Chapter 22: Web Services in C#

Internet & World Wide Web
How to Program, 5/e

Note: This chapter is a copy of Chapter 28 of our book *Visual C# 2010 How to Program*. For that reason, we simply copied the PowerPoint slides for this chapter and *did not* re-numb er them

OBJECTIVES

In this chapter you'll learn:

- How to create WCF web services.
- How XML, JSON, XML-Based Simple Object Access Protocol (SOAP) and Representational State Transfer Architecture (REST) enable WCF web services.
- The elements that comprise WCF web services, such as service references, service endpoints, service contracts and service bindings.
- How to create a client that consumes a WCF web service.
- How to use WCF web services with Windows and web applications.
- How to use session tracking in WCF web services to maintain state information for the client.
- How to pass user-defined types to a WCF web service.

```
1  // Fig. 28.1: IWelcomeSOAPXMLService.cs
2  // WCF web service interface that returns a welcome message through SOAP
3  // protocol and XML data format.
4  using System.ServiceModel;
5
6  [ServiceContract]
7  public interface IWelcomeSOAPXMLService
8  {
9     // returns a welcome message
10    [OperationContract]
11    string Welcome( string yourName );
12 } // end interface IWelcomeSOAPXMLService
```

Fig. 28.1 | WCF web-service interface that returns a welcome message through SOAP protocol and XML format.

```
1  // Fig. 28.2: WelcomeSOAPXMLService.cs
2  // WCF web service that returns a welcome message using SOAP protocol and
3  // XML data format.
4  public class WelcomeSOAPXMLService : IWelcomeSOAPXMLService
5  {
6     // returns a welcome message
7     public string Welcome( string yourName )
8     {
9        return string.Format(
10          "Welcome to WCF Web Services with SOAP and XML, {0}!",
11          yourName );
12    } // end method Welcome
13 } // end class WelcomeSOAPXMLService
```

Fig. 28.2 | WCF web service that returns a welcome message through the SOAP protocol and XML format.

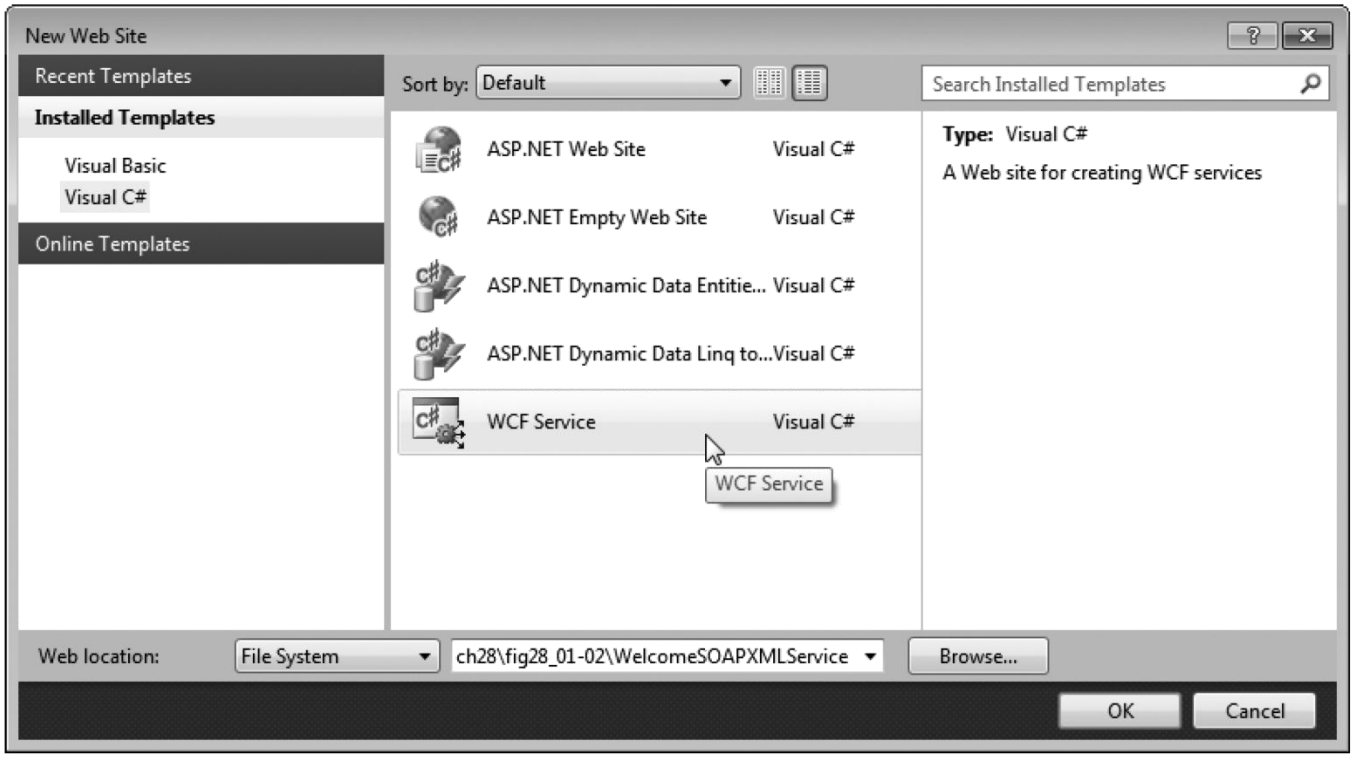


Fig. 28.3 | Creating a WCF Service in Visual Web Developer.

Fig. 28.4 | SVC file rendered in a web browser.

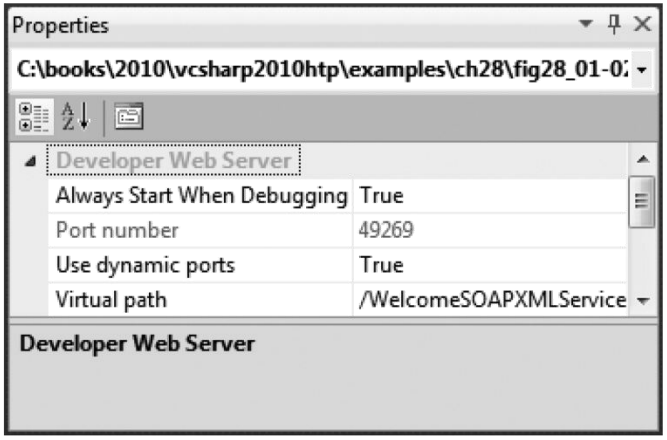


Fig. 28.5 | WCF web service **Properties** window.

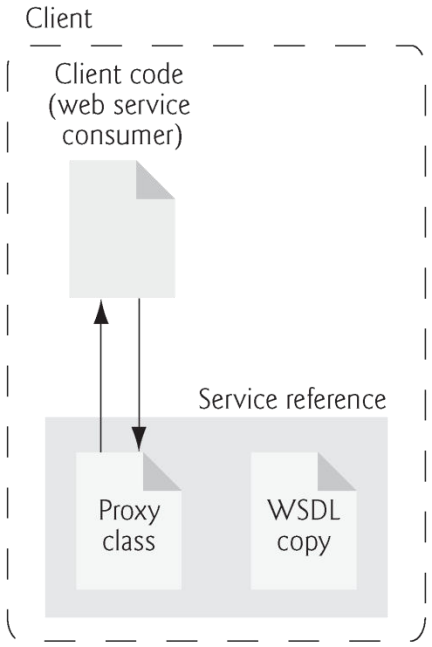


Fig. 28.6 | .NET WCF web-service client after a web-service reference has been added.

