

*Integrating EBay, Google, Amazon, FedEx and more*



# Real World Web Services

O'REILLY®

*Will Iverson*

---

## Project 3: Billing and Faxing

If you manage to successfully sell any quantity of items on the Internet, it's hard to imagine not setting up automated systems to help handle the workload. You see this sort of automation all of the time; for example, you expect an automated system to automatically send a confirmation email when an order has been placed. Upon occasion, however, you may not be fortunate enough to work with individuals that have access to the latest technologies, such as email and the Web (yes, for some people those are the latest technologies). For example, your manufacturing partner may not be up to speed. Even in those situations, you can still probably still rely on that old standby—the fax.

In this chapter, I'll show how to set up a system whereby you receive a payment notification from PayPal and respond by sending a fax containing the order information. PayPal is the well-known payment processing service, <http://www.paypal.com/> (see Figure 6-1). For faxes, we'll be using the web services provided by InterFAX, <http://www.interfax.net/> (shown in Figure 6-2).

PayPal's development offerings are all listed on the main page, but the specific technology we're interested in here is Instant Payment Notification (IPN), <http://www.paypal.com/cgi-bin/webscr?cmd=p/xcl/rec/ipn-intro-outside>. The idea is fairly straightforward: when a PayPal payment is received, the PayPal server sends a notification to a URL you specify. There is no specific developer sign up required to make use of the IPN functionality; you just use the standard PayPal account registration.

The fax services provided by InterFAX allow you to send faxes using any of a variety of technologies; in addition to SOAP, you can also use email. A free developer registration (<http://www.interfax.net/en/dev/index.html>) allows you to send test faxes to a single designated phone number for free (up to a \$10 credit, which can be refreshed for free on request).

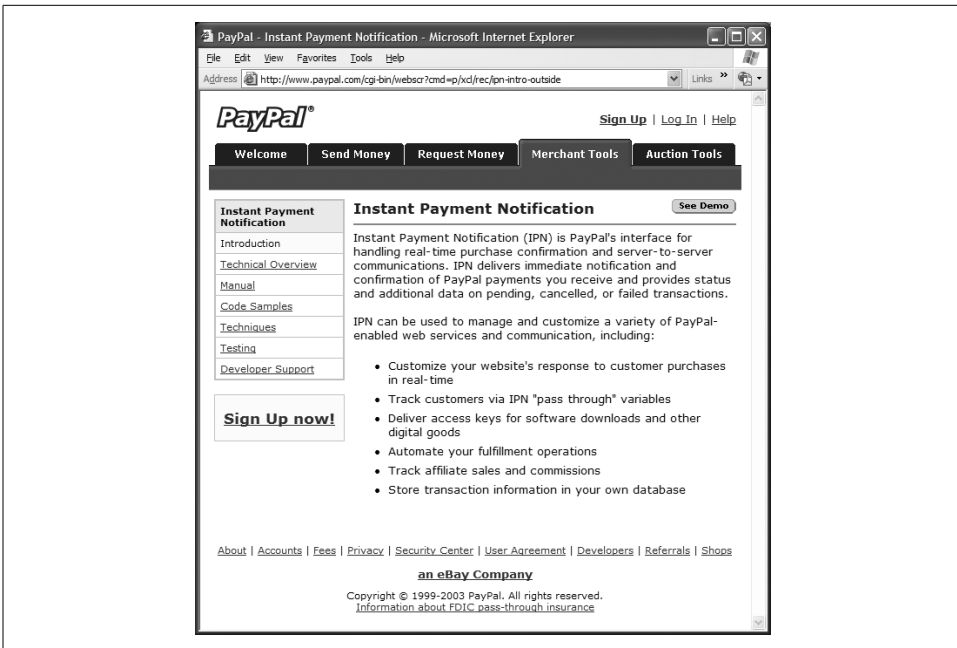


Figure 6-1. PayPal instant payment notification web site



Figure 6-2. InterFAX developer site

# Starting the Transaction

To kick our example off, let's start with a simple HTML page that will initiate our transaction, as shown in Figure 6-3.



Figure 6-3. Initial payment screen

It's not the world's most sophisticated marketing page, but it does the job and is easy to understand. The HTML for this simple page is shown in Example 6-1.

