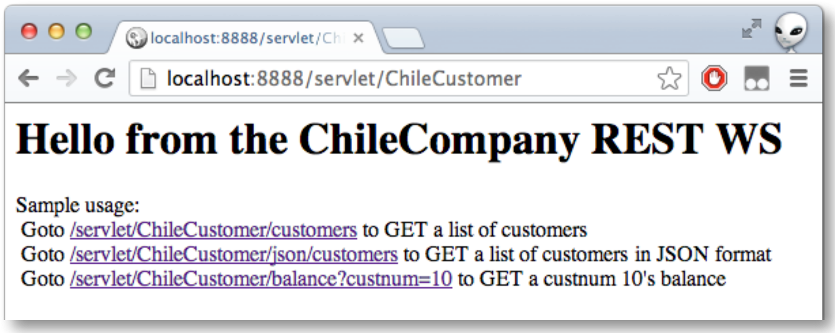
BBjServlets Serving Web Content

If you attended TechCon2013, you saw a demonstration that provided a REST-like Web Service for access to our Chile Company database. Servlets in general help to overcome latency and bandwidth limitations through a platform independent protocol implementation because all the work happens at the server and only the answer traverses the network to the client. This particular BBjServlet demo showed a BBj® program implemented as a BBjServlet, leveraging the HTTP protocol to provide responses from individual URLs to various clients. The demo's functionality included the ability to get a list of customers in multiple formats, including the ever-popular JSON (www.json.org) format, and the ability retrieve a particular customer's balance as illustrated in **Figure 1**. This article delves deeper into the demo's source code, providing a working example of one possible way to take advantage of the BBjServlet's capabilities by providing web-based content to a variety of disparate clients.


localhost:8888/servlet/Ch
localhost:8888/servlet/ChileCustomer

Hello from the ChileCompany REST WS

Sample usage:
Goto /servlet/ChileCustomer/customers to GET a list of customers
Goto /servlet/ChileCustomer/json/customers to GET a list of customers in JSON format
Goto /servlet/ChileCustomer/balance?custnum=10 to GET a custnum 10's balance

Figure 1. A web browser accessing the published service with sample URLs

Scrutinizing the Source

Let's take a closer look at the code necessary to run this demo.

The servlet implementation, an excerpt of which appears in **Figure 2**, begins by creating a BBj custom object, then obtains a servlet data object and registers for the ON_WEB_CONNECTION event in order to handle the incoming HTTP requests.

```
declare BBjServletData sData!

myServlet! = new myServlet()
sData! = BBjAPI().getServletData()
sData!.setCallback(sData!.ON_WEB_CONNECTION, myServlet!, "customer")
```

Figure 2. The beginning of the servlet implementation code

By ***Brian Hipple***
Quality Assurance Supervisor

As covered in a previous Advantage article, *Rest Easy - End Your WSDL Struggles* (links.basis.com/12rest), clients access the servlet with an HTTP GET request. Therefore, the customer method of our servlet is able to ascertain what information the client is requesting by examining the HTTP request's path information. The BBjServlet also supports URL parameters, providing an easy way for a client to refine further their communication with the servlet. The third sample URL from **Figure 1** shows how the client may include a parameter to specify a unique customer in the database by providing the desired customer number.

The custom `mySservlet!` class, referenced in the Callback code in **Figure 2**, is accessed whenever a client makes a connection to our published application. The beginning of the `mySservlet!` class definition is shown in **Figure 3**, and illustrates how the class's customer method obtains the HTTP request and creates an HTTP response object based on the provided BBjServletEvent.

```
REM ================================================================
REM Servlet Class
REM ================================================================
class public myServlet

    method public void customer(BBjServletEvent p_event!)

        declare BBjHttpResponse response!
        declare BBjHttpRequest request!

        LET chan = UNT
        request! = p_event!.getHttpRequest()
        response! = p_event!.getHttpResponse()
        response!.setContentType("text/html")
        open(chan)"JSERVLET"
        finished = 0
```

Figure 3. The beginning of the myServlet class

```
REM ==========================================
REM JSON Customers
REM ==========================================
if (request!.getMethod() = "GET" and request!.getPathInfo()="/json/customers" ) then
    REM Get customers
    customersMap! = #getCustomers()
    it! = customersMap!.keySet().iterator()

    json$ = "{ ""customers"" : ["
    while it!.hasNext()
        json$ = json$ + " {""name"":""" + str(it!.next()) + """},"
    wend
    json$ = json$(1,len(json$)-1)
    json$ = json$ + "] }"
    print(chan) json$
    finished = 1
endif
```

Figure 4. The servlet code that sends the list of customers to the client

The servlet handles each client's HTTP request by performing the appropriate SQL query to the Chile Company's customer table in order to retrieve the requested data. The servlet then builds a response and sends it back to the client by writing to the `JSERVLET` channel. **Figure 4** shows how the servlet handles a GET request for a list of the Chile Company customers in JSON format.

