

BBJSP Command Framework

A Look at the Role of Commands in BBJSP

by Richard Stollar

Hopefully, you have already read the article [Getting Started with BBJSP](#) as this article is a follow-on to that one. If you haven't read it, we strongly recommend that you do so as we'll be continuing to build the [create, read, update and delete](#) (CRUD) application we started.

In this article, we'll be looking at the role of commands in a BBJSP application and the deployment process. Once you've mastered the concepts of the command engine you'll be well on the way to full utilization of the BBJSP framework.

Overview

The BBJSP framework includes a page generator and an engine for processing business logic. Because the presentation layer should not handle the data, logic, and rules, BBJSP provides a Command Engine specifically for that purpose. Commands are small BBj® programs which perform some task like updating a database before redirecting to a BBJSP page or, in more advanced situations, to another command for further logic processing.

Completion of the Read Portion of the CRUD

You should have the first component of our CRUD application completed and, with all things being equal, you can open the page in your web browser and see all the salespeople listed in the table. In this article, we're going to complete the other main components; a form for editing those records and the business logic for interacting with the database. **Figure 1** shows a diagram of the application components including flow control.

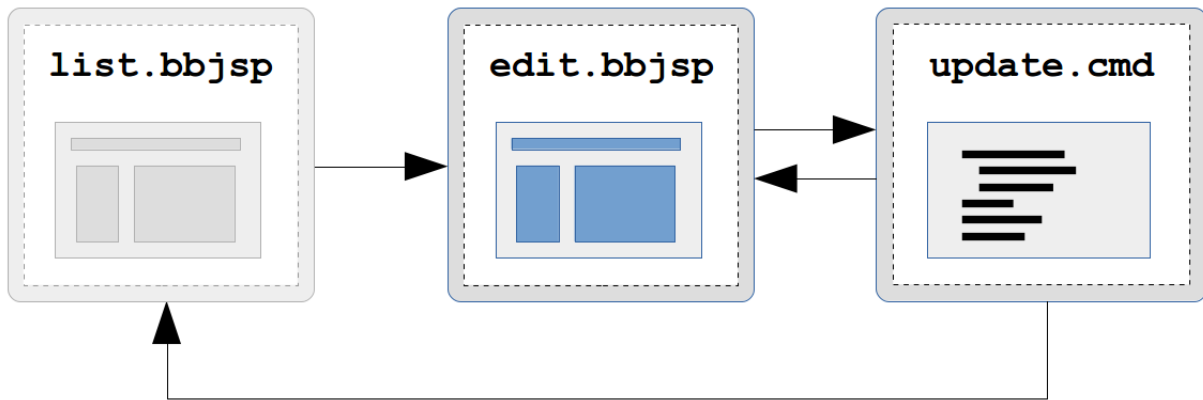


Figure 1: A diagram of the remaining sections of our CRUD application

Before we start writing our edit page, we need a way to link it to the `list.bbjsp` page. This way, the user can click on a salesperson in the listing page to bring up the edit page. We can do this in a few different ways:

1. We could add an edit button to the table
2. We could put a hyperlink on the salesperson list entries
3. We could write some clever JavaScript to spot a double-click event on a table row

We're going to use the second option and add a pair of hyperlinks: one for Edit which we'll add to the existing salesperson Code column, and one for Delete which we'll add in the last table column. **Figure 2** shows the updated table row source in the `list.bbjsp` file with the addition of our two new hyperlinks.

```

<tr>
  <td>
    <a href="${ROOT}/edit.bbjsp?op=U&id=${theRecord['SALESPERSON']}">
      ${theRecord['SALESPERSON']}
    </a>
  </td>
  <td class='lb'>${theRecord['NAME']}</td>
  <td class='lb'>${theRecord['ADDRESS']} ${theRecord['ADDRESS2']}</td>
  <td>${theRecord['CITY']}</td>
  <td>${theRecord['STATE']}</td>
  <td>${theRecord['ZIP']}</td>
  <td class='lb'>${theRecord['PHONE']}</td>
  <td class='lb'>
    <a href="${ROOT}/chileupdater.cmd?op=D&id=${theRecord['SALESPERSON']}">
      Delete
    </a>
  </td>
</tr>

```

Figure 2: The updated `list.bbjsp` excerpt with edit and delete hyperlinks

Now when we view the list in our browser, each salesperson entry will have a link placed on the Code field and a new delete link in the last column. **Figure 3** shows our updated listing page with the new hyperlinks.