*Example 6-1. Payment initialization form*

```
<%@ page contentType="text/html; charset=iso-8859-1"
language="java" import="com.cascadetg.ch06.*" %>
<HTML>
<HEAD><title>ch06 : Simple Money Sender</title>
<link href="../../ch04/www/default.css"
rel="stylesheet" type="text/css">
</HEAD>
<BODY>
  <form action="https://www.paypal.com/cgi-bin/webscr" method="post">
    <p><strong>ch06: Simple Money Sender</strong></p>
    <p>
      <input type="hidden" name="cmd" value="_xclick">
      <input type="hidden" name="business"
value="test_account@cascadetg.com">
      <input type="hidden" name="item_name" value="IPN Test">
      <input type="hidden" name="amount" value="0.01">
      <input type="hidden" name="return"
value="http://67.123.6.118:8080/ch06/www/pay_form.jsp">
      <input type="hidden" name="notify_url"
value="http://67.123.6.118:8080/ch06/www/notification.jsp">
      <!-- Note: The notify_url field is not required if the IPN
URL was set in your account profile)-->
      <input type="submit" name="submit" value="Send $0.01!">
    </p>
  </form>
```

Example 6-1. Payment initialization form (continued)

```
</BODY>  
</HTML>
```

Notice that the form is posted to the PayPal server, but return and notify\_url form elements are used to point back to this server. The IP address shown, 67.123.6.118:8080, is the WAN IP address and port for my server; you'll need to establish an Internet accessible address and port for your own system. Clicking the button takes the user to the PayPal server, as shown in Figure 6-4.

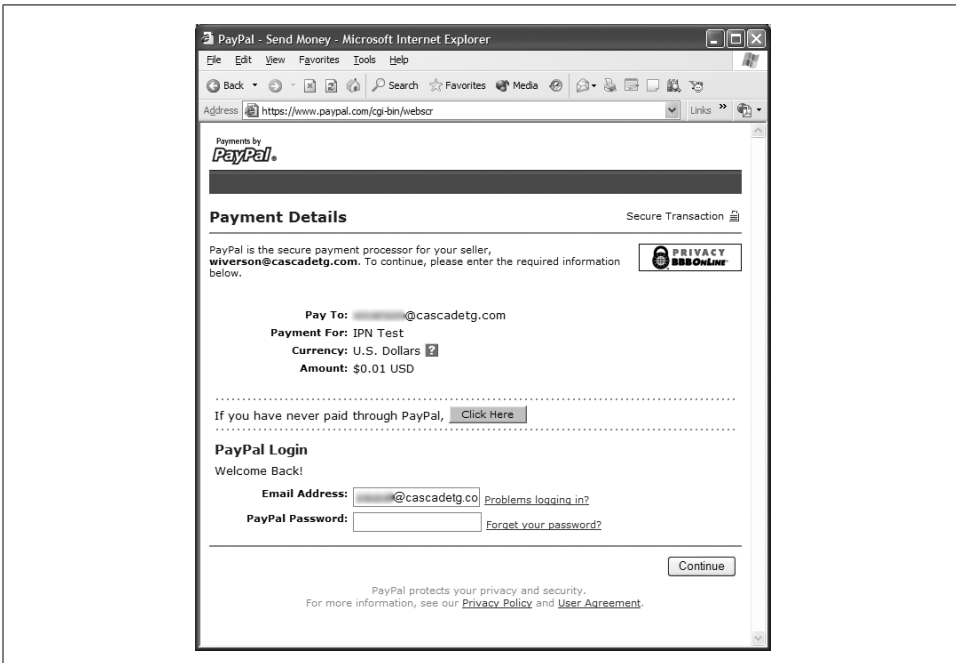


Figure 6-4. Initiating the PayPal transaction

Once at the PayPal server, users can walk through the rest of the transaction using the PayPal service, including entering credit card information and shipping address. When the order is complete, they are returned to the original *pay\_form.jsp* page, ready to send another \$0.01.



The example uses a transaction of \$0.01 for this; it isn't a transaction you'd ever use PayPal for because PayPal and credit card company fees will absorb the penny. You can use PayPal's management services to refund the quantities sent, which is especially important if you need to test sending larger monetary sums.

As an additional feature, the example in this chapter generates faxes when a refund is sent. If an order is cancelled, you'll probably want to let your (fictional) business partner know as soon as possible.

## Getting a Transaction Notification

When an order is processed, PayPal generates a notification that is then posted to the URL specified in the form as the hidden field `notify_url`. This is an example of a web service that relies on ordinary HTTP—no fancy SOAP or other RPC underpinnings.



