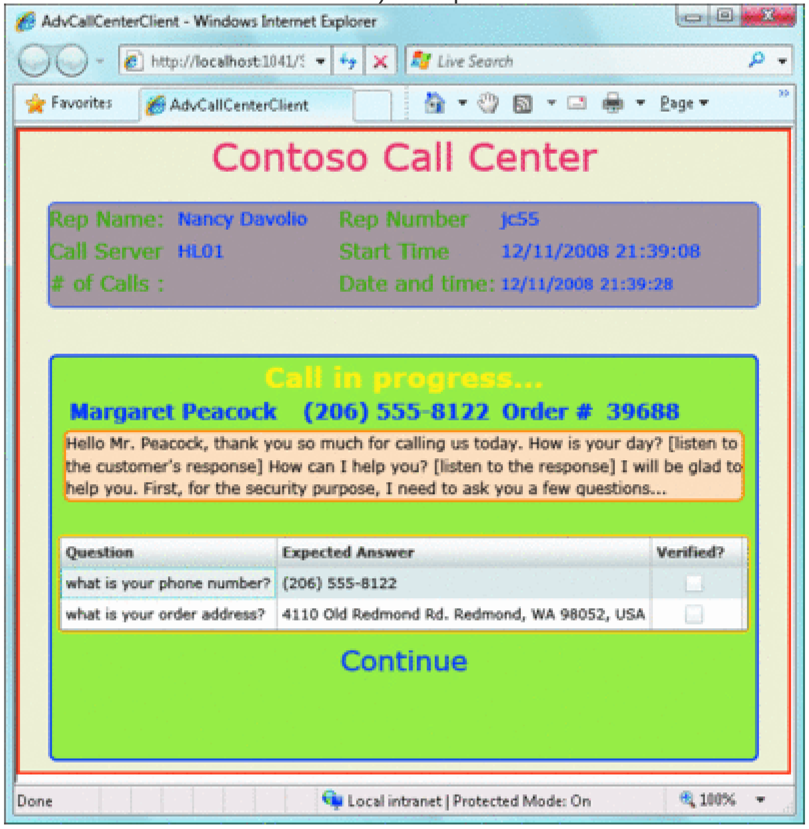
```
 [AspNetCompatibilityRequirements(RequirementsMode =
                            AspNetCompatibilityRequirementsMode.Allowed)]
public class CallService:ICallService, IUserProfile
{
  public AgentScript GetAgentScript(string orderNumber)
  {
    ...
    script.QuestionList = DataUtility.GetSecurityQuestions(orderNumber);
    return script;
  }

  public OrderInfo GetOrderDetails(string orderNumber)
  {
    ...
    oi.Customer = DataUtility.GetCustomerByID(oi.Order.CustomerID);
    return oi;
  }

  public User GetUser(string userID)
  {
    return DataUtility.GetUserByID(userID);
  }
 }
```

Web.Config

Copy Code

```
<system.servicemodel>
   <services>
     <endpoint binding="basicHttpBinding"
contract="AdvBusinessServices.ICallService"/>
     <endpoint binding="basicHttpBinding"
contract="AdvBusinessServices.IUserProfile"/>
   </services>
   <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
<system.servicemodel>
```

The implementation of these service endpoints is not really very interesting, as these are straightforward WCF implementations. For simplicity's sake, I will not use any database for business entities but will use in-memory List objects for storing Customer, Order, and User objects. The DataUtil class (not shown here but available in the code download) encapsulates access to these in-memory List objects.


AdvCallCenterClient - Windows Internet Explorer
http://localhost:1041/
Live Search
Favorites
AdvCallCenterClient
Page
Continue
Done
Local intranet | Protected Mode: On
100%

Contoso Call Center

Rep Name: Nancy Davolio   Rep Number jc55
Call Server HL01   Start Time 12/11/2008 21:39:08
# of Calls :   Date and time: 12/11/2008 21:39:28

Call in progress...

Margaret Peacock (206) 555-8122 Order # 39688

Hello Mr. Peacock, thank you so much for calling us today. How is your day? [listen to the customer's response] How can I help you? [listen to the response] I will be glad to help you. First, for the security purpose, I need to ask you a few questions...

Question	Expected Answer	Verified?
what is your phone number?	(206) 555-8122	☐
what is your order address?	4110 Old Redmond Rd. Redmond, WA 98052, USA	☐

Figure 3 **Agent Script with Security Questions**

WCF service endpoints for Silverlight consumption need access to the ASP.NET pipeline and hence require the AspNetCompatibilityRequirements attribute on the CallService implementation. This has to be matched by the <serviceHostingEnvironment/> setting in the web.config file.

As previously mentioned, Silverlight only supports basicHttpBinding and PollingDuplexHttpBinding. If you use the WCF Service Visual Studio template, it configures the endpoint binding to wsHttpBinding, which must be manually changed to basicHttpBinding before Silverlight can add a service reference for proxy generation. The ASP.NET hosting compatibility changes and binding changes will automatically be taken care of if CallService.svc is added to the AdvBusinessServices project using a Silverlight-enabled WCF Service Visual Studio template.

Service Invocation

Having implemented a Silverlight callable service, now it is time to create service proxies and use them to wire up the UI to the back-end service implementations. You can only generate proxies for WCF services reliably by using the Service References | Add Service Reference sequence in Visual Studio. The proxies in my demo were generated into the namespace CallBusinessProxy. Silverlight only allows asynchronous calls to the network resources, and service invocation is no exception. When a customer call comes in, the Silverlight client will listen to the notification and display an Accept/Reject dialog.