Code	Name	Address	Phone	
BAR	Bartholomew Allen Reardon	8891 N. Coors Road Albuquerque NM 87114	5058976152	Delete
BOB	Becky Olivia Barstow	813 Central Avenue SW Apt 319 Albuquerque NM 87102	5052654721	Delete
CAU	Constance Anne Unger	1462 Old El Paso Drive Albuquerque NM 87103	5052993417	Delete
JAF	Andy foo	88 Erim Abq NM 93838	939939	Delete

Figure 3: The resulting list of salespeople with Edit and Delete hyperlinks

To complete the list page, we also need to give the user a way to add a new record. We can do this by placing one more link after the table, like this:

```
<a href="edit.bbjsp?op=CR"> Add new Salesperson </a>
```

The Edit Page

The edit page allows the user to add a new salesperson or edit an existing one from the listing page. We need an [HTML form](#) that displays the fields from the database table and populates the initial values for an update. **Figure 4** shows the source for our completed edit page.

```

<!DOCTYPE html/>
<%@ taglib uri='/WEB-CFG/tld/core.tld' prefix='c' %>
<%@ taglib uri='/WEB-CFG/tld/sql.tld' prefix='s' %>
<html>
<head>
<style>
  legend {
    padding: 0.2em 0.5em; border:2px solid green; color:green;
    font-weight:bold; background-color:white;
  }
  fieldset {
    border:2px solid green; -moz-border-radius:8px; -webkit-border-radius:8px;
    border-radius:8px; background-color:EEE;
  }
  label { display: block; float: left; font-weight: bold; width: 8em; }
  input { display: block; position: relative; height:2em; margin:3px; }
  input[type=submit] { display:inline; }
  input[type="text"] { width:60%; }
</style>
</head>
<body>
<c:equal value1='${param["op"]}' value2="U">
  <s:query template='tpl'
    var='result'
    datasource='ChileCompany'
    sql='SELECT * FROM SALESREP WHERE SALESPERSON=?'>
    <s:param value='${param["id"]}' />
  </s:query>
  <c:vget data='${result}' id="item" index="0" />
  <c:template id='record' template='${tpl}' data='${item}' />
</c:equal>
<form action="${ROOT}/chileupdater.cmd" method="post">
  <input type='hidden' name='op' value='${param["op"]}' />
  <input type='hidden' name='rate' value='${record["COMM_RATE"]}' />
  <input type='hidden' name='type' value='${record["COMM_TYPE"]}' />

  <fieldset>
    <legend>Sales Person : ${record['SALESPERSON']}</legend>
    <c:equal value1='${param["op"]}' value2="CR">
      <label for='id'>ID:</label>
      <input type='text' name='id' value="${record['SALESPERSON']}" />
    </c:equal>
    <c:notequal value1='${param["op"]}' value2="CR">
      <input type='hidden' name='id' value="${record['SALESPERSON']}" />
    </c:notequal>
    <label for='name'>Name:</label>
    <input type='text' name='name' value='${record["NAME"]}' />
    <label for='addr1'>Address:</label>
    <input type='text' name='addr1' value='${record["ADDRESS"]}' />
    <label for='addr2'>&nbsp;&nbsp;&nbsp;</label>
    <input type='text' name='addr2' value='${record["ADDRESS2"]}' />
    <label for='text'>City:</label>
    <input type='text' name='city' value='${record["CITY"]}' />
    <label for='state'>State:</label>
    <input type='text' name='state' value='${record["STATE"]}' />
    <label for='zip'>ZIP:</label>
    <input type='text' name='zip' value='${record["ZIP"]}' />
    <label for='phone'>Phone:</label>
    <input type='text' name='phone' value='${record["PHONE"]}' />
  </fieldset>
  <br>
  <div style='text-align:center'>
    <input type='submit' name='submitted' value="CANCEL" /> &nbsp;&nbsp;&
    <input type='submit' name='submitted' value="SAVE" />
  </div>
</form>
</body>
</html>

```

Figure 4: The edit.bbjsp page which allows the user to add or edit a salesperson

Now when you click on the first salesperson entry on the list page, the edit page displays as shown in **Figure 5**. When the user clicks the [SAVE] button, the form will POST to a [BBJSP Command](#) which updates the database. The form's action specifies where to send the data upon submission, and in this case, it is set to `chileupdater.cmd`. Building this update command will be our next step.