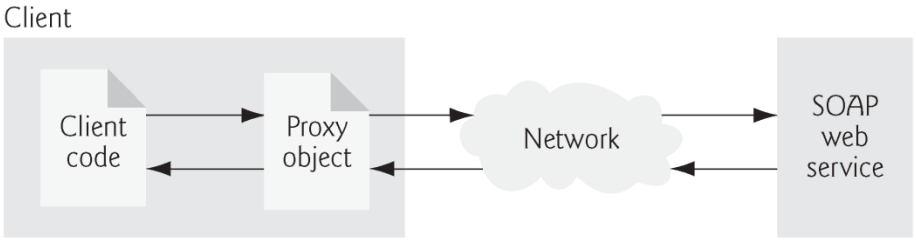
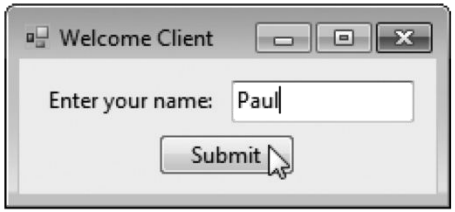


Fig. 28.7 | Interaction between a web-service client and a SOAP web service.

```
1  // Fig. 28.8: WelcomeSOAPXML.cs
2  // Client that consumes the WelcomeSOAPXMLService.
3  using System;
4  using System.Windows.Forms;
5
6  namespace WelcomeSOAPXMLClient
7  {
8     public partial class WelcomeSOAPXML : Form
9     {
10       // declare a reference to web service
11       private ServiceReference.WelcomeSOAPXMLServiceClient client;
12
13       public WelcomeSOAPXML()
14       {
15          InitializeComponent();
16          client = new ServiceReference.WelcomeSOAPXMLServiceClient();
17       } // end constructor
18
19       // creates welcome message from text input and web service
20       private void submitButton_Click( object sender, EventArgs e )
21       {
22          MessageBox.Show( client.Welcome( textBox.Text ), "Welcome" );
23       } // end method submitButton_Click
24    } // end class WelcomeSOAPXML
25 } // end namespace WelcomeSOAPXMLClient
```

Fig. 28.8 | Client that consumes the `WelcomeSOAPXMLService`.

a) User inputs name and clicks **Submit** to send it to the web service



b) Message returned by the web service

Fig. 28.8 | Client that consumes the `WelcomeSOAPXMLService`.

```
1  // Fig. 28.9: IWelcomeRESTXMLService.cs
2  // WCF web service interface. A class that implements this interface
3  // returns a welcome message through REST architecture and XML data format
4  using System.ServiceModel;
5  using System.ServiceModel.Web;
6
7  [ServiceContract]
8  public interface IWelcomeRESTXMLService
9  {
10    // returns a welcome message
11    [OperationContract]
12    [WebGet( UriTemplate = "/welcome/{yourName}" )]
13    string Welcome( string yourName );
14 } // end interface IWelcomeRESTXMLService
```

Fig. 28.9 | WCF web-service interface. A class that implements this interface returns a welcome message through REST architecture and XML data format.

```
1  // Fig. 28.10: WelcomeRESTXMLService.cs
2  // WCF web service that returns a welcome message using REST architecture
3  // and XML data format.
4  public class WelcomeRESTXMLService : IWelcomeRESTXMLService
5  {
6     // returns a welcome message
7     public string Welcome( string yourName )
8     {
9        return string.Format( "Welcome to WCF Web Services"
10          + " with REST and XML, {0}!", yourName );
11    } // end method Welcome
12 } // end class WelcomeRESTXMLService
```

Fig. 28.10 | WCF web service that returns a welcome message using REST architecture and XML data format.

```
<system.serviceModel>
   <behaviors>
      <serviceBehaviors>
         <behavior>
            <!-- To avoid disclosing metadata information, set the
               value below to false and remove the metadata
               endpoint above before deployment -->
            <serviceMetadata httpGetEnabled="true"/>
            <!-- To receive exception details in faults for debugging
               purposes, set the value below to true.  Set to false
               before deployment to avoid disclosing exception
               information -->
            <serviceDebug includeExceptionDetailInFaults="false"/>
         </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
         <behavior>
            <webHttp/>
         </behavior>
      </endpointBehaviors>
   </behaviors>
```

Fig. 28.11 | `WelcomeRESTXMLService` `Web.config` file. (Part 1 of 2.)

```
22      <protocolMapping>
23         <add scheme="http" binding="webHttpBinding"/>
24      </protocolMapping>
25      <serviceHostingEnvironment multipleSiteBindingsEnabled="true"/>
26   </system.serviceModel>
```

Fig. 28.11 | WelcomeRESTXMLService Web.config file. (Part 2 of 2.)

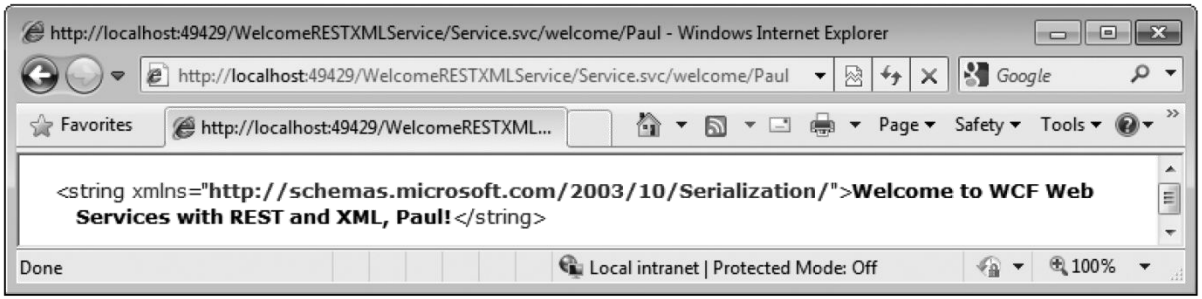


Fig. 28.12 | Response from `WelcomeRESTXMLService` in XML data format.

```
1  // Fig. 28.13: WelcomeRESTXML.cs
2  // Client that consumes the WelcomeRESTXMLService.
3  using System;
4  using System.Net;
5  using System.Windows.Forms;
6  using System.Xml.Linq;
7
8  namespace WelcomeRESTXMLClient
9  {
10    public partial class WelcomeRESTXML : Form
11    {
12       // object to invoke the WelcomeRESTXMLService
13       private WebClient client = new WebClient();
14
15       private XNamespace xmlNamespace = XNamespace.Get(
16          "http://schemas.microsoft.com/2003/10/Serialization/" );
17
```

Fig. 28.13 | Client that consumes the `WelcomeRESTXMLService`. (Part 1 of 4.)

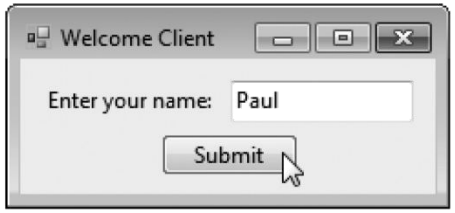
```
18         public WelcomeRESTXML()
19         {
20            InitializeComponent();
21
22            // add DownloadStringCompleted event handler to WebClient
23            client.DownloadStringCompleted +=
24               new DownloadStringCompletedEventHandler(
25               client_DownloadStringCompleted );
26         } // end constructor
27
28         // get user input and pass it to the web service
29         private void submitButton_Click( object sender, EventArgs e )
30         {
31            // send request to WelcomeRESTXMLService
32            client.DownloadStringAsync( new Uri(
33               "http://localhost:49429/WelcomeRESTXMLService/Service.svc/" +
34               "welcome/" + textBox.Text ) );
35         } // end method submitButton_Click
36
```

Fig. 28.13 | Client that consumes the `WelcomeRESTXMLService`. (Part 2 of 4.)

```
37          // process web service response
38          private void client_DownloadStringCompleted(
39             object sender, DownloadStringCompletedEventArgs e )
40          {
41             // check if any error occurred in retrieving service data
42             if ( e.Error == null )
43             {
44                // parse the returned XML string (e.Result)
45                XDocument xmlResponse = XDocument.Parse( e.Result );
46
47                // get the <string> element's value
48                MessageBox.Show( xmlResponse.Element(
49                   xmlNamespace + "string" ).Value, "Welcome" );
50             } // end if
51          } // end method client_DownloadStringCompleted
52       } // end class WelcomeRESTXML
53    } // end namespace WelcomeRESTXMLClient
```

Fig. 28.13 | Client that consumes the `WelcomeRESTXMLService`. (Part 3 of 4.)

a) User inputs name



b) Message sent from **WelcomeRESTXMLService**

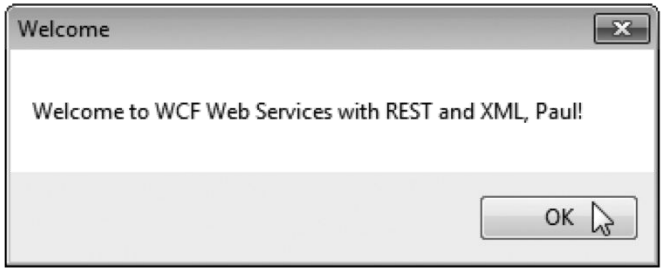


Fig. 28.13 | Client that consumes the `WelcomeRESTXMLService`. (Part 4 of 4.)

```
1  // Fig. 28.14: IWelcomeRESTJSONService.cs
2  // WCF web service interface that returns a welcome message through REST
3  // architecture and JSON format.
4  using System.Runtime.Serialization;
5  using System.ServiceModel;
6  using System.ServiceModel.Web;
7
8  [ServiceContract]
9  public interface IWelcomeRESTJSONService
10 {
11    // returns a welcome message
12    [OperationContract]
13    [WebGet( ResponseFormat = WebMessageFormat.Json,
14       UriTemplate = "/welcome/{yourName}" )]
15    TextMessage Welcome( string yourName );
16 } // end interface IWelcomeRESTJSONService
17
```

Fig. 28.14 | WCF web-service interface that returns a welcome message through REST architecture and JSON format. (Part 1 of 2.)

```
18  // class to encapsulate a string to send in JSON format
19  [DataContract]
20  public class TextMessage
21  {
22     // automatic property message
23     [DataMember]
24     public string Message {get; set; }
25  } // end class TextMessage
```

Fig. 28.14 | WCF web-service interface that returns a welcome message through REST architecture and JSON format. (Part 2 of 2.)

```
1  // Fig. 28.15: WelcomeRESTJSONService.cs
2  // WCF web service that returns a welcome message through REST
3  // architecture and JSON format.
4  public class WelcomeRESTJSONService : IWelcomeRESTJSONService
5  {
6     // returns a welcome message
7     public TextMessage Welcome( string yourName )
8     {
9        // add welcome message to field of TextMessage object
10       TextMessage message = new TextMessage();
11       message.Message = string.Format(
12          "Welcome to WCF Web Services with REST and JSON, {0}!",
13          yourName );
14       return message;
15    } // end method Welcome
16 } // end class WelcomeRESTJSONService
```

Fig. 28.15 | WCF web service that returns a welcome message through REST architecture and JSON format.