Once a call is accepted by the agent, the next step in the process is to call the Web service to retrieve the agent script that corresponds to the call situation. For this demo, I will only be using one script as displayed in **Figure 3**. The displayed script contains a greeting as well as a list of security questions. The agent will ensure that a minimum number of questions are answered before moving forward with the assistance.

The agent script is retrieved by accessing the ICallService.GetAgentScript(), providing the order number as the input. Consistent with the asynchronous programming model enforced by the Silverlight Web services stack, the GetAgentScript() is available as CallServiceClient.BeginGetAgentScript(). While making the service call, you will need to provide a callback handler, GetAgentScriptCallback, as shown in **Figure 4**.

Figure 4 Service Invocation and Silverlight UI Change

Copy Code

```
class Page:UserControl
{
   ...
   void _notifyCallPopup_OnAccept(object sender, EventArgs e)
   {
     AcceptMessage acceptMsg = new AcceptMessage();
     acceptMsg.RepNumber = ClientGlobals.currentUser.RepNumber;
     ClientGlobals.socketClient.SendAsync(acceptMsg);
     this.borderCallProgressView.DataContext = ClientGlobals.callInfo;
     ICallService callService = new CallServiceClient();
     IAsyncResult result =
        callService.BeginGetAgentScript(ClientGlobals.callInfo.OrderNumber,
                     GetAgentScriptCallback, callService);
     //do a preemptive download of user control
     ThreadPool.QueueUserWorkItem(ExecuteControlDownload);
     //do a preemptive download of the order information
     ThreadPool.QueueUserWorkItem(ExecuteGetOrderDetails,
                ClientGlobals.callInfo.OrderNumber);
   }

   void GetAgentScriptCallback(IAsyncResult asyncReseult)
   {

     ICallService callService = asyncReseult.AsyncState as ICallService;
     CallBusinessProxy.AgentScript svcOutputAgentScript =
                     callService.EndGetAgentScript(asyncReseult);
     ClientEntityTranslator astobas =
                               SvcScriptToClientScript.entityTranslator;
     ClientEntities.AgentScript currentAgentScript =
                             astobas.ToClientEntity(svcOutputAgentScript)
                             as ClientEntities.AgentScript;
     Interlocked.Exchange<ClientEntities.AgentScript>(ref
                   ClientGlobals.currentAgentScript, currentAgentScript);
     if (this.Dispatcher.CheckAccess())
     {
       this.borderAgentScript.DataContext = ClientGlobals.agentScript;
       ...
       this.hlVerifyContinue.Visibility = Visibility.Visible;
     }
     else
     {
       this.Dispatcher.BeginInvoke(
        delegate()
        {
          this.borderAgentScript.DataContext = ClientGlobals.agentScript;
          ...
          this.hlVerifyContinue.Visibility = Visibility.Visible;

        } );
       }
     }
   private void ExecuteControlDownload(object state)
   {
     WebClient webClient = new WebClient();
     webClient.OpenReadCompleted += new
       OpenReadCompletedEventHandler(OrderDetailControlDownloadCallback);
     webClient.OpenReadAsync(new Uri("/ClientBin/AdvOrderClientControls.dll",
                                                    UriKind.Relative));
   }
   ...
}
```

Since the result of the service call can only be retrieved from the callback handler, any changes to the Silverlight application state will have to happen in the callback handler.

CallServiceClient.BeginGetAgentScript() is invoked by _notifyCallPopup_OnAccept running on the UI thread and queues the asynchronous request and immediately returns to the next statement. Since the agent script is not yet available, you have to wait until the callback is triggered before you cache the script and data bind it to the UI.

Successful completion of the service call triggers GetAgentScriptCallback, which retrieves the agent script, populates a global variable, and adjusts the UI by data binding the agent script to the appropriate UI elements. While adjusting the UI, the GetAgentScriptCallback will make sure that it is updated on the UI thread through the use of Dispatcher.CheckAccess().

UIElement.Dispatcher.CheckAccess() will compare the UI thread ID with that of the worker thread and return true if both threads are the same; otherwise, it returns false. When GetAgentScriptCallback executes on a worker thread (actually, since this will always execute on a worker thread you could simply call Dispatcher.BeginInvoke), CheckAccess() will return false and the UI will be updated by dispatching an anonymous delegate through Dispatcher.Invoke().

Synchronized Service Calls

Because of the asynchronous nature of the Silverlight networking environment, it is almost impossible to make an asynchronous service call on the UI thread and wait for it to complete with the intention of changing the application state based on the results of the call. In **Figure 4**, _notifyCallPopup_OnAccept needs to retrieve order details, transform the output message into a client entity, and save it to a global variable in a thread-safe manner. To accomplish this, one may be tempted to write the handler code as shown here:

Copy Code

```
CallServiceClient client = new CallServiceClient();
client.GetOrderDetailsAsync(orderNumber);
this._orderDetailDownloadHandle.WaitOne();
//do something with the results
```

But this code will freeze the application when it hits the this._orderDetailDownloadHandle.WaitOne() statement. This is because the WaitOne() statement blocks the UI thread from receiving any dispatched messages from other threads. Instead, you can schedule the worker thread to execute the service call, wait for the call to complete, and finish the post processing of the service output in its entirety on the worker thread. This technique is shown in **Figure 5**. To prevent the inadvertent use of blocking calls in the UI thread, I wrapped ManualResetEvent inside a custom SLManualResetEvent and test for UI thread when a call to WaitOne() is made.