Using a simple HTTP request/response is perhaps the most basic, universal real world web service. It works with virtually every programming language and requires no special configuration to use. It's a classic case of the simple solution being the best solution.

Example 6-2 shows the JSP used to receive notifications from the PayPal server.

### *Example 6-2. Notification JSP*

```
<%@ page contentType="text/html; charset=iso-8859-1"
language="java" import="com.cascadetg.ch06.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<%
new PayPalReciept().handleRequest(request, response);
response.setStatus(200);
%>
```

The line `response.setStatus(200)` notifies PayPal that the notification has successfully been handled. By default, PayPal resends the notification until it gets a status code of 200 back, indicating that the transmission was successful. Disabling this line or substituting a different response code is an easy way to generate additional test load for your server without having to manually enter a lot of transactions.

The bulk of the work for this application is handled by supporting Java classes, as shown in Figure 6-5.

The first class, shown in Example 6-3, exposes one main method, `handleRequest()`, to the JSP page. In the `handleRequest()` method, the first thing to be done is verify that the data sent by PayPal is correct with the `verifyRequest()` method. The `verifyRequest()` method retrieves the parameters sent by PayPal and then posts

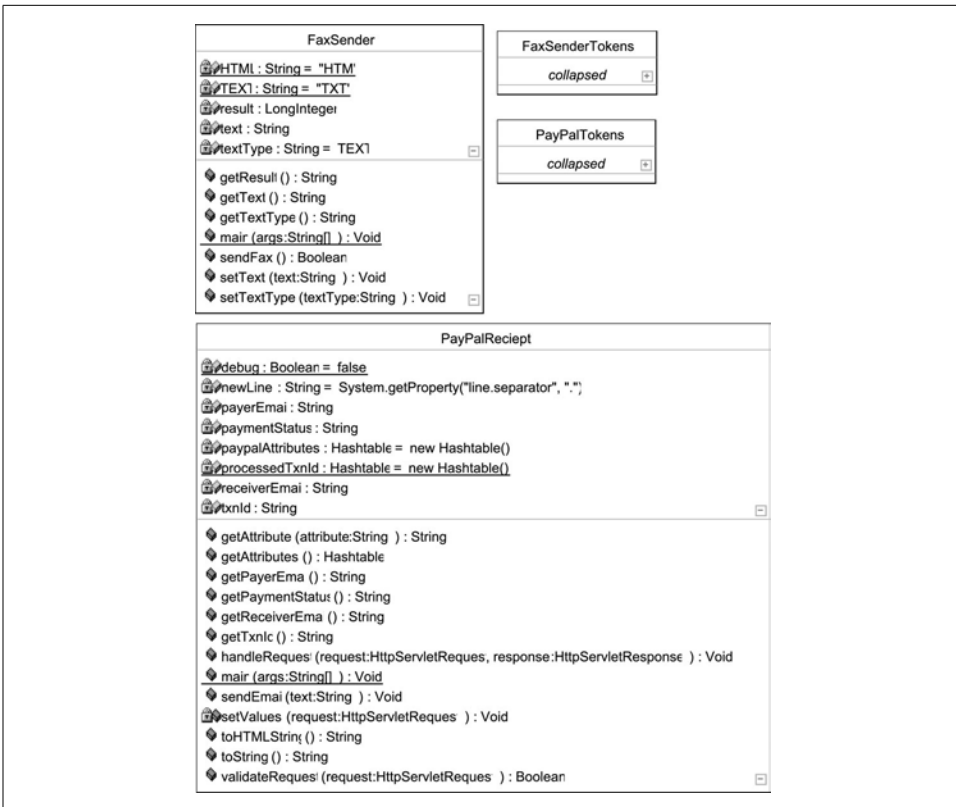


Figure 6-5. PayPal and Fax classes

them back to PayPal via an HTTPS connection, with an additional parameter added (`cmd=notify-validate`).