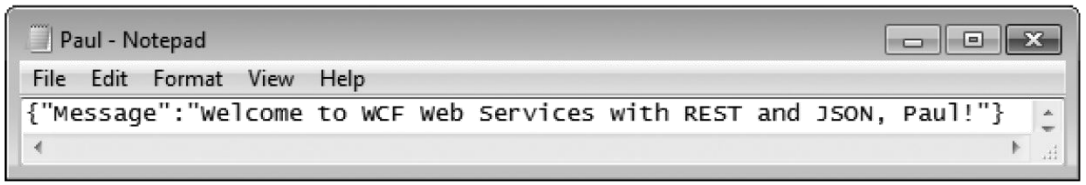


Fig. 28.16 | Response from `WelcomeRESTJSONService` in JSON data format.

```
1  // Fig. 28.17: WelcomeRESTJSONForm.cs
2  // Client that consumes the WelcomeRESTJSONService.
3  using System;
4  using System.IO;
5  using System.Net;
6  using System.Runtime.Serialization.Json;
7  using System.Text;
8  using System.Windows.Forms;
9
10 namespace WelcomeRESTJSONClient
11 {
12    public partial class WelcomeRESTJSONForm : Form
13    {
14       // object to invoke the WelcomeRESTJSONService
15       private WebClient client = new WebClient();
16
```

Fig. 28.17 | Client that consumes the `WelcomeRESTJSONService`. (Part 1 of 4.)

```
17         public WelcomeRESTJSONForm()
18         {
19            InitializeComponent();
20
21            // add DownloadStringCompleted event handler to WebClient
22            client.DownloadStringCompleted+=
23               new DownloadStringCompletedEventHandler(
24                  client_DownloadStringCompleted );
25         } // end constructor
26
27         // get user input and pass it to the web service
28         private void submitButton_Click( object sender, EventArgs e )
29         {
30            // send request to WelcomeRESTJSONService
31            client.DownloadStringAsync( new Uri(
32               "http://localhost:49579/WelcomeRESTJSONService/Service.svc/"
33               + "welcome/" + textBox.Text ) );
34         } // end method submitButton_Click
35
```

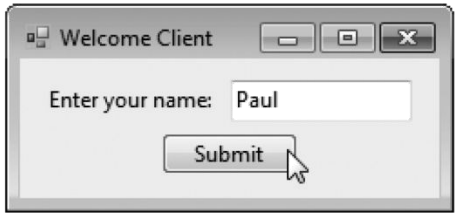
Fig. 28.17 | Client that consumes the `WelcomeRESTJSONService`. (Part 2 of 4.)

```
36          // process web service response
37          private void client_DownloadStringCompleted(
38             object sender, DownloadStringCompletedEventArgs e )
39          {
40             // check if any error occurred in retrieving service data
41             if ( e.Error == null )
42             {
43                // deserialize response into a TextMessage object
44                DataContractJsonSerializer JSONSerializer =
45                   new DataContractJsonSerializer( typeof( TextMessage ) );
46                TextMessage message =
47                   ( TextMessage ) JSONSerializer.ReadObject( new
48                   MemoryStream( Encoding.Unicode.GetBytes( e.Result ) ) );
49
50                // display Message text
51                MessageBox.Show( message.Message, "Welcome" );
52             } // end if
53          } // end method client_DownloadStringCompleted
54       } // end class WelcomeRESTJSONForm
55
```

Fig. 28.17 | Client that consumes the `WelcomeRESTJSONService`. (Part 3 of 4.)

```
56      // TextMessage class representing a JSON object
57      [Serializable]
58      public class TextMessage
59      {
60         public string Message;
61      } // end class TextMessage
62   } // end namespace WelcomeRESTJSONClient
```

a) User inputs name.



b) Message sent from `WelcomeRESTJSONService`.

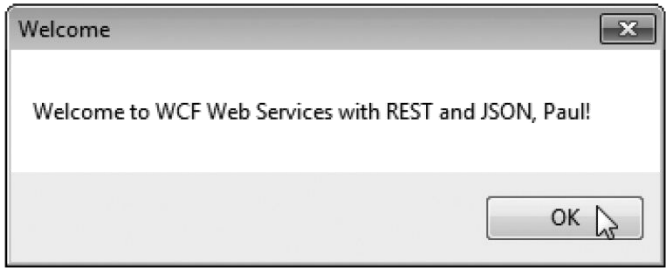


Fig. 28.17 | Client that consumes the `WelcomeRESTJSONService`. (Part 4 of 4.)

```
1  // Fig. 28.18: IBlackjackService.cs
2  // Blackjack game WCF web service interface.
3  using System.ServiceModel;
4
5  [ServiceContract( SessionMode = SessionMode.Required )]
6  public interface IBlackjackService
7  {
8     // deals a card that has not been dealt
9     [OperationContract]
10    string DealCard();
11
12    // creates and shuffle the deck
13    [OperationContract]
14    void Shuffle();
15
16    // calculates value of a hand
17    [OperationContract]
18    int GetHandValue( string dealt );
19 } // end interface IBlackjackService
```

Fig. 28.18 | Blackjack game WCF web-service interface.

```
1  // Fig. 28.19: BlackjackService.cs
2  // Blackjack game WCF web service.
3  using System;
4  using System.Collections.Generic;
5  using System.ServiceModel;
6
7  [ServiceBehavior( InstanceContextMode = InstanceContextMode.PerSession )]
8  public class BlackjackService : IBlackjackService
9  {
10    // create persistent session deck of cards object
11    List< string > deck = new List< string >();
12
13    // deals card that has not yet been dealt
14    public string DealCard()
15    {
16       string card = deck[ 0 ]; // get first card
17       deck.RemoveAt( 0 ); // remove card from deck
18       return card;
19    } // end method DealCard
20
```

Fig. 28.19 | Blackjack game WCF web service. (Part 1 of 4.)

```
21      // creates and shuffles a deck of cards
22      public void Shuffle()
23      {
24         Random randomObject = new Random(); // generates random numbers
25
26         deck.Clear(); // clears deck for new game
27
28         // generate all possible cards
29         for ( int face = 1; face <= 13; face++ ) // loop through faces
30            for ( int suit = 0; suit <= 3; suit++ ) // loop through suits
31               deck.Add( face + " " + suit ); // add card (string) to deck
32
33         // shuffles deck by swapping each card with another card randomly
34         for ( int i = 0; i < deck.Count; i++ )
35         {
36            // get random index
37            int newIndex = randomObject.Next( deck.Count - 1 );
38
39            // save current card in temporary variable
40            string temporary = deck[ i ];
41            deck[ i ] = deck[ newIndex ]; // copy randomly selected card
42
43            // copy current card back into deck
44            deck[ newIndex ] = temporary;
45         } // end for
46      } // end method Shuffle
```

Fig. 28.19 | Blackjack game WCF web service. (Part 2 of 4.)

```
47
48      // computes value of hand
49      public int GetHandValue( string dealt )
50      {
51         // split string containing all cards
52         string[] cards = dealt.Split( '\t' ); // get array of cards
53         int total = 0; // total value of cards in hand
54         int face; // face of the current card
55         int aceCount = 0; // number of aces in hand
56
57         // loop through the cards in the hand
58         foreach ( var card in cards )
59         {
60            // get face of card
61            face = Convert.ToInt32(
62               card.Substring( 0, card.IndexOf( ' ' ) ) );
63
64            switch ( face )
65            {
66               case 1: // if ace, increment aceCount
67                  ++aceCount;
68                  break;
```

Fig. 28.19 | Blackjack game WCF web service. (Part 3 of 4.)

```
            case 11: // if jack add 10
            case 12: // if queen add 10
            case 13: // if king add 10
               total += 10;
               break;
            default: // otherwise, add value of face
               total += face;
               break;
         } // end switch
      } // end foreach

      // if there are any aces, calculate optimum total
      if ( aceCount > 0 )
      {
         // if it is possible to count one ace as 11, and the rest
         // as 1 each, do so; otherwise, count all aces as 1 each
         if ( total + 11 + aceCount - 1 <= 21 )
            total += 11 + aceCount - 1;
         else
            total += aceCount;
      } // end if

      return total;
   } // end method GetHandValue
} // end class BlackjackService
```

Fig. 28.19 | Blackjack game WCF web service. (Part 4 of 4.)

```
1  // Fig. 28.20: Blackjack.cs
2  // Blackjack game that uses the BlackjackService web service.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6  using System.Collections.Generic;
7  using System.Resources;
8
9  namespace BlackjackClient
10 {
11    public partial class Blackjack : Form
12    {
13       // reference to web service
14       private ServiceReference.BlackjackServiceClient dealer;
15
16       // string representing the dealer's cards
17       private string dealersCards;
18
19       // string representing the player's cards
20       private string playersCards;
21
22       // list of PictureBoxes for card images
23       private List< PictureBox > cardBoxes;
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 1 of 18.)