Figure 5 Retrieve Order Details

Copy Code

```
void _notifyCallPopup_OnAccept(object sender, EventArgs e)
{
  ...
  ThreadPool.QueueUserWorkItem(ExecuteGetOrderDetails,
        ClientGlobals.callInfo.OrderNumber);
}
private SLManualResetEvent _ orderDetailDownloadHandle = new
        SLManualResetEvent();
  private void ExecuteGetOrderDetails(object state)
{
  CallServiceClient client = new CallServiceClient();
  string orderNumber = state as string;
  client.GetOrderDetailsCompleted += new
        EventHandler<GetOrderDetailsCompletedEventArgs>
        (GetOrderDetailsCompletedCallback);
  client.GetOrderDetailsAsync(orderNumber);
  this._orderDetailDownloadHandle.WaitOne();
  //translate entity and save it to global variable
  ClientEntityTranslator oito = SvcOrderToClientOrder.entityTranslator;
  ClientEntities.Order currentOrder =
        oito.ToClientEntity(ClientGlobals.serviceOutputOrder)
        as ClientEntities.Order;
  Interlocked.Exchange<ClientEntities.Order>(ref ClientGlobals.
       currentOrder, currentOrder);
}

void GetOrderDetailsCompletedCallback(object sender,
        GetOrderDetailsCompletedEventArgs e)
  {
    Interlocked.Exchange<OrderInfo>(ref ClientGlobals.serviceOutputOrder,
         e.Result);
    this._orderDetailDownloadHandle.Set();
  }
```

Since SLManualResetEvent is a general-purpose class, you can't depend on the Dispatcher.CheckAccess() of a particular control. ApplicationHelper.IsUiThread() can check Application.RootVisual.Dispatcher.CheckAccess(); however, access to this method will trigger an invalid cross-thread access exception. So the only reliable way of testing this in a worker thread, when there is no access to a UIElement instance, is to use Deployment.Current.Dispatcher.CheckAccess() as shown here:

Copy Code

```
public static bool IsUiThread()
    {
        if (Deployment.Current.Dispatcher.CheckAccess())
            return true;
        else
            return false;
    }
```

For the background execution of tasks, instead of using ThreadPool.QueueUserWorkItem, you could use BackGroundWorker, which will also use ThreadPool but allows you to wire handlers that can execute on the UI thread. This pattern allows the execution of several service calls in parallel and waits for all the calls to complete using SLManualResetEvent.WaitOne() before the results are aggregated for further processing.

Message Entity Translation

The GetAgentScriptCallback also translates the output message entities (also known as DataContracts) from the service into a client-side entity that represents the client-side usage semantics. For example, the design of server-side message entities may not care about data binding while paying close attention to the multiuse nature of the service that will have to serve a broad range of uses, not only the call center.

Also, it is a good practice not to have tight coupling with the message entities, because changes to the message entities will not be within the client's control. The practice of translating message entities to client-side entities is not just applicable to Silverlight, but is generally applicable to any Web service consumer when wanting to avoid design-time tight coupling.
I decided to keep the implementation of the entity translators very simple—no exotic nested generics, lambda expressions, or inversion of control containers. ClientEntityTranslator is an abstract class that defines the ToClientEntity() method, which every subclass must override:
Copy Code
```
public abstract class ClientEntityTranslator
{
  public abstract ClientEntities.ClientEntity ToClientEntity(object
                                                  serviceOutputEntity);
}
```
Each child class is unique to a service exchange type; hence I will create as many translators as necessary. In my demo, I have made three types of service calls: IUserProfile.GetUser(), ICallService.GetAgentScript(), and ICallService.GetOrderDetails(). So I created three translators, as shown in **Figure 6**.
Figure 6 Message Entity to Client-Side Entity Translator
Copy Code
```
public class SvcOrderToClientOrder : ClientEntityTranslator
{
  //singleton
  public static ClientEntityTranslator entityTranslator = new
                                          SvcOrderToClientOrder();
  private SvcOrderToClientOrder() { }
  public override ClientEntities.ClientEntity ToClientEntity(object
                                                   serviceOutputEntity)
  {
    CallBusinessProxy.OrderInfo oi = serviceOutputEntity as
                                         CallBusinessProxy.OrderInfo;
    ClientEntities.Order bindableOrder = new ClientEntities.Order();
    bindableOrder.OrderNumber = oi.Order.OrderNumber;
    //code removed for brevity  ...
    return bindableOrder;
  }
}

public class SvcUserToClientUser : ClientEntityTranslator
{
    //code removed for brevity  ...
}

public class SvcScriptToClientScript : ClientEntityTranslator
{
    //code removed for brevity  ...
    }
}
```
If you noticed, the above translators are stateless and employ a singleton pattern. The translator must be able to inherit from ClientEntityTranslator for consistency and needs to be a singleton to avoid garbage-collection churn.
I keep reusing the same instance whenever the respective service call is made. I could also create ServiOutputEntityTranslator for service interaction that requires large input messages (which generally is the case for transactional service invocation) with the following class definition:
Copy Code
```
public abstract class ServiOutputEntityTranslator
{
  public abstract object ToServiceOutputEntity(ClientEntity
                                                             clientEntity);
}
```

If you notice the return value of the above function, it is "object", as I don't control the base class of the message entities (in this demo I could, but not in the real world). The type safety will be implemented by the respective translators. For the simplicity of the demo, I don't save any data back to the server, so this demo does not include any translators for converting client entities to message entities.

Silverlight State Change after the Service Calls

Silverlight visual state change can only be performed by the code executing on the UI thread. Since the asynchronous execution of the service calls always returns the results on the callback handler, the handler will be the right place to make changes to the visual or non-visual state of the application.
Non-visual state changes should be exchanged in a thread-safe manner if there might be multiple services trying to modify the shared state asynchronously. It is always recommended that you check Deployment.Current.Dispatcher.CheckAccess() before modifying the UI.