*Example 6-3. A PayPal receipt*

```
package com.cascadetg.ch06;
```

```
import java.util.*;
import java.net.*;
import java.io.*;
```

```
import javax.servlet.http.*;
import javax.mail.*;
import javax.mail.internet.*;
```

```
public class PayPalReciept
{
    // Used to store retrived values from the PayPal submission.
    String paymentStatus;
    String txnId;
    String receiverEmail;
    String payerEmail;
```

Example 6-3. A PayPal receipt (continued)

```
private static final boolean debug = false;

// System.getProperty line to get the proper new line character[s].
String newLine = System.getProperty("line.separator", ".");

// Keep track of sent transactions. Note that this is in-memory
// storage only - for a "real" system you would want to persist
// this information, as well as the rest of fields of this object.
// (mostly likely to a database of some sort).
private static Hashtable processedTxnId = new Hashtable();

// This method takes an incoming request and validates it both
// against the PayPal server and some internal logic.
public boolean validateRequest(HttpServletRequest request)
{
    try
    {
        // Read the post from PayPal system.
        Enumeration parameters = request.getParameterNames();
        // We then add a "cmd" attribute to send back to PayPal
        // to indicate that we want to validate the request.
        StringBuffer send =
            new StringBuffer("cmd=_notify-validate");

        // Here, we put all of the parameters passed in from the
        // PayPal notification POST.
        while (parameters.hasMoreElements())
        {
            String paramName = (String)parameters.nextElement();
            String paramValue = request.getParameter(paramName);
            send.append("&");
            send.append(paramName);
            send.append("=");
            send.append(URLEncoder.encode(paramValue));
        }

        if (debug)
            System.out.println(send.toString());

        // This next sequence opens a connection to the PayPal
        // server, sets up the connection, and writes the sent
        // parameters back to the PayPal server.
        URL paypalServer =
            new URL("https://www.paypal.com/cgi-bin/webscr");
        URLConnection paypalConnection =
            paypalServer.openConnection();
        paypalConnection.setDoOutput(true);
        paypalConnection.setRequestProperty(
            "Content-Type",
            "application/x-www-form-urlencoded");
        PrintWriter paypalServerWriter =
```

*Example 6-3. A PayPal receipt (continued)*

```
        new PrintWriterpaypalConnection.getOutputStream());
paypalServerWriter.println(send);
paypalServerWriter.close();

if (debug)
    System.out.println("Sent to PayPal server.");

// We then need to read the response from the PayPal
// server.
BufferedReader in =
    new BufferedReader(
        new InputStreamReader(
            paypalConnection.getInputStream()));
String paypalResponse = in.readLine();
in.close();

if (debug)
    System.out.println(
        "Read PayPal server response = " + paypalResponse);

// Set the values of this object from the values sent
// by the initial request. If these values are verified,
// we'll want them for later. If the values aren't
// verified, or something else is wrong, we'll want
// to track them for logging purposes.
setValues(request);

// If everything is ok, the response back should be
// VERIFIED. Otherwise, something went wrong.
if (paypalResponse.equals("VERIFIED"))
{
    // If it isn't completed, it's a status message of
    // some sort. We're only interested in Completed
    // payments.
    if (!paymentStatus.equals("Completed"))
        return false;

    // Make sure that we are the actual recipients of
    // the money.
    if (receiverEmail
        .compareToIgnoreCase(PayPalTokens.paypalEmail)
        != 0)
        return false;

    // Check the in-memory cache to verify that we
    // haven't already handled this transaction.
    if (processedTxnId.get(this.getTxnId()) != null)
        return false;

    // Everything looks good, so let's add this to the
    // transaction cache.
    processedTxnId.put(this.getTxnId(), this.getTxnId());
}
```

Example 6-3. A PayPal receipt (continued)

```
        return true;
    } else
    {
        System.out.println("Invalid PayPal transaction!");
        System.out.println(this.toString());
        return false;
    }
} catch (Exception e)
{
    System.out.println("Unable to connect to PayPal server.");
    e.printStackTrace();
    return false;
}
}

// "Flatten" the object to a String.
public String toString()
{
    StringBuffer output = new StringBuffer();
    Enumeration outEnum = this.getAttributes().keys();

    while (outEnum.hasMoreElements())
    {
        String outputStr = (String)outEnum.nextElement();
        output.append(outputStr);
        output.append(" : ");
        output.append(paypalAttributes.get(outputStr).toString());
        output.append(newLine);
    }

    return output.toString();
}

public String toHTMLString()
{
    StringBuffer htmlString = new StringBuffer();
    htmlString.append("<HTML><BODY>");
    htmlString.append("<TABLE HEIGHT='100%' WIDTH='100%'>");
    htmlString.append("<TR><TD>");

    Enumeration myValues = this.getAttributes().keys();
    while (myValues.hasMoreElements())
    {
        String next = (String)myValues.nextElement();
        htmlString.append(next);
        htmlString.append(" : ");
        htmlString.append(this.getAttribute(next).toString());
        htmlString.append("<BR>");
        htmlString.append(newLine);
    }
}
```