```
24          private int currentPlayerCard; // player's current card number
25          private int currentDealerCard; // dealer's current card number
26
27          private ResourceManager pictureLibrary =
28             BlackjackClient.Properties.Resources.ResourceManager;
29
30          // enum representing the possible game outcomes
31          public enum GameStatus
32          {
33             PUSH, // game ends in a tie
34             LOSE, // player loses
35             WIN, // player wins
36             BLACKJACK // player has blackjack
37          } // end enum GameStatus
38
39          public Blackjack()
40          {
41             InitializeComponent();
42          } // end constructor
43
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 2 of 18.)

```
44          // sets up the game
45          private void Blackjack_Load( object sender, EventArgs e )
46          {
47             // instantiate object allowing communication with web service
48             dealer = new ServiceReference.BlackjackServiceClient();
49
50             // put PictureBoxes into cardBoxes List
51             cardBoxes = new List<PictureBox>(); // create list
52             cardBoxes.Add( pictureBox1 );
53             cardBoxes.Add( pictureBox2 );
54             cardBoxes.Add( pictureBox3 );
55             cardBoxes.Add( pictureBox4 );
56             cardBoxes.Add( pictureBox5 );
57             cardBoxes.Add( pictureBox6 );
58             cardBoxes.Add( pictureBox7 );
59             cardBoxes.Add( pictureBox8 );
60             cardBoxes.Add( pictureBox9 );
61             cardBoxes.Add( pictureBox10 );
62             cardBoxes.Add( pictureBox11 );
63             cardBoxes.Add( pictureBox12 );
64             cardBoxes.Add( pictureBox13 );
65             cardBoxes.Add( pictureBox14 );
66             cardBoxes.Add( pictureBox15 );
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 3 of 18.)

```
67            cardBoxes.Add( pictureBox16 );
68            cardBoxes.Add( pictureBox17 );
69            cardBoxes.Add( pictureBox18 );
70            cardBoxes.Add( pictureBox19 );
71            cardBoxes.Add( pictureBox20 );
72            cardBoxes.Add( pictureBox21 );
73            cardBoxes.Add( pictureBox22 );
74         } // end method Blackjack_Load
75
76         // deals cards to dealer while dealer's total is less than 17,
77         // then computes value of each hand and determines winner
78         private void DealerPlay()
79         {
80            // reveal dealer's second card
81            string[] cards = dealersCards.Split( '\t' );
82            DisplayCard( 1, cards[1] );
83
84            string nextCard;
85
86            // while value of dealer's hand is below 17,
87            // dealer must take cards
88            while ( dealer.GetHandValue( dealersCards ) < 17 )
89            {
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 4 of 18.)

```
90              nextCard = dealer.DealCard(); // deal new card
91              dealersCards += '\t' + nextCard; // add new card to hand
92
93              // update GUI to show new card
94              MessageBox.Show( "Dealer takes a card" );
95              DisplayCard( currentDealerCard, nextCard );
96              ++currentDealerCard;
97           } // end while
98
99           int dealersTotal = dealer.GetHandValue( dealersCards );
100          int playersTotal = dealer.GetHandValue( playersCards );
101
102          // if dealer busted, player wins
103          if ( dealersTotal > 21 )
104          {
105             GameOver( GameStatus.WIN );
106          } // end if
107          else
108          {
109             // if dealer and player have not exceeded 21,
110             // higher score wins; equal scores is a push.
111             if ( dealersTotal > playersTotal ) // player loses game
112                GameOver( GameStatus.LOSE );
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 5 of 18.)

```
113             else if ( playersTotal > dealersTotal ) // player wins game
114                GameOver( GameStatus.WIN );
115             else // player and dealer tie
116                GameOver( GameStatus.PUSH );
117          } // end else
118       } // end method DealerPlay
119
120       // displays card represented by cardValue in specified PictureBox
121       public void DisplayCard( int card, string cardValue )
122       {
123          // retrieve appropriate PictureBox
124          PictureBox displayBox = cardBoxes[ card ];
125
126          // if string representing card is empty,
127          // set displayBox to display back of card
128          if ( string.IsNullOrEmpty( cardValue ) )
129          {
130             displayBox.Image =
131                ( Image ) pictureLibrary.GetObject( "cardback" );
132             return;
133          } // end if
134
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 6 of 18.)

```
135          // retrieve face value of card from cardValue
136          string face =
137             cardValue.Substring( 0, cardValue.IndexOf( ' ' ) );
138
139          // retrieve the suit of the card from cardValue
140          string suit =
141             cardValue.Substring( cardValue.IndexOf( ' ' ) + 1 );
142
143          char suitLetter; // suit letter used to form image file name
144
145          // determine the suit letter of the card
146          switch ( Convert.ToInt32( suit ) )
147          {
148             case 0: // clubs
149                suitLetter = 'c';
150                break;
151             case 1: // diamonds
152                suitLetter = 'd';
153                break;
154             case 2: // hearts
155                suitLetter = 'h';
156                break;
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 7 of 18.)

```
157             default: // spades
158                suitLetter = 's';
159                break;
160          } // end switch
161
162          // set displayBox to display appropriate image
163          displayBox.Image = ( Image ) pictureLibrary.GetObject(
164             "_" + face + suitLetter );
165       } // end method DisplayCard
166
167       // displays all player cards and shows
168       // appropriate game status message
169       public void GameOver( GameStatus winner )
170       {
171          string[] cards = dealersCards.Split( '\t' );
172
173          // display all the dealer's cards
174          for ( int i = 0; i < cards.Length; i++ )
175             DisplayCard( i, cards[ i ] );
176
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 8 of 18.)

```
177          // display appropriate status image
178          if ( winner == GameStatus.PUSH ) // push
179             statusPictureBox.Image =
180                ( Image ) pictureLibrary.GetObject( "tie" );
181          else if ( winner == GameStatus.LOSE ) // player loses
182             statusPictureBox.Image =
183                ( Image ) pictureLibrary.GetObject( "lose" );
184          else if ( winner == GameStatus.BLACKJACK )
185             // player has blackjack
186             statusPictureBox.Image =
187                ( Image ) pictureLibrary.GetObject( "blackjack" );
188          else // player wins
189             statusPictureBox.Image =
190                ( Image ) pictureLibrary.GetObject( "win" );
191
192          // display final totals for dealer and player
193          dealerTotalLabel.Text =
194             "Dealer: " + dealer.GetHandValue( dealersCards );
195          playerTotalLabel.Text =
196             "Player: " + dealer.GetHandValue( playersCards );
197
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 9 of 18.)

```
198          // reset controls for new game
199          stayButton.Enabled = false;
200          hitButton.Enabled = false;
201          dealButton.Enabled = true;
202       } // end method GameOver
203
204       // deal two cards each to dealer and player
205       private void dealButton_Click( object sender, EventArgs e )
206       {
207          string card; // stores a card temporarily until added to a hand
208
209          // clear card images
210          foreach ( PictureBox cardImage in cardBoxes )
211             cardImage.Image = null;
212
213          statusPictureBox.Image = null; // clear status image
214          dealerTotalLabel.Text = string.Empty; // clear dealer total
215          playerTotalLabel.Text = string.Empty; // clear player total
216
217          // create a new, shuffled deck on the web service host
218          dealer.Shuffle();
219
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 10 of 18.)

```
220         // deal two cards to player
221         playersCards = dealer.DealCard(); // deal first card to player
222         DisplayCard( 11, playersCards ); // display card
223         card = dealer.DealCard(); // deal second card to player
224         DisplayCard( 12, card ); // update GUI to display new card
225         playersCards += '\t' + card; // add second card to player's hand
226
227         // deal two cards to dealer, only display face of first card
228         dealersCards = dealer.DealCard(); // deal first card to dealer
229         DisplayCard( 0, dealersCards ); // display card
230         card = dealer.DealCard(); // deal second card to dealer
231         DisplayCard( 1, string.Empty ); // display card face down
232         dealersCards += '\t' + card; // add second card to dealer's hand
233
234         stayButton.Enabled = true; // allow player to stay
235         hitButton.Enabled = true; // allow player to hit
236         dealButton.Enabled = false; // disable Deal Button
237
238         // determine the value of the two hands
239         int dealersTotal = dealer.GetHandValue( dealersCards );
240         int playersTotal = dealer.GetHandValue( playersCards );
241
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 11 of 18.)