Cross-Domain Policies

Unlike the media applications and the applications that show banner ads, real enterprise-class LOB applications require integration with a variety of service-hosting environments. For example, the call center application referenced throughout the article is typical of the enterprise application. This application hosted on a Web site accesses a stateful socket server for screen-pop, WCF-based Web services for accessing LOB data, and it may download additional XAP packages (Zipped Silverlight deployment packages) from a different domain. It will use yet another domain for transmitting instrumentation data.
The Silverlight sandbox doesn't by default allow network access to any domain other than the domain of origin—advcallclientweb as you saw back in **Figure 1**. The Silverlight runtime checks for the opt-in policies when the application accesses any domain other than the domain of origin. Here is a typical list of the service-hosting scenarios that need to support cross-domain policy requests by the client:

- Web services hosted in a service process (or a console application for simplicity)
- Web services hosted on IIS or other Web servers
- TCP Services hosted in a service process (or a console app)

I discussed cross-domain policy implementation for TCP services last month and hence will focus on Web services hosted in custom processes and inside IIS.
While it is straightforward to implement cross-domain policies for Web service endpoints hosted in IIS, the other cases require knowledge of the nature of the policy requests and responses.

Cross Domain Policies for Web Services Hosted Outside IIS

For effective state management, there may be cases where one may want to host services in an OS process outside IIS. For cross-domain access of such WCF services, the process will have to host policies at the root of the HTTP endpoint. When a cross-domain Web service is invoked, Silverlight makes an HTTP Get request to clientaccesspolicy.xml. If the service is hosted inside IIS, the clientaccesspolicy.xml file can be copied to the root of the Web site and IIS will do the rest in serving the file. In case of custom hosting on the local machine, http://localhost:<port>/clientaccesspolicy.xml should be a valid URL.
Since the call center demo does not use any custom hosted Web services, I will use a simple TimeService in a console application to demonstrate the concepts. The console will expose a representational state transfer (REST) endpoint using the new REST capabilities of the Microsoft .NET Framework 3.5. UriTemplate property has to be precisely set to the literal shown in **Figure 7**.

Figure 7 Implementation for Custom-Hosted WCF Services

Copy Code

```
[ServiceContract]
public interface IPolicyService
{
    [OperationContract]
    [WebInvoke(Method = "GET", UriTemplate = "/clientaccesspolicy.xml")]
    Stream GetClientAccessPolicy();
}
public class PolicyService : IPolicyService
{
    public Stream GetClientAccessPolicy()
    {
        FileStream fs = new FileStream("PolicyFile.xml", FileMode.Open);
        return fs;
    }
}
```

The interface name or the method name has no bearing on the outcome; you can choose anything you like. WebInvoke has other properties such as RequestFormat and ResponseFormat, which are by default set to XML; we don't need to specify them explicitly. We are also relying on the default value of the BodyStyle property to be BodyStyle.Bare, which means that the response won't be wrapped.

The service implementation is very simple; it merely streams the clientaccesspolicy.xml in response to the Silverlight client request. The policy file name can be of your own choosing, and you can call it anything you like. The implementation of the policy service is shown in **Figure 7**.

Now we have to configure the IPolicyService for REST-style serving of HTTP requests. The App.Config of the console application (ConsoleWebServices) is shown in **Figure 8**. There are a few things to note about the special configuration need: the binding of the ConsoleWebServices.IPolicyServer endpoint has to be set to webHttpBinding. Also, the IPolicyService endpoint behavior should be configured with WebHttpBehavior as shown in the configuration file. The base address of the PolicyService should be set to the root URL (as in http://localhost:3045/) and the endpoint address should be left empty (such as <endpoint address=" " ... contract="ConsoleWebServices.IPolicyService" />.

Figure 8 WCF Settings for Custom Hosting Environment

Copy Code

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <!-- IPolicyService end point should be configured with 
           webHttpBinding-->
      <service name="ConsoleWebServices.PolicyService">
         <endpoint address=""
               behaviorConfiguration="ConsoleWebServices.WebHttp"
               binding="webHttpBinding"
               contract="ConsoleWebServices.IPolicyService" />
         <host>
           <baseAddresses>
             <add baseAddress="http://localhost:3045/" />
           </baseAddresses>
         </host>
      </service>
      <service behaviorConfiguration="ConsoleWebServices.TimeServiceBehavior"
               name="ConsoleWebServices.TimeService">
         <endpoint address="TimeService" binding="basicHttpBinding"
               contract="ConsoleWebServices.ITimeService">
         </endpoint>
         <host>
            <baseAddresses>
              <add baseAddress="http://localhost:3045/TimeService.svc" />
            </baseAddresses>
         </host>
       </service>
     </services>
     <behaviors>
```