Example 6-3. A PayPal receipt (continued)

```
        htmlString.append("</TD></TR></TABLE></BODY></HTML>");
        return htmlString.toString();
    }

    // PayPal can send a variety of attributes back as part of a
    // transaction. We're interested in all of them, so we'll note
    // them all.
    Hashtable paypalAttributes = new Hashtable();

    public String getAttribute(String attribute)
    {
        return (String)paypalAttributes.get((String)attribute);
    }

    public Hashtable getAttributes()
    {
        return paypalAttributes;
    }

    /**
     * Reads the incoming values and fills out the object. Notice that
     * we are also recording the incoming IP address - if someone is
     * sending fake requests, the IP address can be an important bit of
     * information.
     *
     * @param request
     */
    private void setValues(HttpServletRequest request)
    {
        paypalAttributes = new Hashtable(request.getParameterMap());
        Enumeration attributes = request.getParameterNames();
        while (attributes.hasMoreElements())
        {
            String temp = (String)attributes.nextElement();
            paypalAttributes.put(temp, request.getParameter(temp));
        }

        paypalAttributes.put(
            "incoming_ip",
            request.getRemoteAddr().toString());

        paymentStatus = request.getParameter("payment_status");
        txnId = request.getParameter("txn_id");
        receiverEmail = request.getParameter("receiver_email");
        payerEmail = request.getParameter("payer_email");
    }

    /**
     * The main entry point from the JSP page request. We'll look at
     * the request and validate it, and depending on the results, we'll
     * either send a notification email OR a fax and a notification
```

*Example 6-3. A PayPal receipt (continued)*

```
* email.
*/
public void handleRequest(
    HttpServletRequest request,
    HttpServletResponse response)
{
    if (validateRequest(request))
    {
        if (debug)
            System.out.print("Sending fax... " + this.toString());
        FaxSender myFaxSender = new FaxSender();
        myFaxSender.setText(this.toHTMLString());
        myFaxSender.setTextType(FaxSender.HTML);
        if (debug)
            System.out.print("fax prepped...");
        myFaxSender.sendFax();
        this.paypalAttributes.put(
            "fax_id",
            myFaxSender.getResult().toString());

        if (debug)
            System.out.println("Fax sent.");

    }

    sendEmail(this.toString());
}

// These are the usual retrieval methods a'la the JavaBean
// patterns. Notice that there are only retrieval methods - no
// setters are provided.

public String getPayerEmail()
{
    return payerEmail;
}

public String getPaymentStatus()
{
    return paymentStatus;
}

public String getReceiverEmail()
{
    return receiverEmail;
}

/** Returns the transaction ID (aka TxnId) */
public String getTxnId()
{
    return txnId;
}
```

Example 6-3. A PayPal receipt (continued)

```
    }

    public void sendEmail(String text)
    {
        try
        {
            java.util.Properties myProperties = new Properties();
            myProperties.put("mail.smtp.host", PayPalTokens.mailhost);
            myProperties.put("mail.smtp.auth", "true");

            Session mySession = Session.getInstance(myProperties);
            //mySession.setDebug(true);

            Transport myTransport = mySession.getTransport("smtp");
            myTransport.connect(
                PayPalTokens.mailhost,
                PayPalTokens.mailhost_username,
                PayPalTokens.mailhost_password);

            Message myMessage =
                new javax.mail.internet.MimeMessage(mySession);
            myMessage.setFrom(
                new InternetAddress(
                    PayPalTokens.paypalEmail,
                    "PayPal Listener"));
            myMessage.addRecipient(
                Message.RecipientType.TO,
                new InternetAddress(PayPalTokens.paypalEmail));
            myMessage.setSubject("PayPal Notification");
            myMessage.setContent(this.toHTMLString(), "text/html");
            Address[] temp =
                { new InternetAddress(PayPalTokens.paypalEmail)};
            myMessage.setReplyTo(temp);
            myMessage.saveChanges();

            myTransport.sendMessage(
                myMessage,
                myMessage.getAllRecipients());

            myTransport.close();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    public static void main(String[] args)
    {
        (new PayPalReceipt()).sendEmail("test");
    }
}
```

If PayPal doesn't respond with a VERIFIED token, it's possible that someone is attempting to forge a transaction. If you see this occur in a real world environment, you should take immediate, aggressive steps to deal with this; it may be an attempt by a hacker to steal funds or even your identity.