```
242            // if hands equal 21, it is a push
243            if ( dealersTotal == playersTotal && dealersTotal == 21 )
244               GameOver( GameStatus.PUSH );
245            else if ( dealersTotal == 21 ) // if dealer has 21, dealer wins
246               GameOver( GameStatus.LOSE );
247            else if ( playersTotal == 21 ) // player has blackjack
248               GameOver( GameStatus.BLACKJACK );
249
250            // next dealer card has index 2 in cardBoxes
251            currentDealerCard = 2;
252
253            // next player card has index 13 in cardBoxes
254            currentPlayerCard = 13;
255         } // end method dealButton
256
257         // deal another card to player
258         private void hitButton_Click( object sender, EventArgs e )
259         {
260            string card = dealer.DealCard(); // deal new card
261            playersCards += '\t' + card; // add new card to player's hand
262
263            DisplayCard( currentPlayerCard, card ); // display card
264            ++currentPlayerCard;
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 12 of 18.)

```
265
266             // determine the value of the player's hand
267             int total = dealer.GetHandValue( playersCards );
268
269             // if player exceeds 21, house wins
270             if ( total > 21 )
271                GameOver( GameStatus.LOSE );
272             else if ( total == 21 ) // if player has 21, dealer's turn
273             {
274                hitButton.Enabled = false;
275                DealerPlay();
276             } // end if
277          } // end method hitButton_Click
278
279          // play the dealer's hand after the player chooses to stay
280          private void stayButton_Click( object sender, EventArgs e )
281          {
282             stayButton.Enabled = false; // disable Stay Button
283             hitButton.Enabled = false; // disable Hit Button
284             dealButton.Enabled = true; // enable Deal Button
285             DealerPlay(); // player chose to stay, so play the dealer's hand
286          } // end method stayButton_Click
287       } // end class Blackjack
288    } // end namespace BlackjackClient
```

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 13 of 18.)

a) Initial cards dealt to the player and the dealer when the user presses the **Deal** button.

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 14 of 18.)

b) Cards after the player presses the **Hit** button once, then the **Stay** button. In this case, the player wins the game with a higher total than the dealer.

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 15 of 18.)

c) Cards after the player presses the **Hit** button once, then the **Stay** button. In this case, the player busts (exceeds 21) and the dealer wins the game.

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 16 of 18.)

d) Cards after the player presses the **Deal** button. In this case, the player wins with Blackjack because the first two cards are an ace and a card with a value of 10 (a jack in this case).

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 17 of 18.)

e) Cards after the player presses the **Stay** button. In this case, the player and dealer push—they have the same card total.

Fig. 28.20 | Blackjack game that uses the `BlackjackService` web service. (Part 18 of 18.)

```
1  // Fig. 28.21: IReservationService.cs
2  // Airline reservation WCF web service interface.
3  using System.ServiceModel;
4
5  [ServiceContract]
6  public interface IReservationService
7  {
8     // reserves a seat
9     [OperationContract]
10    bool Reserve( string seatType, string classType );
11 } // end interface IReservationService
```

Fig. 28.21 | Airline reservation WCF web-service interface.

```
1  // Fig. 28.22: ReservationService.cs
2  // Airline reservation WCF web service.
3  using System.Linq;
4
5  public class ReservationService : IReservationService
6  {
7     // create ticketsDB object to access Tickets database
8     private TicketsDataContext ticketsDB = new TicketsDataContext();
9
10    // checks database to determine whether matching seat is available
11    public bool Reserve( string seatType, string classType )
12    {
13       //  LINQ query to find seats matching the parameters
14       var result =
15          from seat in ticketsDB.Seats
16          where ( seat.Taken == false ) && ( seat.Type == seatType ) &&
17             ( seat.Class == classType )
18          select seat;
19
20       // get first available seat
21       Seat firstAvailableSeat = result.FirstOrDefault();
22
```

Fig. 28.22 | Airline reservation WCF web service. (Part 1 of 2.)

```
23          // if seat is available seats, mark it as taken
24          if ( firstAvailableSeat != null )
25          {
26             firstAvailableSeat.Taken = true; // mark the seat as taken
27             ticketsDB.SubmitChanges(); // update
28             return true; // seat was reserved
29          } // end if
30
31          return false; // no seat was reserved
32       } // end method Reserve
33    } // end class ReservationService
```

Fig. 28.22 | Airline reservation WCF web service. (Part 2 of 2.)

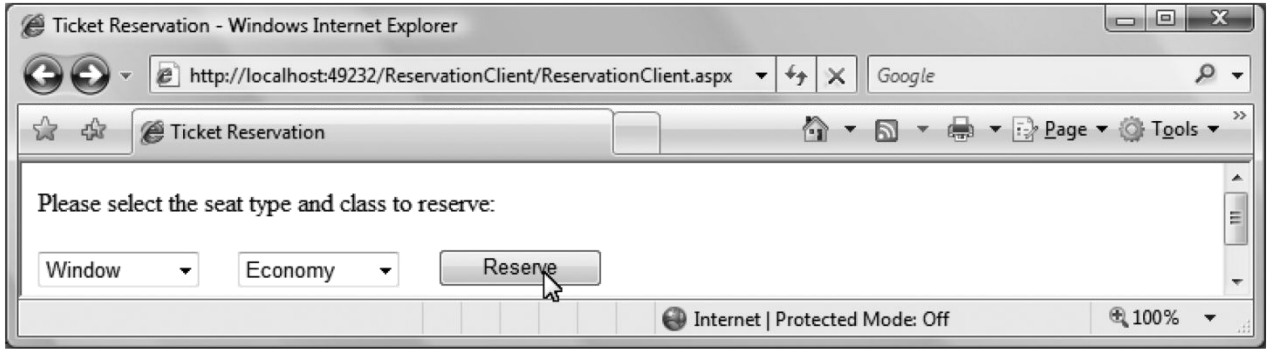


Fig. 28.23 | ASPX file that takes reservation information.

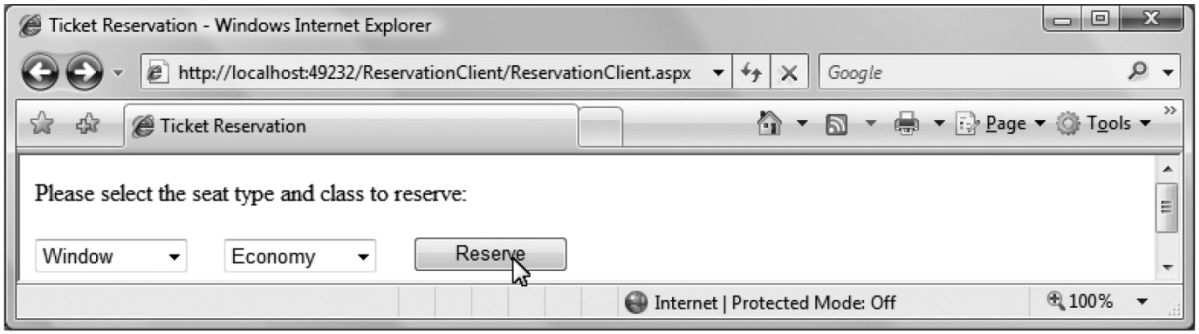
```
1  // Fig. 28.24: ReservationClient.aspx.cs
2  // ReservationClient code behind file.
3  using System;
4
5  public partial class ReservationClient : System.Web.UI.Page
6  {
7     // object of proxy type used to connect to ReservationService
8     private ServiceReference.ReservationServiceClient ticketAgent =
9        new ServiceReference.ReservationServiceClient();
10
11    // attempt to reserve the selected type of seat
12    protected void reserveButton_Click( object sender, EventArgs e )
13    {
14       // if the ticket is reserved
15       if ( ticketAgent.Reserve( seatList.SelectedItem.Text,
16          classList.SelectedItem.Text ) )
17       {
18          // hide other controls
19          instructionsLabel.Visible = false;
20          seatList.Visible = false;
21          classList.Visible = false;
22          reserveButton.Visible = false;
23          errorLabel.Visible = false;
24
```