Sales Person : BAR

Name:	<input type="text" value="Bartholomew Allen Reardon"/>
Address:	<input type="text" value="8891 N. Coors Road"/>
	<input type="text"/>
City:	<input type="text" value="Albuquerque"/>
State:	<input type="text" value="NM"/>
ZIP:	<input type="text" value="87114"/>
Phone:	<input type="text" value="5058976152"/>

Figure 5: The edit page displaying the first salesperson entry

The Quintessential Command in BBJSP

The basic idea of a command is to execute some business logic, such as updating a database entry, then tell the framework what to do next. In our CRUD application, the command will be responsible for updating the database based upon the fields POSTed from the form on the edit page. Depending on whether the update succeeds, the command will tell the framework to either return to the listing page or to stay on the editing page to resolve the error.

When coding BBJSP commands, you must follow an [Object-Oriented Programming](#) (OOP) design approach and use [BBj Custom Objects](#) because all commands must implement the [BBjspCommand](#) interface. This interface mandates that you have an `execute(commandContext!)` method which the framework calls. The method should process the request and make the determination about how the framework will continue beyond the execution of the command. Let's look at the most simple command code before building our CRUD command.

```

class public MyCommand implements BBjspCommand

    method public BBjspCommandResult execute(BBjspCommandContext context!)
        result! = context!.getResult()
        rem == insert business logic here ==
        result!.setForward("ok")
        methodret result!
    methodend

classend

```

Figure 6: A custom execute command which implements the BBjspCommand interface

Looking at the command example in **Figure 6**, you can see that [BBjspCommand](#) receives a [BBjspCommandContext](#) in the execute method. Calling `getResult()` on the context returns a [BBjspCommandResult](#). Lastly, the code sets the command's result forward option and returns the result object.

The purpose of the forward option is to tell the framework how to proceed after the execute method completes. When you configure a command in EM, which we'll get to soon, you need to tell BBJSP about the possible exit points. The diagram in **Figure 7** shows both named exit points; `list.bbjsp` and `edit.bbjsp`, for the update command. We call each named exit point a *forward* and it points to new URL. Therefore, your command specifies the named exit point, or forward, for the framework to follow.

