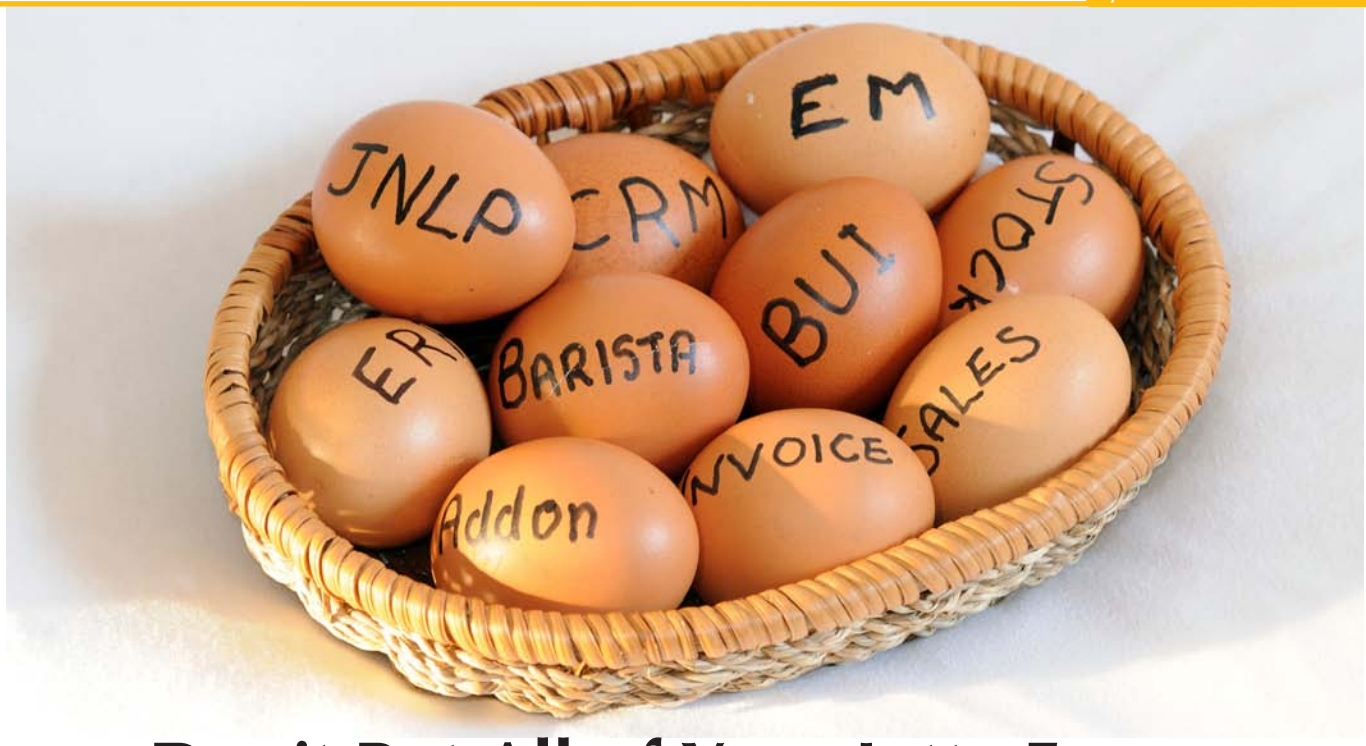


Don't Put All of Your Jetty Eggs in One Context

For those of you who like to look for the changes that BASIS makes in each BBj® upgrade, you may have noticed in version 14.12 a new file called **`jetty.xml`**. This addition wasn't, as some may think, a stealthy way for us to migrate our configurations into xml but rather a way to provide much greater flexibility in how to deploy your applications and pave the way for things to come. With this change, the benefits reach a wider audience. This article takes a closer look at the benefits, and will help you understand the added power we're providing and help you modify your deployments to take advantage of this power.

Configure Jetty the New Way

So why did BASIS decide to confuse users with yet another configuration file?

You may not have even thought about what is going on behind the scenes – you write your application and you configure it in Enterprise Manager (EM) to be accessible via a web browser. Perhaps you'll run it as a BUI application, or use Java Web Start/JNLP to launch it on your desktop via the Web, but either way, you will deploy it within the internal Jetty web server. However, there is only one server that is making all of your applications available and that means all of your applications are visible to all of your users. We decided that it was time to enhance that.