```
        <endpointBehaviors>
          <!--end point behavior is used by REST endpoints like
              IPolicyService described above-->
          <behavior name="ConsoleWebServices.WebHttp">
            <webHttp />
          </behavior>
        </endpointBehaviors>
       ...
      </behaviors>
    </system.serviceModel>
</configuration>
```

Lastly, the console-hosted services, such as the TimeService shown in the code samples as well as the configuration, should be configured to have a URL that looks similar to their IIS counterparts. For example, the URL of an IIS-hosted TimeService endpoint on default HTTP may look like the following: http://localhost/TimeService.svc. In this case, the metadata can be obtained from http://localhost/TimeService.svc?WSDL.

However, in the case of console hosting, the metadata can be obtained by appending "?WSDL" to the base address of the service host. In the configuration shown in **Figure 8**, you can see that the base address of the TimeService is set to http://localhost:3045/TimeService.svc, hence the metadata can be obtained from http://localhost:3045/TimeService.svc?WSDL.

This URL is similar to what we use in IIS hosting. If you set the host base address to http://localhost:3045/TimeService.svc/, then the metadata URL will be http://localhost:3045/TimeService.svc/?WSDL, which looks a little bit odd. So watch out for this behavior as it can save you time in figuring out the metadata URL.

Cross-Domain Policies for Services Hosted inside IIS

As discussed previously, deploying cross-domain policies for IIS-hosted services is straightforward: you just copy the clientaccesspolicy.xml file to the root of the site on which the Web services are hosted. As you saw in **Figure 1**, the Silverlight app is hosted on advcallclientweb (localhost:1041) and accesses business services from AdvBusinessServices (localhost:1043). The Silverlight runtime requires clientaccesspolicy.xml to be deployed at the root of the AdvBusinessServices Web site with the code shown in **Figure 9**.

Figure 9 Clientaccesspolicy.xml for IIS-Hosted Web Services

Copy Code

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
        <!--allows the access of Silverlight application with localhost:1041
           as the domain of origin-->
        <domain uri="http://localhost:1041"/>
        <!--allows the access of call simulator Silverlight application
           with localhost:1042 as the domain of origin-->
        <domain uri="http://localhost:1042"/>
      </allow-from>
      <grant-to>
        <resource path="/" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

If you recall the cross-domain policy format for the socket server (advpolicyserver) from the first installment of this series, the format of <allow-from> is similar. The difference is in the <grant-to> section where the socket server requires a <socket-resource> setting with port range and protocol attributes, as shown here:

Copy Code

```
<grant-to>
```

```
  <socket-resource port="4530" protocol="tcp" />
</grant-to>
```

If you create the WCF service-hosting site using the ASP.NET Web site template and add WCF endpoints later, the test Web server will map the virtual directory to the name of the project (such as "/AdvBusinessServices"). This should be changed to "/" in the property pages of the project so that the clientaccesspolicy.xml can be served from the root. If you don't change this, the clientaccesspolicy.xml will not be at the root, and Silverlight applications will get server errors when the service is accessed. Note that this will not be a problem for the Web sites created using the WCF Web service project template.

Figure 10 Login Control Using PasswordBox

Copy Code

```
<UserControl x:Class="AdvCallCenterClient.Login">
  <Border x:Name="LayoutRoot" ... >
    <Grid x:Name="gridLayoutRoot">
     <Border x:Name="borderLoginViw" ...>
       <TextBlock Text="Pleae login.." Style="{StaticResource headerStyle}"/>
       <TextBlock Text="Rep ID" Style="{StaticResource labelStyle}"/>
       <TextBox x:Name="txRepID" Style="{StaticResource valueStyle}"/>
       <TextBlock Text="Password" Style="{StaticResource labelStyle}"/>
       <PasswordBox x:Name="pbPassword" PasswordChar="*"/>
       <HyperlinkButton x:Name="hlLogin" Content="Click to login"
            ToolTipService.ToolTip="Clik to login" Click="hlLogin_Click" />
     </Border>
     <TextBlock x:Name="tbLoginStatus" Foreground="Red" ... />
      ...
</UserControl>

public partial class Login : UserControl
{
  public Login()
  {
    InitializeComponent();
  }
  public event EventHandler<EventArgs> OnSuccessfulLogin;
  private void hlLogin_Click(object sender, RoutedEventArgs e)
  {
    //validate the login
    AuthenticationProxy.AuthenticationServiceClient authService
                  = new AuthenticationProxy.AuthenticationServiceClient();
    authService.LoginCompleted += new
                EventHandler< AuthenticationProxy.LoginCompletedEventArgs>
                                           (authService_LoginCompleted);
    authService.LoginAsync(this.txRepID.Text, this.pbPassword.Password,
                                                         null, false);
  }

  void authService_LoginCompleted(object sender,
                           AuthenticationProxy.LoginCompletedEventArgs e)
  {
    if (e.Result == true)
    {
       if (OnSuccessfulLogin != null)
          OnSuccessfulLogin(this, null);
    }
    else
    {
      this.tbLoginStatus.Text = "Invalid user id or password";
    }

