Java Tip 34: POSTing via Java

Learn how to POST data to Web servers in Java

Summary
This tip shows you how to send POST requests to Web servers and retrieve the response from an applet. *(800 words)*

TIP INDEX ▲ NEXT TIP ▶

By John D. Mitchell

The plethora of browsers in use turns the relatively simple act of performing POST-style HTTP requests into a bit of a pain. In this tip we shall provide the recipe for making POSTs simple again.

Everybody using the Web is at least passingly familiar with POST HTTP request. POST requests are used to send out things like HTML form data from a Web page to a Web server. A good example of a POST form is the feedback form at the bottom of this page. The browser sends the form data as part of the POST request, and the Web server sends back a response (which is usually in the form of another Web page).

The POST request, however, does not have to be used with an HTML form. It is just another HTTP request/response protocol. We can use it for whatever nefarious (or legitimate :-) deeds we care to. For instance, we can use POSTs to a CGI-bin script on a Web server rather than a raw socket to a database server. The advantage is you don't have to worry about some users not being able to use your applet because they happen to sit behind a firewall that refuses to allow random socket connections to the outside world.

You can see the results of invoking the Web server's CGI-bin script directly. Here is the Java POSTing applet in action:

The results of the Java applet should be here.

The code for dealing with posts is similar to the code for the Java application we developed for Java Tip #19 to copy files from Web sites. The core differences have to do with the security restrictions placed on applets.

```
URL                 url;
URLConnection       urlConn;
DataOutputStream    printout;
```

```
DataInputStream     input;

// URL of CGI-Bin script.
url = new URL (getCodeBase().toString() + "env.cgi");

// URL connection channel.
urlConn = url.openConnection();

// Let the run-time system (RTS) know that we want input.
urlConn.setDoInput (true);

// Let the RTS know that we want to do output.
urlConn.setDoOutput (true);

// No caching, we want the real thing.
urlConn.setUseCaches (false);

// Specify the content type.
urlConn.setRequestProperty
     ("Content-Type", "application/x-www-form-urlencoded");

// Send POST output.
printout = new DataOutputStream (urlConn.getOutputStream ());

String content =
     "name=" + URLEncoder.encode ("Buford Early") +
     "&email=" + URLEncoder.encode ("buford@known-space.com");

printout.writeBytes (content);
printout.flush ();
printout.close ();

// Get response data.
input = new DataInputStream (urlConn.getInputStream ());

String str;
while (null != ((str = input.readLine())))
     {
     System.out.println (str);
     textArea.appendText (str + "\n");
     }

input.close ();
```

First we build the URL to the CGI-bin script on the server. We use **getCodeBase()** to give us the URL from which the applet was delivered. This is important since the (untrusted) applet is only allowed to contact the host from which it came.

After opening the URL connection, we have to tell the runtime system explicitly what we are going to be doing. We want to both send and receive data on this connection so we set both the input and output flags. We then turn off the use of caching since we want to make sure that the data we read back from the server is actually in response to the data that we send it. Finally, we set the HTTP request's "content-type" field to be "application/x-www-form-urlencoded." That is necessary since some version of Netscape's browser has a bug which does not give out this proper content type when doing POSTs.

Now, after chanting all of those necessary magical incantations, we can create and open an output stream for that URL connection. We then write the data we want POSTed to the cgi-bin script on the server. It is *very* important that we close the output stream immediately after we write the data. If we don't close the stream then the cgi-bin script will sit waiting for more data (and so will not yet be sending its response).

Once we have sent out all of the data, we then open up an input stream for the URL connection and read all of the server's response.

That's all there is to it! ■

About the author
Alternately as employee, consultant, and principal of his own company, John has invested the last ten years in developing cutting-edge computer software and advising other developers. John co-authored *Making Sense of Java: A Guide for Managers and the Rest of Us* and *Dummies 101: Java Programming* and has published articles in programming journals. In addition to writing the Java Tips column for *JavaWorld*, he moderates the comp.lang.tcl.announce newsgroup. Reach John at john.mitchell@javaworld.com

Resources

- env.tcgi http://www.javaworld.com/javaworld/javatips/javatip34/env.tcgi
- common.pl http://www.javaworld.com /javaworld/javatips/javatip34/common.pl
- code http://www.javaworld.com/javaworld/javatips/javatip34/Happy.java
- Java Tip #19 http://www.javaworld.com/javatips/jw-javatip19.html