Let's start by looking at the new **`jetty.xml`** file that is behind the configuration. **Figure 1** shows the default file that BBjServices created automatically for you.

```
<?xml version="1.0" encoding="UTF-8"?>
<jetty-config>
  <contexts>
    <bbj docbase="$basis_home/htdocs" path="/" >
      <welcome-file>index.html</welcome-file>
    </bbj>
    <bbjem path="/bbjem"/>
    <resource name="htdocs" path="/files"
              resource="$basis_home/htdocs" />
    <resource name="documentation" path="/documentation"
              resource="$basis_home/documentation" />
    <resource name="baristahelp" path="/BaristaHelp/WebHelp"
              resource="$basis_home/barista/sys/help/WebHelp" />
    <resource name="addonhelp" path="/AddonHelp/WebHelp"
              resource="$basis_home/apps/aon/help/WebHelp" />
  </contexts>
  <admin user="admin" password="B1NhCfk1XmL0/u1WA8aoKQ=="
         encrypted="true"/>
</jetty-config>
```

Figure 1. The beginning of the myServlet class

Richard Stollar
Software Developer

Context is the key word here. A context is a place where an application exists. Remember that these applications are BUI, Web Start, web services, or web servlets, but currently there is only one context at **`http://{server}:8888`** or **`http://127.0.0.1:8888`**. All of your applications are in that context, but they don't have to be. BASIS provides a way for you to create new contexts and decide which applications are available in which context.

The Root BBj Context

The standard **`jetty.xml`** defines several contexts, the most significant of which is the BBj or root context that by default contains all of the applications. When you deploy an application – BUI, JNLP, or servlet – you can specify a custom name for the context; if you don't specify a custom context, then Jetty will assume the root context. Keep this root context in mind as you read through this article as we'll mention it from time to time.

Hostnames and IP Addresses

So let's step back a little bit and think about the hostname and what it means. When you open a page in your browser, you likely use a nice friendly name like google.com and the browser looks up google.com to find its IP address. Read on for the details of how that works or if you are already familiar with IP addresses, skip to the next section in this article 'Hostnames and how BBj Services Handles Them'.

The browser is the client and the server is, well, it's the server. All clients need to obtain an IP address for the server and this is done using hostname resolution. If the client doesn't already know the IP address for the server (it can be listed in a special hosts file), then the client uses a domain name server (DNS) to look up the IP address of the server.

The browser communicates with that address using the HTTP protocol and all requests have a header that contains the name of the server you want to talk to, google.com in our case. When the server receives the request, it looks at the header to decide if and how it will deal with that request. For example, www.google.com will go to the search engine we're all familiar with, whereas translate.google.com will go to the Google translator application. It could be that both of these friendly names resolve to the same IP address or it could be that they don't, but what's important is that it is the server's job to decide if and indeed how it will respond.

Hostnames and how BBjServices Handles Them

Your machine running BBjServices exists on a particular IP address, 10.0.0.10 or 192.168.0.10, for example. It will most likely have at least one hostname that should be known by the clients, **`jupiter`** for example, but you can use almost any hostname.

In addition to a specific hostname or IP address, your computer has what we call a *loopback device*. The loopback device is most commonly referred to as localhost or by the IP address 127.0.0.1.

Therefore, by default any hostname that resolves to the IP address of your server gains access to the root context we were talking about earlier. The URL **`http://juniper:8888/`** will display the BBjServices welcome page. All of your Web Start applications will be in **`/jnlp/*`**, your web services in **`/webservice/*`**, your servlets in **`/servlet/*`** and your BUI applications in **`/apps/*`**.

Creating Custom Contexts

Keeping all of your eggs in the same basket, or in the same *context*, is easier to administer but does not suit everyone. Suppose you have some applications that your customers access and another set of applications that your internal staff use. You may need to create a level of separation between them so that the external users cannot access the internal applications even if they can correctly guess their names. What you need to do is tell Jetty what hostnames it should accept and which of your precious applications live there. Here's how we go about it.

Consider that you are developing a servlet-based application that requires the deployment of multiple servlets, but at the same time, you want to restrict this application to a specific host or sub-domain. To achieve this goal, create a new context in **`jetty.xml`** with the basic XML content as shown in **Figure 2**.

```
<context name="myapp" path="/" docbase="\var\www\myapp" >
  <host>myapp.juniper</host>
  <welcome-file>index.html</welcome-file>
</context>
```

Figure 2. A custom context

Be sure to insert this context XML element immediately before the **`</contexts>`** marker.