  }
}
```

Application Security

One of the critical requirements of an LOB application is authentication; before the call center agent can start the shift, he will authenticate by giving a user ID and password. In ASP.NET Web applications, this can easily be done by taking advantage of the membership provider and the server-side ASP.NET login controls.

In Silverlight, there are two ways to enforce authentication: authentication outside and authentication inside.

Authentication outside is very straightforward and is similar to the authentication implementation of ASP.NET applications. With this approach, authentication happens in an ASP.NET-based Web page before the Silverlight application is displayed. The authentication context can be transferred into the Silverlight application through InitParams parameter before a Silverlight application is loaded or through a custom Web service call (to extract the authentication state information) after the application is loaded.

This approach has its place when the Silverlight application is part of a larger ASP.NET/HTML-based system. However, in cases where Silverlight is the main driver of the application, it is natural to perform authentication inside Silverlight. I will use the Silverlight 2 PasswordBox control to capture the password and authenticate using the ASP.NET AuthenticationService WCF endpoint for validating the user's credentials. AuthenticationService, ProfileService, and RoleService are part of the new namespace—System.Web.ApplicationServices—which was new with the .NET Framework 3.5. **Figure 10** shows the XAML for the Login control created for this purpose. The Login control calls ASP.NET AuthenticationService.LoginAsync() with the user ID and password that was entered.

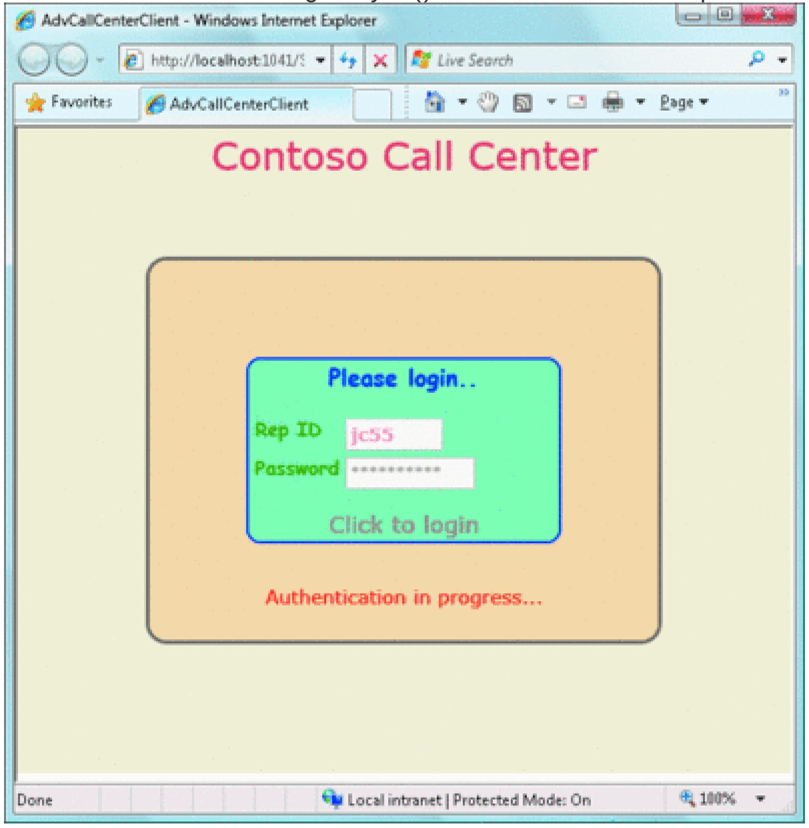


Figure 11 **Login Custom Silverlight Control**

The login screen of the call center, shown in **Figure 11**, is not sophisticated but serves the purpose of the demo. I implemented a handler for dealing with the LoginCompleted event inside the control so that it is self-contained for displaying invalid login messages and password-resetting dialogues for sophisticated implementations. Upon a successful login, the event OnSuccessfulLogin will be triggered to tell the parent

control (Application.RootVisual in this case) to display the first application screen populated with the user information.
The LoginCompleted (ctrlLoginView_OnSuccessfulLogin) handler located inside the main Silverlight Page will invoke the profile service hosted on the business services Web site, as shown in **Figure 12**.
AuthenticationService by default is not mapped to any .svc endpoint; therefore, I will map .svc file to the physical implementation, as shown here:
Copy Code
```
<!-- AuthenticationService.svc -->
<%@ ServiceHost Language="C#" Service="System.Web.ApplicationServices.
    AuthenticationService" %>
```
Figure 12 Usage of Login.xaml inside the Page.xaml
Copy Code
```
<!-- Page.xaml of the main UserControl attached to RootVisual-->
<UserControl x:Class="AdvCallCenterClient.Page" ...>
   <page:Login x:Name="ctrlLoginView" Visibility="Visible"
         OnSuccessfulLogin="ctrlLoginView_OnSuccessfulLogin"/>
   ...
</UserControl>
<!-- Page.xaml.cs of the main UserControl attached to RootVisual-->
public partial class Page : UserControl
{
   ...