Fig. 28.24 | ReservationClient code-behind file. (Part 1 of 2.)

```
25            // display message indicating success
26            Response.Write( "Your reservation has been made. Thank you." );
27         } // end if
28         else // service method returned false, so signal failure
29         {
30            // display message in the initially blank errorLabel
31            errorLabel.Text = "This type of seat is not available. " +
32               "Please modify your request and try again.";
33         } // end else
34      } // end method reserveButton_Click
35   } // end class ReservationClient
```

Fig. 28.24 | ReservationClient code-behind file. (Part 2 of 2.)

a) Selecting a seat



b) Seat is reserved successfully

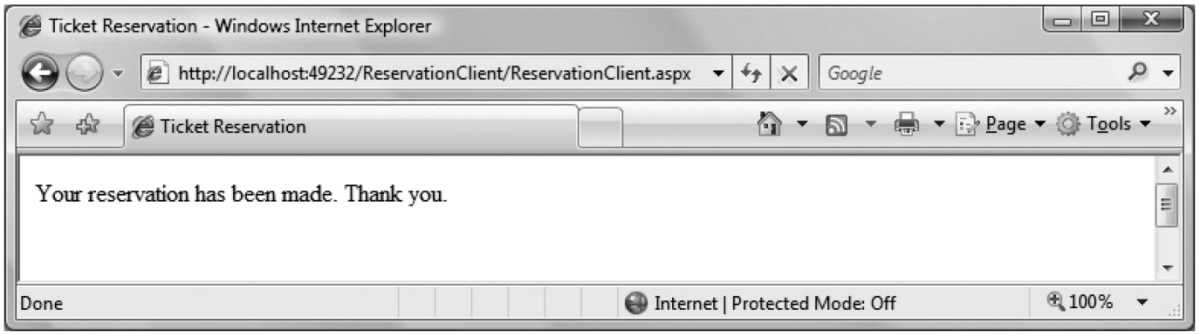


Fig. 28.25 | Ticket reservation web-application sample execution. (Part 1 of 2.)

c) Attempting to reserve another seat

d) No seats match the requested type and class

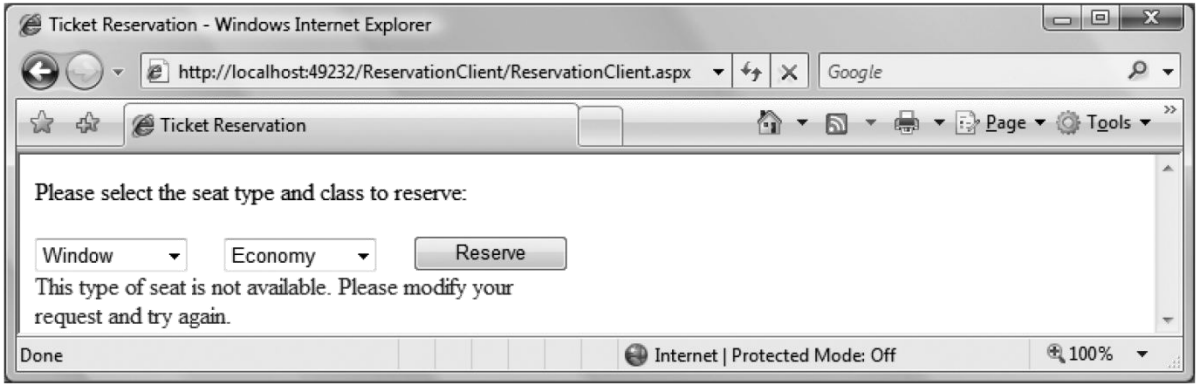


Fig. 28.25 | Ticket reservation web-application sample execution. (Part 2 of 2.)

```
1  // Fig. 28.26: Equation.cs
2  // Class Equation that contains information about an equation.
3  using System.Runtime.Serialization;
4
5  [DataContract]
6  public class Equation
7  {
8     // automatic property to access the left operand
9     [DataMember]
10    private int Left { get; set; }
11
12    // automatic property to access the right operand
13    [DataMember]
14    private int Right { get; set; }
15
16    // automatic property to access the result of applying
17    // an operation to the left and right operands
18    [DataMember]
19    private int Result { get; set; }
20
21    // automatic property to access the operation
22    [DataMember]
23    private string Operation { get; set; }
```

Fig. 28.26 | Class `Equation` that contains information about an equation. (Part 1 of 4.)

```
24
25      // required default constructor
26      public Equation()
27         : this( 0, 0, "add" )
28      {
29         // empty body
30      } // end default constructor
31
32      // three-argument constructor for class Equation
33      public Equation( int leftValue, int rightValue, string type )
34      {
35         Left = leftValue;
36         Right = rightValue;
37
38         switch ( type ) // perform appropriate operation
39         {
40            case "add": // addition
41               Result = Left + Right;
42               Operation = "+";
43               break;
44            case "subtract": // subtraction
45               Result = Left - Right;
46               Operation = "-";
47               break;
```

Fig. 28.26 | Class `Equation` that contains information about an equation. (Part 2 of 4.)

```
48            case "multiply": // multiplication
49               Result = Left * Right;
50               Operation = "*";
51               break;
52         } // end switch
53      } // end three-argument constructor
54
55      // return string representation of the Equation object
56      public override string ToString()
57      {
58         return string.Format( "{0} {1} {2} = {4}", Left, Operation,
59            Right, Result );
60      } // end method ToString
61
62      // property that returns a string representing left-hand side
63      [DataMember]
64      private string LeftHandSide
65      {
66         get
67         {
68            return string.Format( "{0} {1} {2}", Left, Operation, Right );
69         } // end get
```

Fig. 28.26 | Class `Equation` that contains information about an equation. (Part 3 of 4.)

```
70          set
71          {
72             // empty body
73          } // end set
74       } // end property LeftHandSide
75
76       // property that returns a string representing right-hand side
77       [DataMember]
78       private string RightHandSide
79       {
80          get
81          {
82             return Result.ToString();
83          } // end get
84          set
85          {
86             // empty body
87          } // end set
88       } // end property RightHandSide
89    } // end class Equation
```

Fig. 28.26 | Class `Equation` that contains information about an equation. (Part 4 of 4.)

```
1  // Fig. 28.27: IEquationGeneratorService.cs
2  // WCF REST service interface to create random equations based on a
3  // specified operation and difficulty level.
4  using System.ServiceModel;
5  using System.ServiceModel.Web;
6
7  [ServiceContract]
8  public interface IEquationGeneratorService
9  {
10    // method to generate a math equation
11    [OperationContract]
12    [WebGet( UriTemplate = "equation/{operation}/{level}" )]
13    Equation GenerateEquation( string operation, string level );
14 } // end interface IEquationGeneratorService
```

Fig. 28.27 | WCF REST service interface to create random equations based on a specified operation and difficulty level.

```
1  // Fig. 28.28: EquationGeneratorService.cs
2  // WCF REST service to create random equations based on a
3  // specified operation and difficulty level.
4  using System;
5
6  public class EquationGeneratorService : IEquationGeneratorService
7  {
8     // method to generate a math equation
9     public Equation GenerateEquation( string operation, string level )
10    {
11       // calculate maximum and minimum number to be used
12       int maximum =
13          Convert.ToInt32( Math.Pow( 10, Convert.ToInt32( level ) ) );
14       int minimum =
15          Convert.ToInt32( Math.Pow( 10, Convert.ToInt32( level ) - 1 ) );
16
17       Random randomObject = new Random(); // generate random numbers
18
```

Fig. 28.28 | WCF REST service to create random equations based on a specified operation and difficulty level. (Part 1 of 2.)

```
19          // create Equation consisting of two random
20          // numbers in the range minimum to maximum
21          Equation newEquation = new Equation(
22             randomObject.Next( minimum, maximum ),
23             randomObject.Next( minimum, maximum ), operation );
24
25          return newEquation;
26       } // end method GenerateEquation
27    } // end class EquationGeneratorService
```

Fig. 28.28 | WCF REST service to create random equations based on a specified operation and difficulty level. (Part 2 of 2.)

```
1  // Fig. 28.29: MathTutor.cs
2  // Math tutor using EquationGeneratorServiceXML to create equations.
3  using System;
4  using System.Net;
5  using System.Windows.Forms;
6  using System.Xml.Linq;
7
8  namespace MathTutorXML
9  {
10    public partial class MathTutor : Form
11    {
12       private string operation = "add"; // the default operation
13       private int level = 1; // the default difficulty level
14       private string leftHandSide; // the left side of the equation
15       private int result; // the answer
16       private XNamespace xmlNamespace =
17          XNamespace.Get( "http://schemas.datacontract.org/2004/07/" );
18
19       // object used to invoke service
20       private WebClient service = new WebClient();
21
```