## Responding to the Transaction

Assuming the PayPal data is validated, a fax and an email are sent. If the data isn't validated, just an email is sent (helpful for immediate notification of a potential hacker). The email processing is handled using the standard JavaMail API (<http://java.sun.com/products/javamail/>).



To recap, an unencrypted HTML form from a local server starts the transaction. This form specified an unencrypted notification URL for the receipt of the data (although an HTTPS connection can be used instead). Therefore, the HTTPS connection back to PayPal to validate this is pretty important; otherwise, the application might be getting a fake order that just happens to look like a PayPal request.

An example email, formatted as HTML is shown in Figure 6-6.

```
From: "PayPal Listener" <[redacted]@cascadetg.com>
To: [redacted]@cascadetg.com
Subject: PayPal Notification

address_owner : 1
notify_version : 1.5
address_zip : [redacted]
payment_fee : 0.01
mc_fee : 0.01
address_state : CA
receiver_id : [redacted]
fax_id : [redacted]
quantity : 1
address_status : confirmed
payer_email : [redacted]
txn_id : [redacted]
verify_sign : [redacted]
address_street : [redacted]
incoming_ip : 64.4.241.140
payment_type : instant
mc_currency : USD
last_name : [redacted]
payment_date : 20:29:50 Jan 04, 2004 PST
txn_type : web_accept
paypal_address_id : [redacted]
address_name : [redacted]
address_country : United States
ebay_address_id : [redacted]
custom :
mc_gross : 0.01
payment_status : Completed
item_number : [redacted]
address_city : [redacted]
payer_id : [redacted]
receiver_email : [redacted]@cascadetg.com
first_name : [redacted]
tax : 0.00
payer_status : unverified
business : [redacted]@cascadetg.com
payment_gross : 0.01
item_name : IPN Test
```

Figure 6-6. Email notification

It's easy to imagine adding additional logic to the `PayPalReciept.toHTMLString()` method for a more appealing and easier to understand design, but it contains all of the needed information.

The fax sending is encapsulated in a simple Java class, as shown in Example 6-4.

*Example 6-4. Fax sender code*

```
package com.cascadetg.ch06;

import cc.interfax.www.*;
import java.net.URL;
import org.apache.axis.client.*;
import javax.xml.namespace.QName;

/**
 * http://www.interfax.net/en/dev.html
 *
 * C:\devenv\axis-1_1\lib>java -classpath
 * commons-logging.jar;log4j-1.2.8.jar;wsdl4j.jar;axis.jar;
 * commons-discovery.jar;jax
 * rpc.jar;saaj.jar org.apache.axis.wsdl.WSDL2Java
 * http://ws.interfax.net/dfs.asmx?wsdl
 */

public class FaxSender
{
    public static final String TEXT = "TXT";
    public static final String HTML = "HTM";

    String text;
    String textType = TEXT;

    long result;

    /**
     * This is the main method used to send a Fax. You'll want to set
     * the text to be sent and the text type before calling this.
     *
     * @return true if successful, false if not.
     */
    public boolean sendFax()
    {
        try
        {
            // Create an instance of the Web Service Object
            InterFaxSoapStub ifs =
                new InterFaxSoapStub(
                    new URL("http://ws.interfax.net/DFS.asmx"),
                    new Service(new QName("SendCharFax")));
        }
    }
}
```

Example 6-4. Fax sender code (continued)

```
// Invoke the SendCharFax method
result =
    ifs.sendCharFax(
        FaxSenderTokens.faxUsername,
        FaxSenderTokens.faxPassword,
        FaxSenderTokens.faxTestFaxNumber,
        text,
        textType);

if (result > 0)
{
    // Positive returned value indicates that the fax was
    // sent successfully.
    // The return value is the Transaction ID.
    System.out.println(
        "Fax submitted properly. Transaction number: "
        + result);
    return true;
} else
{
    // Negative returned value indicates sending failure.
    // See error code definitions
    System.out.println(
        "Error sending fax! Error code " + result);
    return false;
}
} catch (Exception e)
{
    e.printStackTrace();
    return false;
}
}

public String getText()
{
    return text;
}

public void setText(String text)
{
    this.text = text;
}

public String getTextType()
{
    return textType;
}

public void setTextType(String textType)
{
    this.textType = textType;
}
```