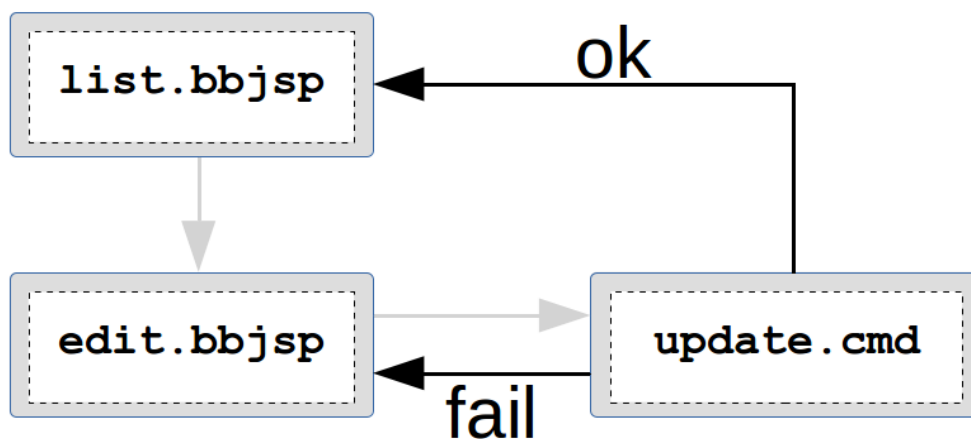


Figure 7: Command exit points, also known as forwards

The CRUD Command

Our CRUD application is fairly simple and only needs one command program to handle all updates because the HTTP request data includes the type of update. **Figure 8** shows the final command code that we're going to use:

```

class public UpdaterCommand implements BBjspCommand

method public BBjspCommandResult execute(BBjspCommandContext context!)
  declare BBjspCommandResult result!
  result! = context!.getResult()
  request! = context!.getRequest()
  let fail = 1
  let operation$ = request!.getParameter("op")
  let ch=sqlunt
  sqlopen(ch)"ChileCompany"
  switch(1)
  case operation$="CR"
    sql$ = "INSERT INTO SALESREP "
    sql$ = sql$ + "(SALESPERSON, NAME, ADDRESS, ADDRESS2, CITY, STATE, ZIP, PHONE) "
    sql$ = sql$ + "VALUES(?,?,?,?,?,?,?,?)"
    sp$ = request!.getParameter("id")
    na$ = request!.getParameter("name")
    a1$ = request!.getParameter("addr1")
    a2$ = request!.getParameter("addr2")
    cy$ = request!.getParameter("city")
    st$ = request!.getParameter("state")
    zc$ = request!.getParameter("zip")
    ph$ = request!.getParameter("phone")
    sqlprep (ch)sql$
    sqlexec (ch,err=*next)sp$,na$,a1$,a2$,cy$,st$,zc$,ph$; fail = 0
    break
  case operation$="U"
    sql$ = "UPDATE SALESREP SET NAME=?, "
    sql$ = sql$ + "ADDRESS=?, "
    sql$ = sql$ + "ADDRESS2=?, "
    sql$ = sql$ + "CITY=?, "
    sql$ = sql$ + "STATE=?, "
    sql$ = sql$ + "ZIP=?, "
    sql$ = sql$ + "PHONE=? "
    sql$ = sql$ + "WHERE SALESPERSON=?"
    sp$ = request!.getParameter("id")
    na$ = request!.getParameter("name")
    a1$ = request!.getParameter("addr1")
    a2$ = request!.getParameter("addr2")
    cy$ = request!.getParameter("city")
    st$ = request!.getParameter("state")
    zc$ = request!.getParameter("zip")
    ph$ = request!.getParameter("phone")
    sqlprep (ch)sql$
    sqlexec (ch,err=*next)na$,a1$,a2$,cy$,st$,zc$,ph$,sp$; fail = 0
    break
  case operation$="D"
    sql$ = "DELETE FROM SALESREP WHERE SALESPERSON=?"
    sp$ = request!.getParameter("id")
    sqlprep (ch)sql$
    sqlexec (ch,err=*next)sp$; fail = 0
    break
  swend
  sqlclose(ch)

  if fail then
    result!.setForward("FAIL")
    result!.addParameter("id",request!.getParameter("id"))
  else
    result!.setForward("OK")
  endif
  methodret result!

methodend

classend

```

Figure 8: The Commands.bbj program code for the CRUD application

This server-side code extracts the required data from the request, updates the database, and sets the forward. We now need to save and deploy this command.

Saving the Command Program File

BBJSP commands are BBJ program files and, like all BBJ programs, they should be locatable within the runtime PREFIX. When you first created your CRUD context you specified a **config.bbx** file. That file should contain a PREFIX line which includes locations for your BBJSP program files, including this **Commands.bbj** file. You should NOT put your BBJSP program source files in the same folder as you put your BBJSP pages because they would be viewable by the client in the browser.

Deploying the Command

Now that the command program exists on disk, we need to configure the command for the CRUD application in Enterprise Manager. Open EM then go to the Web > Context Configuration entry. Select your Chile context and you'll see an expandable section called BBJSP Commands. Expanding that section reveals a list of all defined commands. To add our first command, click on the plus icon next to the command table, as shown in **Figure 9**.

▼ BBJSP Commands

Path	Source File	Class Name



Figure 9: The BBJSP Commands list in Enterprise Manager

Figure 10: shows the fully-defined command configuration:

There's quite a bit going on in there but actually, it's straightforward. The first set of fields tells the BBJSP framework everything about the program to execute: the URL or path for the command, the program's source file, and the name of the class that we'll use.