   private void ctrlLoginView_OnSuccessfulLogin(object sender, EventArgs e)
   {
     Login login = sender as Login;
     login.Visibility = Visibility.Collapsed;
     CallBusinessProxy.UserProfileClient userProfile
                           = new CallBusinessProxy.UserProfileClient();
     userProfile.GetUserCompleted += new
     EventHandler<GetUserCompletedEventArgs>(userProfile_GetUserCompleted);
     userProfile.GetUserAsync(login.txRepID.Text);
   }
   ...
   void userProfile_GetUserCompleted(object sender,
                                            GetUserCompletedEventArgs e)
   {
     CallBusinessProxy.User user = e.Result;
     UserToBindableUser utobu = new UserToBindableUser(user);
     ClientGlobals.currentUser = utobu.Translate() as ClientEntities.User;
     //all the time the service calls will be complete on a worker thread
     //so the following check is redunant but done to be safe
     if (!this.Dispatcher.CheckAccess())
     {
       this.Dispatcher.BeginInvoke(delegate()
       {
         this.registrationView.DataContext = ClientGlobals.currentUser;
         this.ctrlLoginView.Visibility = Visibility.Collapsed;
         this.registrationView.Visibility = Visibility.Visible;
       });
      }
    }
}
```
Silverlight can only call Web services that are configured to be called by scripting environments such as AJAX. Like all AJAX callable services, the AuthenticationService service needs access to the ASP.NET runtime environment. I provide this access by setting <serviceHostingEnvironment

aspNetCompatibilityEnabled="true"/> directly under the <system.servicemodel> node. In order for the authentication service to be callable by the Silverlight login process (or to be called by AJAX), the web.config must be set according to the directions in "How to: Enable the WCF Authentication Service." The services will be automatically configured for Silverlight if they are created using the Silverlight-enabled WCF Service template located in the Silverlight category.

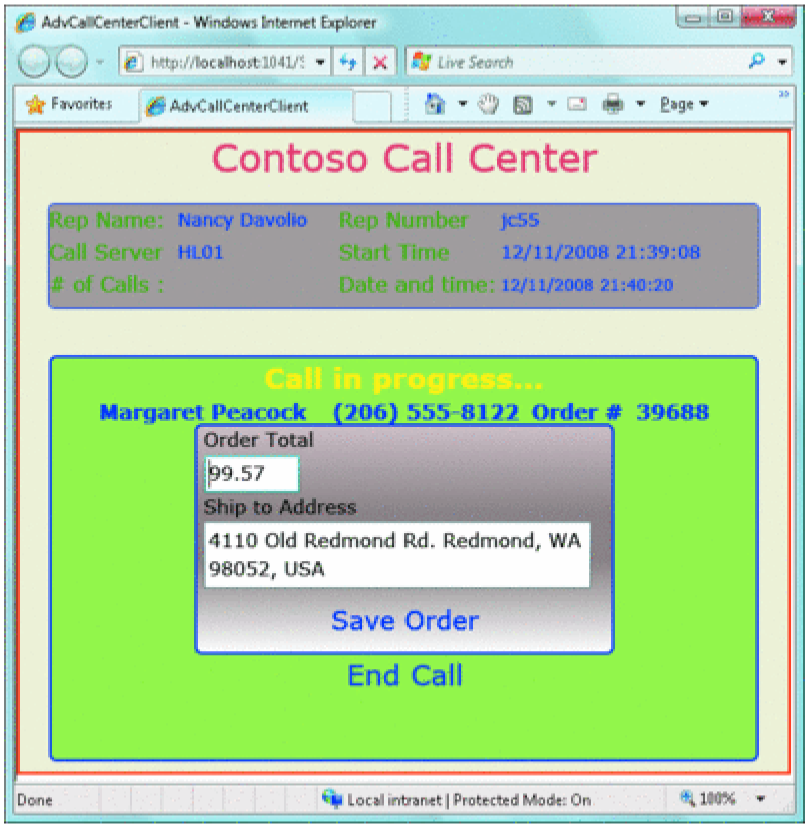
Figure 13 shows the edited configuration with important elements necessary for the Authentication service. In addition to the service configuration, I also replaced the SQL Server configuration setting for aspnetdb that stores authentication information. Machine.config defines a LocalSqlServer setting that expects asp-netdb.mdf to be embedded into the App_Data directory of the Web site. This configuration setting removes the default setting and points it to the aspnetdb attached to the SQL Server instance. This can easily be changed to point to a database instance running on a separate machine.

Figure 13 Settings for ASP.NET Authentication Service

Copy Code