*Example 6-4. Fax sender code (continued)*

```
    }

    public String getResult()
    {
        return Long.toString(result);
    }

    public static void main(String[] args)
    {
        FaxSender test = new FaxSender();
        test.setTextType(TEXT);
        test.setText("This is a test.");
        System.out.println(test.sendFax());
        System.out.println("Done!");
    }
}
```

The code shown in Example 6-4 uses a standard Apache Axis SOAP service. To use this code, you need to generate the bindings using the Apache Axis tools (as originally described in Chapter 3). The command that generates the InterFAX code is shown in Example 6-5.

*Example 6-5. Generating the InterFAX bindings*

```
C:\devenv\axis-1_1\lib>java -classpath commons-logging.jar;log4j-1.2.8.jar;wsdl4j.jar;axis.jar;commons-discovery.jar;jaxrpc.jar;saa.jar org.apache.axis.wsdl.WSDL2Java http://ws.interfax.net/dfs.asmx?wsdl
```

Arguably, it's harder to generate the HTML to send as the content of the fax than it actually is to make the call into the fax service and send the fax. As shown in Figure 6-7, you can see that the same HTML formatting used for the email is also used for the fax.

InterFAX does, of course, charge for faxes sent through their system. The nice thing is that you never need to worry about setting up or maintaining a fax system yourself. While it's possible to envision using the standard Java printing APIs to generate faxes that are sent via a local fax modem, this can rapidly become very complicated and expensive. It certainly takes more time to develop and maintain, plus you'll pay telephone charges.



If you intend to be working with faxes very much—in particular, if you wish to design nicely formatted faxes—you will likely go mad debugging the results using a normal fax machine. You may wish to consider using a fax-to-email provider such as j2.com (<http://www.j2.com>). There is, of course, a certain pleasant irony in using InterFAX to generate faxes only to then use the j2.com to turn the fax back into an email.

```

address_owner : 1
notify_version : 1.5
address_zip : 
payment_fee : 0.01
mc_fee : 0.01
address_state : A
receiver_id : ZE4CQEBXYWLYQ
quantity : 1
address_status : confirmed
payer_email : @cascadetg.com
txn_id : 
verify_sign : 
address_street : 
incoming_ip : 
payment_type : instant
mc_currency : USD
last_name : 
payment_date : 20:29:50 Jan 04, 2004 PST
txn_type : web_accept
paypal_address_id : 
address_name : 
address_country : United States
ebay_address_id : 
custom : 
mc_gross : 0.01
payment_status : Completed
item_number : 
address_city : 
payer_id : 
receiver_email : @cascadetg.com
first_name : 
tax : 0.00
payer_status : unverified
business : @cascadetg.com
payment_gross : 0.01
item_name : IPN Test

```

Figure 6-7. Received fax

The secure information (such as passwords and account names) has been broken into a separate class. The first, for PayPal, is shown in Example 6-6, and the second for the InterFAX service, is shown in Example 6-7.

*Example 6-6. PayPal tokens*

```

package com.cascadetg.ch06;

public class PayPalTokens
{
    // Put your PayPal registered email address here. You want to
    // make sure that payments are sent to the correct address.
    static final String paypalEmail = "test_account@cascadetg.com";

    // Mail configuration
    static final String mailhost = "smtp.mail.myisp.com";
    static final String mailhost_username = "mail_username";
    static final String mailhost_password = "mail_password";
}

```

The code assumes that you need to provide authentication to use the target SMTP server (typical for most ISPs today).

*Example 6-7. Fax tokens*

```

package com.cascadetg.ch06;

```

*Example 6-7. Fax tokens (continued)*

```
public class FaxSenderTokens
{
    public static String faxUsername = "fax_account";
    public static String faxPassword = "fax_password";
    public static String faxDevPartnerID = "12345678";

    public static String faxTestFaxNumber = "0014155551234";
}
```

Notice that the U.S. phone number is shown with a three-digit country code (001) prefixing the 415 (San Francisco area) phone number. Developers in non-U.S. countries are likely used to this method for specifying full phone numbers. If you're not, consult your local phone book for information on country codes and international dialing rules.

You've now used two different web services to provide for an automated sales system and managed to add both fax and email notification capabilities. It's easy to imagine adding even more capabilities as the support organization grows.