The code begins by calling a `getCustomers()` method, defined later in the source code, that executes an SQL statement and returns a java HashMap data structure with the list of customers. It then builds the string result by iterating over the HashMap, adding the customers into a JSON array object. When it has finished creating the string representation of the JSON response, the servlet sends the result to the client by printing the string to the JSERVLET channel.

Making Your Own Servlet

To make a BBjServlet available to clients, follow these three easy steps:

1. **Create the application configuration.** The BBjApplication object requires configuration information such as the program to run, which config file to use, and the working directory. The BBjAppConfig object encapsulates this program information and stores it so that the program can run as a servlet in an automated fashion. This is much like an autorun program or scheduling task.
2. **Obtain the servlet registry**. The BBjServletRegistry manages all currently running servlets and assists in publishing BBjAppConfig objects at specific paths. The registry is obtained from the BBjAdmin, in which administrative rights are required to add and remove servlets to and from the built-in Jetty Web Server.
3. **Publish the servlet.** Simply select a path, which starts with "/servlet/," and publish the application using the BBjAppConfig and BBjServletRegistry objects obtained in steps #1 and #2. As clients connect to the service, new interpreters start up to handle the request and are pooled, depending on load. As the load declines, these interpreters will shut down. Licenses are only checked out during request handling.

```
declare BBjAdmin admin!
declare BBjServletRegistry registry!
declare BBjConfig config!
declare BBjAppConfig application!

REM create
config! = BBjAPI().getConfig()
application! = config!.makeEmptyAppConfig()
application!.setWorkingDirectory(dsk("")+dir(""))
application!.setConfigFile(System.getProperty("com.basis.server.configDirectory")+"/config.min")
application!.setProgramName("chileCustomerService.src")

REM obtain
admin! = BBjAPI().getAdmin("admin", "admin123")
registry! = admin!.getServletRegistry()

REM publish
registry!.unpublish("/ChileCustomer", err=*next)
registry!.publish("/ChileCustomer", application!)

REM Goto http://localhost:8888/servlet/ChileCustomer to test
release
```

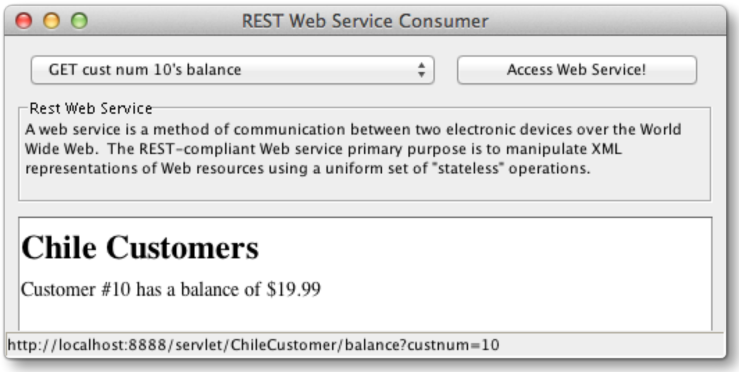
Figure 5. The code required to publish a servlet

Figure 5 shows the source code that encompasses all three steps.

Accessing Your Servlet

Any HTTP enabled application or tool can perform servlet invocation, including web browsers as shown in **Figure 1**. For the scope of this article, we are going to write the client in our favorite computer language, BBj. The BBj application provides a user interface shown in **Figure 6**, which allows the user to select the desired Web Service function to invoke from a listbox.

Upon user selection of a servlet function, the appropriate request creates an HTTP client using the functionality provided in the Java org.apache.commons.httpclient package. The request then executes, and the response returns and is displayed in a static text field, as shown in the code excerpt in **Figure 7**.



A web service is a method of communication between two electronic devices over the World Wide Web. The REST-compliant Web service primary purpose is to manipulate XML representations of Web resources using a uniform set of "stateless" operations.

Chile Customers

Customer #10 has a balance of $19.99

http://localhost:8888/servlet/ChileCustomer/balance?custnum=10

Figure 6. Our Web Service client written in BBj

```
CASE 3
   request!= new org.apache.commons.httpclient.methods.GetMethod(getBalanceURL$)
   client!.executeMethod(request!)
   returnString$=request!.getResponseBodyAsString()
   restView!.setText(returnString$)
   restStatus!.setText(getBalanceURL$)
break
```

Figure 7. An excerpt of the client code's SWITCH statement that communicates with the Web Service

Summary

As you can see, it takes very little code to create a servlet program and associated client programs. The fact that it is stateless and is built on a well-known and easy-to-use HTTP protocol makes it the most attainable way to share data and business rules with disparate systems. When your next project calls for applications to communicate over HTTP, think BBjServlet! ■

links.basis.com/13code

For more information, refer to *Rest Easy - End Your WSDL Struggles* at links.basis.com/12rest