```
//web.config
<Configuration>
  <connectionStrings>
  <!-- removal and addition of LocalSqlServer setting will override the
   default asp.net security database used by the ASP.NET Configuration tool
   located in the Visul Studio Project menu-->
  <remove name="LocalSqlServer"/>
    <add name="LocalSqlServer" connectionString="Data
             Source=localhost\SqlExpress;Initial Catalog=aspnetdb; ... />
</connectionStrings>
<system.web.extensions>
   <scripting>
     <webServices>
   <authenticationService enabled="true" requireSSL="false"/>
     </webServices>
   </scripting>
</system.web.extensions>
...
<authentication mode="Forms"/>
...
<system.serviceModel>
   <services>
     <service name="System.Web.ApplicationServices.AuthenticationService"
              behaviorConfiguration="CommonServiceBehavior">
    <endpoint
              contract="System.Web.ApplicationServices.AuthenticationService"
              binding="basicHttpBinding" bindingConfiguration="useHttp"
              bindingNamespace="http://asp.net/ApplicationServices/v200"/>
     </service>
   </services>
   <bindings>
     <basicHttpBinding>
    <binding name="useHttp">
          <!--for production use mode="Transport" -->
      <security mode="None"/>
     </binding>
     </basicHttpBinding>
   </bindings>
   ...
   <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
</system.serviceModel>
</configuration>
```

To preserve encapsulation of the Login control and to maintain design-time loose coupling with the parent control, success of the login process is communicated by triggering the OnSuccessfulLogin event. The Application.RootVisual (which is a Page class) will execute the necessary business process to display the first screen upon successful login. The first screen displayed after a successful login is the registrationView, as shown in the userProfile_GetUserCompleted method of **Figure 12**. Before this view is displayed, I will retrieve user information by calling CallBusinessProxy.UserProfileClient.GetUserAsync(). Please take note of the asynchronous service call, which is similar to the business service integration I'll discuss later.
Be aware that the previous configuration does not use secure sockets layer (SSL); you must modify it to use SSL when building for production systems.



Contoso Call Center

Rep Name: Nancy Davolio   Rep Number jc55
Call Server HL01   Start Time 12/11/2008 21:39:08
# of Calls :   Date and time: 12/11/2008 21:40:20

Call in progress...
Margaret Peacock (206) 555-8122 Order # 39688
Order Total
99.57
Ship to Address
4110 Old Redmond Rd. Redmond, WA 98052, USA

Figure 14 **OrderDetails.xaml Control with Order Details Populated**

Application Partitioning

One of the factors contributing to the Silverlight application startup time is the size of the initial package. The guidelines for the size of the XAP package are no different than the page weight for Web applications. Bandwidth is a limited resource. The stringent response times of Web applications require that you pay close attention to the Silverlight application startup time.
In addition to the processing time spent before the first UserControl is displayed, the size of the application package has direct influence on this important quality of the application. In order to improve startup speed, you must avoid monolithic XAP files that can grow to tens of megabytes in size for complex applications.
The Silverlight application can be broken down into a collection of XAP files; individual DLLs; or individual XML files, images, and any other types with recognized MIME types. In the call center scenario, to demonstrate granular application partitioning, I will deploy the OrderDetail Silverlight control as a separate DLL (AdvOrderClientControls.dll) along with AdvCallCenterClient.xap into the ClientBin directory of the AdvCallClientWeb project (refer back to **Figure 1**).
The DLL will be downloaded preemptively on the worker thread when the agent accepts the incoming call. The call you saw in **Figure 4**—ThreadPool.QueueUserWorkItem(ExecuteControlDownload)—is responsible for this. Once the caller answers security questions, I will use reflection to create an instance of the OrderDetail

control and add it to the control tree before displaying it on the screen. **Figure 14** shows OrderDetail.xaml control loaded into the control tree with order details populated.

The DLL containing the OrderDetail control is deployed to the same Web site as the call center client, which is typical of the DLLs belonging to the same application, so I won't run into any cross-domain issues in this case. However, this may not be the case with services, because Silverlight applications may access services deployed on multiple domains, including local ones and in the cloud, as shown in the architecture diagram (again, refer back to **Figure 1**).

The ExecuteControlDownload method (see **Figure 4**) runs on a background worker thread and uses the WebClient class for downloading the DLL. WebClient, by default, assumes that the download happens from the domain of origin and hence only uses relative URIs.

The OrderDetailControlDownloadCallback handler receives the DLL stream and creates the assembly using ResourceUtility.GetAssembly() shown in **Figure 15**. Because creation of the assembly must happen on the UI thread, I will dispatch the GetAssembly() and the (thread-safe) assignment of the assembly to the global variable to the UI thread:

Copy Code
```
void OrderDetailControlDownloadCallback(object sender,
       OpenReadCompletedEventArgs e)
  {
    this.Dispatcher.BeginInvoke(delegate() {
    Assembly asm = ResourceUtility.GetAssembly(e.Result);
    Interlocked.Exchange<Assembly>(ref
        ClientGlobals.advOrderControls_dll, asm ); });
  }
```

Figure 15 Utility Functions to Extract Resources

Copy Code
```
public class ResourceUtility
{
  //helper function to retrieve assembly from a package stream
  public static Assembly GetAssembly(string assemblyName, Stream
                                                        packageStream)
  {
    StreamResourceInfo srInfo =
    Application.GetResourceStream(
              new StreamResourceInfo(packageStream, "application/binary"),
              new Uri(assemblyName, UriKind.Relative));
    return GetAssembly(srInfo.Stream);
  }
  //helper function to retrieve assembly from a assembly stream
  public static Assembly GetAssembly(Stream assemblyStream)
  {
    AssemblyPart assemblyPart = new AssemblyPart();
    return assemblyPart.Load(assemblyStream);
  }
  //helper function to create an XML document from the stream
  public static XElement GetXmlDocument(Stream xmlStream)
  {
    XmlReader reader = XmlReader.Create(xmlStream);
    XElement element = XElement.Load(reader);
    return element;
  }
  //helper function to create an XML document from the default package
  public static XElement GetXmlDocumentFromXap(string fileName)
  {
    XmlReaderSettings settings = new XmlReaderSettings();
    settings.XmlResolver = new XmlXapResolver();
    XmlReader reader = XmlReader.Create(fileName);
    XElement element = XElement.Load(reader);
    return element;
  }
  //gets the UIElement from the default package
  public static UIElement GetUIElementFromXaml(string xamlFileName)
  {
    StreamResourceInfo streamInfo = Application.GetResourceStream(new
                              Uri(xamlFileName, UriKind.Relative));
    string xaml = new StreamReader(streamInfo.Stream).ReadToEnd();
    UIElement uiElement = null;
    try
    {
      uiElement = (UIElement)XamlReader.Load(xaml);
    }
    catch
    {
      throw new SLApplicationException(string.Format("Can't create
                              UIElement from {0}", xamlFileName));
    }
    return uiElement;
  }
}
```

Since the dispatched delegate runs on a different thread than the callback handler, you have to be conscious of the state of the objects that are accessed from the anonymous delegate. In the previous code, the state of the downloaded DLL stream is really important. You may not write code that reclaims the resources of the stream inside the OrderDetailControlDownloadCallback function. Such code will prematurely dispose of the downloaded stream before the UI thread has a chance to create the assembly. I will use reflection to create an instance of the OrderDetail user control and add it to the Panel as shown here:

Copy Code

```
_orderDetailContol = ClientGlobals.advOrderControls_dll.CreateInstance
                  ("AdvOrderClientControls.OrderDetail") as UserControl;
spCallProgressPanel.Children.Add(_orderDetailContol);
```

ResourceUtility in **Figure 15** also shows various utility functions to extract UIElement from the XAML and the XML document from the downloaded streams and default packages.

Productivity and Beyond

I have looked at Silverlight from a traditional enterprise application perspective, touching several architectural aspects of the application. Implementation of push notifications with Silverlight sockets is an enabler of LOB scenarios such as call centers. With the upcoming release of Internet Explorer 8.0—which is slated to include six concurrent HTTP connections per host—push notification implementation over the Internet will be more compelling when using duplex WCF binding. Integration with LOB data and processes is as easy as it is in traditional desktop applications.

This will be a huge productivity boost when compared with AJAX and other rich Internet application (RIA) platforms. Silverlight applications can be secured using the WCF authentication and authorization endpoints provided by ASP.NET in its latest release. I hope this little exploration of LOB application development with Silverlight will motivate you to take Silverlight beyond media and advertising scenarios.

Hanu Kommalapati is a Platform Strategy Advisor at Microsoft, and in this role he advises enterprise customers in building scalable line-of-business applications on Silverlight and Azure Services platforms.