Fig. 28.29 | Math tutor using EquationGeneratorServiceXML to create equations. (Part 1 of 9.)

```
22          public MathTutor()
23          {
24             InitializeComponent();
25
26             // add DownloadStringCompleted event handler to WebClient
27             service.DownloadStringCompleted +=
28                new DownloadStringCompletedEventHandler(
29                service_DownloadStringCompleted );
30          } // end constructor
31
32          // generates new equation when user clicks button
33          private void generateButton_Click( object sender, EventArgs e )
34          {
35             // send request to EquationGeneratorServiceXML
36             service.DownloadStringAsync( new Uri(
37                "http://localhost:49732/EquationGeneratorServiceXML" +
38                "/Service.svc/equation/" + operation + "/" + level ) );
39          } // end method generateButton_Click
40
```

Fig. 28.29 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 2 of 9.)

```
41         // process web service response
42         private void service_DownloadStringCompleted(
43            object sender, DownloadStringCompletedEventArgs e )
44         {
45            // check if any errors occurred in retrieving service data
46            if ( e.Error == null )
47            {
48               // parse response and get LeftHandSide and Result values
49               XDocument xmlResponse = XDocument.Parse( e.Result );
50               leftHandSide = xmlResponse.Element(
51                  xmlNamespace + "Equation" ).Element(
52                  xmlNamespace + "LeftHandSide" ).Value;
53               result = Convert.ToInt32( xmlResponse.Element(
54                  xmlNamespace + "Equation" ).Element(
55                  xmlNamespace + "Result" ).Value );
56
57               // display left side of equation
58               questionLabel.Text = leftHandSide;
59               okButton.Enabled = true; // enable okButton
60               answerTextBox.Enabled = true; // enable answerTextBox
61            } // end if
62         } // end method client_DownloadStringCompleted
63
```

Fig. 28.29 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 3 of 9.)

```
64         // check user's answer
65         private void okButton_Click( object sender, EventArgs e )
66         {
67            if ( !string.IsNullOrEmpty( answerTextBox.Text ) )
68            {
69               // get user's answer
70               int userAnswer = Convert.ToInt32( answerTextBox.Text );
71
72               // determine whether user's answer is correct
73               if ( result == userAnswer )
74               {
75                  questionLabel.Text = string.Empty; // clear question
76                  answerTextBox.Clear(); // clear answer
77                  okButton.Enabled = false; // disable OK button
78                  MessageBox.Show( "Correct! Good job!", "Result" );
79               } // end if
80               else
81               {
82                  MessageBox.Show( "Incorrect. Try again.", "Result" );
83               } // end else
84            } // end if
85         } // end method okButton_Click
86
```

Fig. 28.29 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 4 of 9.)

```
87          // set the operation to addition
88          private void additionRadioButton_CheckedChanged( object sender,
89             EventArgs e )
90          {
91             if ( additionRadioButton.Checked )
92                operation = "add";
93          } // end method additionRadioButton_CheckedChanged
94
95          // set the operation to subtraction
96          private void subtractionRadioButton_CheckedChanged( object sender,
97             EventArgs e )
98          {
99             if ( subtractionRadioButton.Checked )
100               operation = "subtract";
101         } // end method subtractionRadioButton_CheckedChanged
102
103         // set the operation to multiplication
104         private void multiplicationRadioButton_CheckedChanged(
105            object sender, EventArgs e )
106         {
107            if ( multiplicationRadioButton.Checked )
108               operation = "multiply";
109         } // end method multiplicationRadioButton_CheckedChanged
```

Fig. 28.29 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 5 of 9.)

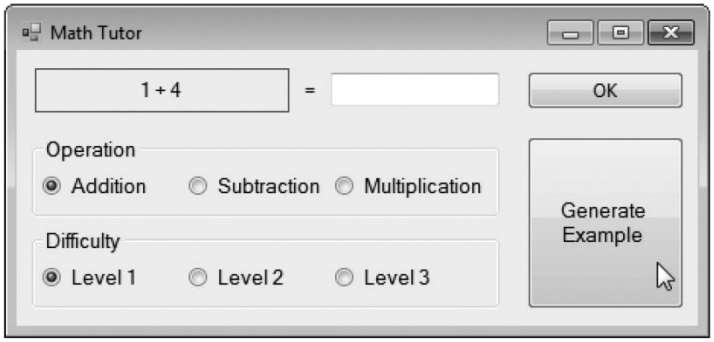
```
110
111         // set difficulty level to 1
112         private void levelOneRadioButton_CheckedChanged( object sender,
113            EventArgs e )
114         {
115            if ( levelOneRadioButton.Checked )
116               level = 1;
117         } // end method levelOneRadioButton_CheckedChanged
118
119         // set difficulty level to 2
120         private void levelTwoRadioButton_CheckedChanged( object sender,
121            EventArgs e )
122         {
123            if ( levelTwoRadioButton.Checked )
124               level = 2;
125         } // end method levelTwoRadioButton_CheckedChanged
126
```

Fig. 28.29 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 6 of 9.)

```
127         // set difficulty level to 3
128         private void levelThreeRadioButton_CheckedChanged( object sender,
129            EventArgs e )
130         {
131            if ( levelThreeRadioButton.Checked )
132               level = 3;
133         } // end method levelThreeRadioButton_CheckedChanged
134      } // end class MathTutor
135   } // end namespace MathTutorXML
```

Fig. 28.29 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 7 of 9.)

a) Generating a level 1 addition equation



b) Answering the question incorrectly

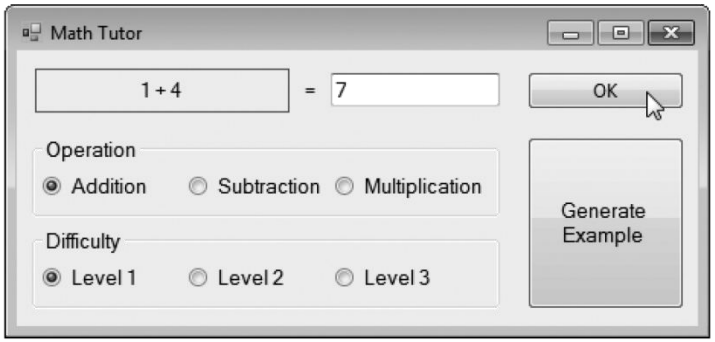


Fig. 28.29 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 8 of 9.)

c) Answering the question correctly

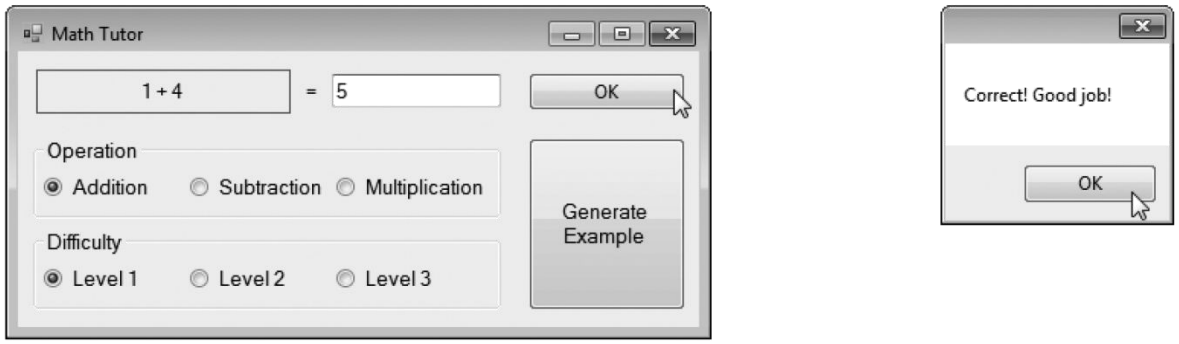


Fig. 28.29 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 9 of 9.)

```
1  // Fig. 28.30: IEquationGeneratorService.cs
2  // WCF REST service interface to create random equations based on a
3  // specified operation and difficulty level.
4  using System.ServiceModel;
5  using System.ServiceModel.Web;
6
7  [ServiceContract]
8  public interface IEquationGeneratorService
9  {
10    // method to generate a math equation
11    [OperationContract]
12    [WebGet( ResponseFormat = WebMessageFormat.Json,
13       UriTemplate = "equation/{operation}/{level}" )]
14    Equation GenerateEquation( string operation, string level );
15 } // end interface IEquationGeneratorService
```

Fig. 28.30 | WCF REST service interface to create random equations based on a specified operation and difficulty level.

```
1  // Fig. 28.31: MathTutorForm.cs
2  // Math tutor using EquationGeneratorServiceJSON to create equations.
3  using System;
4  using System.IO;
5  using System.Net;
6  using System.Runtime.Serialization.Json;
7  using System.Text;
8  using System.Windows.Forms;
9
10 namespace MathTutorJSON
11 {
12    public partial class MathTutorForm : Form
13    {
14       private string operation = "add"; // the default operation
15       private int level = 1; // the default difficulty level
16       private Equation currentEquation;  // represents the Equation
17
18       // object used to invoke service
19       private WebClient service = new WebClient();
20
```