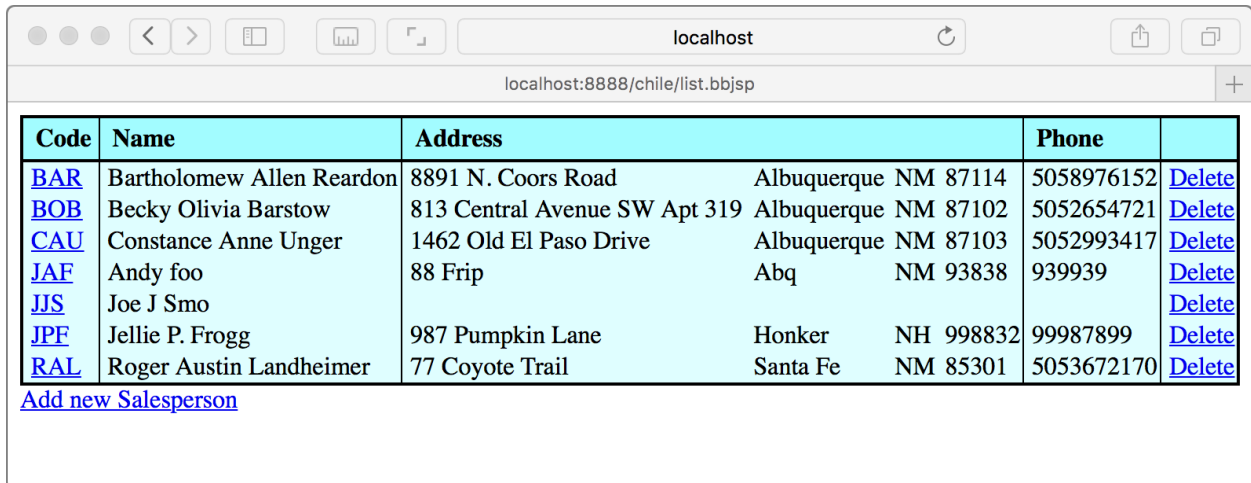
Below that, the table shows the redirects with their name, destination URL path, and their redirect flag. A redirect causes the server to send the HTTP response status code 302 with a location containing a new URL to the client's browser. When the browser receives the status, it makes a new HTTP request to the redirect URL. Setting the redirect flag is important because it defines what the user sees after the command has finished.

For this example, regardless of how we set command's redirect value, the user will see the updated salesperson listing after saving their edits. But if we set the redirect to `false` then the user will see `/chileupdater.cmd` in their browser's URL toolbar. That is not normally desirable so we'll set it to `true`. This way the user will see `/chile/index.bbjsp` in their URL toolbar which is what we want in most cases.

All that's left to do is to save our configuration and restart BBJServices to finalize our changes.

Running the Application

When we open our BBJSP application in the browser, we're presented with the list of salespersons, as shown in **Figure 12**.



Code	Name	Address	Phone	
BAR	Bartholomew Allen Reardon	8891 N. Coors Road Albuquerque NM 87114	5058976152	Delete
BOB	Becky Olivia Barstow	813 Central Avenue SW Apt 319 Albuquerque NM 87102	5052654721	Delete
CAU	Constance Anne Unger	1462 Old El Paso Drive Albuquerque NM 87103	5052993417	Delete
JAF	Andy foo	88 Frip Abq NM 93838	939939	Delete
JJS	Joe J Smo			Delete
JPF	Jellie P. Frogg	987 Pumpkin Lane Honker NH 998832	99987899	Delete
RAL	Roger Austin Landheimer	77 Coyote Trail Santa Fe NM 85301	5053672170	Delete

[Add new Salesperson](#)

Figure 12: The listing page showing the link to add a new salesperson

Clicking on a salesperson link loads the edit page, shown in **Figure 13**, allowing us to change the record in the database.

localhost

localhost:8888/chile/edit.bbjsp?op=U&id=BAR

Sales Person : BAR

Name: Bartholomew Allen Reardon

Address: 8891 N. Coors Road

City: Albuquerque

State: NM

ZIP: 87114

Phone: 5058976152

CANCEL SAVE

Figure 13: Updating a salesperson in the editing page

After modifying the data, we submit the form via the [Save] button and our code updates the record in the underlying database. We are then returned to the list page which reflects our changes to the data.

Summary

As you have read, updating records in a table can be quite simple when broken down into distinct units. We have all the key components of the [model-view-controller](#) (MVC) architecture in place and we can easily change one of the components, like the View component, without the need to recode the Controller. In our next article, AJAXing the CRUD, watch in wonder as we turn this little application into an [AJAX](#)-enabled single-page web application with Javascript and all the bells and whistles!



Check out the example available for download at links.basis.com/16code