Fig. 28.31 | Math tutor using EquationGeneratorServiceJSON. (Part 1 of 9.)

```
21          public MathTutorForm()
22          {
23             InitializeComponent();
24
25             // add DownloadStringCompleted event handler to WebClient
26             service.DownloadStringCompleted +=
27                new DownloadStringCompletedEventHandler(
28                   service_DownloadStringCompleted );
29          } // end constructor
30
31          // generates new equation when user clicks button
32          private void generateButton_Click( object sender, EventArgs e )
33          {
34             // send request to EquationGeneratorServiceJSON
35             service.DownloadStringAsync( new Uri(
36                "http://localhost:50238/EquationGeneratorServiceJSON" +
37                "/Service.svc/equation/" + operation + "/" + level ) );
38          } // end method generateButton_Click
39
```

Fig. 28.31 | Math tutor using `EquationGeneratorServiceJSON`. (Part 2 of 9.)

```
40         // process web service response
41         private void service_DownloadStringCompleted(
42            object sender, DownloadStringCompletedEventArgs e )
43         {
44            // check if any errors occurred in retrieving service data
45            if ( e.Error == null )
46            {
47               // deserialize response into an Equation object
48               DataContractJsonSerializer JSONSerializer =
49                  new DataContractJsonSerializer( typeof( Equation ) );
50               currentEquation =
51                  ( Equation ) JSONSerializer.ReadObject( new
52                  MemoryStream( Encoding.Unicode.GetBytes( e.Result ) ) );
53
54               // display left side of equation
55               questionLabel.Text = currentEquation.LeftHandSide;
56               okButton.Enabled = true; // enable okButton
57               answerTextBox.Enabled = true; // enable answerTextBox
58            } // end if
59         } // end method client_DownloadStringCompleted
60
```

Fig. 28.31 | Math tutor using `EquationGeneratorServiceJSON`. (Part 3 of 9.)

```
61         // check user's answer
62         private void okButton_Click( object sender, EventArgs e )
63         {
64            if ( !string.IsNullOrEmpty( answerTextBox.Text ) )
65            {
66               // determine whether user's answer is correct
67               if ( currentEquation.Result ==
68                  Convert.ToInt32( answerTextBox.Text ) )
69               {
70                  questionLabel.Text = string.Empty; // clear question
71                  answerTextBox.Clear(); // clear answer
72                  okButton.Enabled = false; // disable OK button
73                  MessageBox.Show( "Correct! Good job!", "Result" );
74               } // end if
75               else
76               {
77                  MessageBox.Show( "Incorrect. Try again.", "Result" );
78               } // end else
79            } // end if
80         } // end method okButton_Click
81
```

Fig. 28.31 | Math tutor using `EquationGeneratorServiceJSON`. (Part 4 of 9.)

```
82          // set the operation to addition
83          private void additionRadioButton_CheckedChanged( object sender,
84             EventArgs e )
85          {
86             if ( additionRadioButton.Checked )
87                operation = "add";
88          } // end method additionRadioButton_CheckedChanged
89
90          // set the operation to subtraction
91          private void subtractionRadioButton_CheckedChanged( object sender,
92             EventArgs e )
93          {
94             if ( subtractionRadioButton.Checked )
95                operation = "subtract";
96          } // end method subtractionRadioButton_CheckedChanged
97
98          // set the operation to multiplication
99          private void multiplicationRadioButton_CheckedChanged(
100            object sender, EventArgs e )
101         {
102            if ( multiplicationRadioButton.Checked )
103               operation = "multiply";
104         } // end method multiplicationRadioButton_CheckedChanged
```

Fig. 28.31 | Math tutor using `EquationGeneratorServiceJSON`. (Part 5 of 9.)

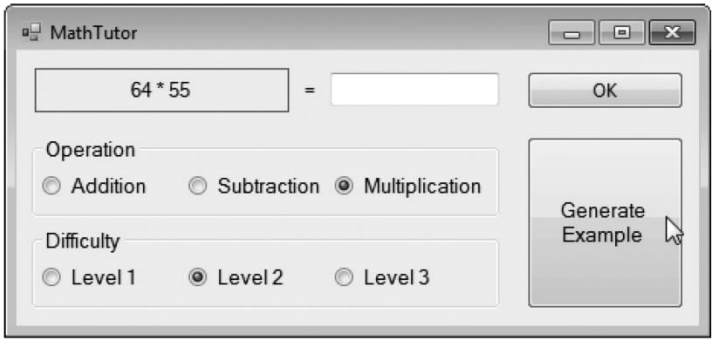
```
105
106         // set difficulty level to 1
107         private void levelOneRadioButton_CheckedChanged( object sender,
108            EventArgs e )
109         {
110            if ( levelOneRadioButton.Checked )
111               level = 1;
112         } // end method levelOneRadioButton_CheckedChanged
113
114         // set difficulty level to 2
115         private void levelTwoRadioButton_CheckedChanged( object sender,
116            EventArgs e )
117         {
118            if ( levelTwoRadioButton.Checked )
119               level = 2;
120         } // end method levelTwoRadioButton_CheckedChanged
121
```

Fig. 28.31 | Math tutor using `EquationGeneratorServiceJSON`. (Part 6 of 9.)

```
122         // set difficulty level to 3
123         private void levelThreeRadioButton_CheckedChanged( object sender,
124            EventArgs e )
125         {
126            if ( levelThreeRadioButton.Checked )
127               level = 3;
128         } // end method levelThreeRadioButton_CheckedChanged
129      } // end class MathTutorForm
130   } // end namespace MathTutorJSON
```

Fig. 28.31 | Math tutor using `EquationGeneratorServiceJSON`. (Part 7 of 9.)

a) Generating a level 2 multiplication equation



b) Answering the question incorrectly

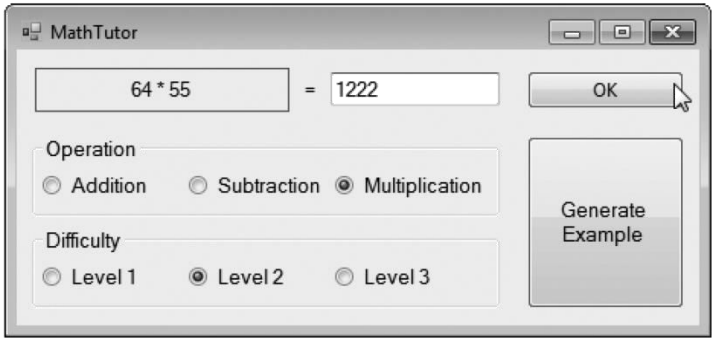


Fig. 28.31 | Math tutor using `EquationGeneratorServiceJSON`. (Part 8 of 9.)

c) Answering the question correctly

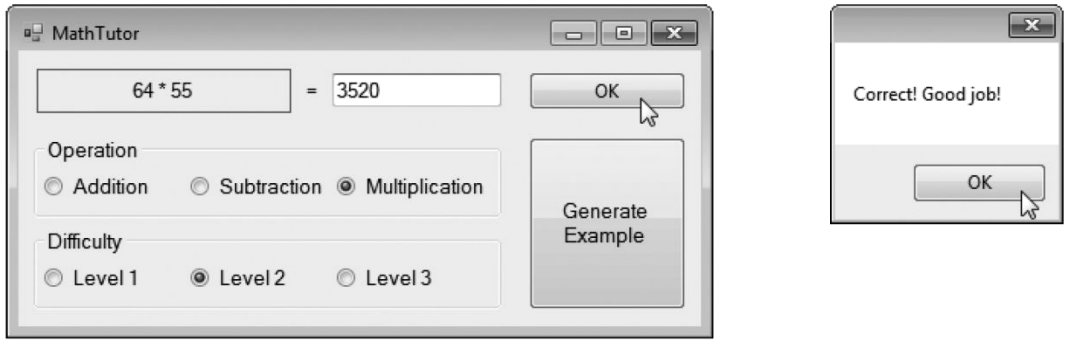


Fig. 28.31 | Math tutor using `EquationGeneratorServiceJSON`. (Part 9 of 9.)

```
1  // Fig. 28.32: Equation.cs
2  // Equation class representing a JSON object.
3  using System;
4
5  namespace MathTutorJSON
6  {
7     [Serializable]
8     class Equation
9     {
10       public int Left = 0;
11       public string LeftHandSide = null;
12       public string Operation = null;
13       public int Result = 0;
14       public int Right = 0;
15       public string RightHandSide = null;
16    } // end class Equation
17 } // end namespace MathTutorJSON
```

Fig. 28.32 | Equation class representing a JSON object.