Now, when you restart BBjServices, you will have a new context available called **`myapp`** into which you can deploy your servlets. We assigned the new context a **`<host>`** element that tells Jetty the hostname for the context. Your servlets, for example, will be available through **`http://myapp.juniper:8888/servlets/*`**. The **`\var\www\myapp`** folder specified in docbase is where static content such as images or html pages should be saved.

Deploying Your Servlets

If you've created a servlet-based application, you'll know that keeping the servlet deployed is extremely important. You will most likely dedicate a BBj program to deploying the servlets and add this servlet deployment program to the autorun list.

You will have to make a small change to your servlet deployment program in order to deploy your servlets to a specific context. Currently, your servlet deployer will contain a line of code similar to this:

```
registry!.publish("some_path", myServlet!)
```

If you leave this code alone, then Jetty will deploy your servlets in the root context as before. However, to specify a context, add its name as the first argument like this:

```
registry!.publish("myapp", "some_path", myServlet!)
```

Once deployed, users will only be able to access your servlet through that context. Creating your own custom contexts also ensures that sessions are unique and that session data for servlets in one context remains isolated from servlets in another context. Even if you deploy the same servlet to two different contexts, they will not share session data. The same client can access both servlets without corrupting data.

To simplify deployment, you can specify the servlets to register within a context when Jetty starts. To achieve this, add a servlet entry to your custom context. **Figure 3** shows a servlet entry that will make your servlet available on **`http://myapp.juniper:8888/servlets/myservlet`**.

```
<context name="myapp" path="/" docbase="\var\www\myapp">
  <host>myapp.juniper</host>
  <welcome-file>index.html</welcome-file>
  <servlet name="myservlet"
           program="MyServlet.bbj"
           config="/usr/bbj/cfg/config.bbx" />
</context>
```

Figure 3. Adding a servlet to the custom context

You can add as many servlet entries as required to the context and they will deploy automatically each time your BBjServices restart. Each servlet entry requires three pieces of information –

1. The ***name*** of the servlet as it will be used in the URL (in this case, **`myservlet`**).
2. The ***program*** file name that should be located in some path in your config.bbx file's prefix (in this case, **`MyServlet.bbj`**).
3. The ***config*** file used when executing the servlet (in this case, **`C:\BASIS\cfg\config.bbx`**); if you do not specify a config file, **`myservlet`** will use the default config.bbx.

In order for the automatic deployer to function, you need to set the admin entry in **`jetty.xml`** by specifying your admin user's username and password.

Specifying the Admin User's Username and Password

If you change the username or password that you use for your admin user, then you will need to update the admin entry in **`jetty.xml`**. To set the user credentials, edit **`jetty.xml`**, enter the correct user and password in the admin entry, set the *encrypted* flag to false and save the file. It would need to look something like this:

```
<admin user="admin" password="admin123" encrypted="false" />
```

Don't worry about entering the password in clear text because BBjServices will encrypt it automatically when they start and after restarting BBjServices, the entry will look more like this:

```
<admin user="admin" password="B1NhCfk1XmLO/u1WA8aoKQ==" encrypted="true" />
```

Deploying Applications to a Custom Context

Another element that you may wish to control is the deployment of your applications. Up until now, your applications have all lived in the same place, but in line with the other changes we've put in place we decided to give you greater control and flexibility of that too. You can specify a custom context for all your applications, that's BUI and Web Start/JNLP applications as well as web services. In much the same way described above for servlets, you can control which applications Jetty places in which context or you can again leave the deployment to the root context. The EM allows you to select a context from the list of available contexts and restrict access to a specified hostname within that context.

If editing configuration files is your thing then you'll know that all BUI and Web Start applications are configured through **`bui.ini`** and **`jnlp.ini`**, which are both in the **`basis_home\cfg folder`** and these files list the applications deployed. Inside each applications' configuration you can add a **`CONTEXT={context_name}`** to control which context the application will be deployed to. For example, **`CONTEXT=test`** will deploy the application to the test context. Each web service also has its own configuration file, which

can be found in the **basis_home\cfg\webservices** folder and again you can specify the context by adding **context={*context_name*}** to the file.

For those of you that don't want to mess with the configuration files directly, then the EM is the place to go. For each of the different application types you can configure the context where the application will go.

Whether you configured contexts with the EM or you edited the configuration files manually, the result is the same; you've placed your application within that custom context. Remember to define the custom context before assigning an application to it, or the application will be invisible and you really don't want that!

Figure 4 shows setting the context in action for web services, but the same principle applies on the BUI and JNLP configuration pages. When contexts have been defined in **jetty.xml** you can select which context the application will be deployed to from the dropdown list.

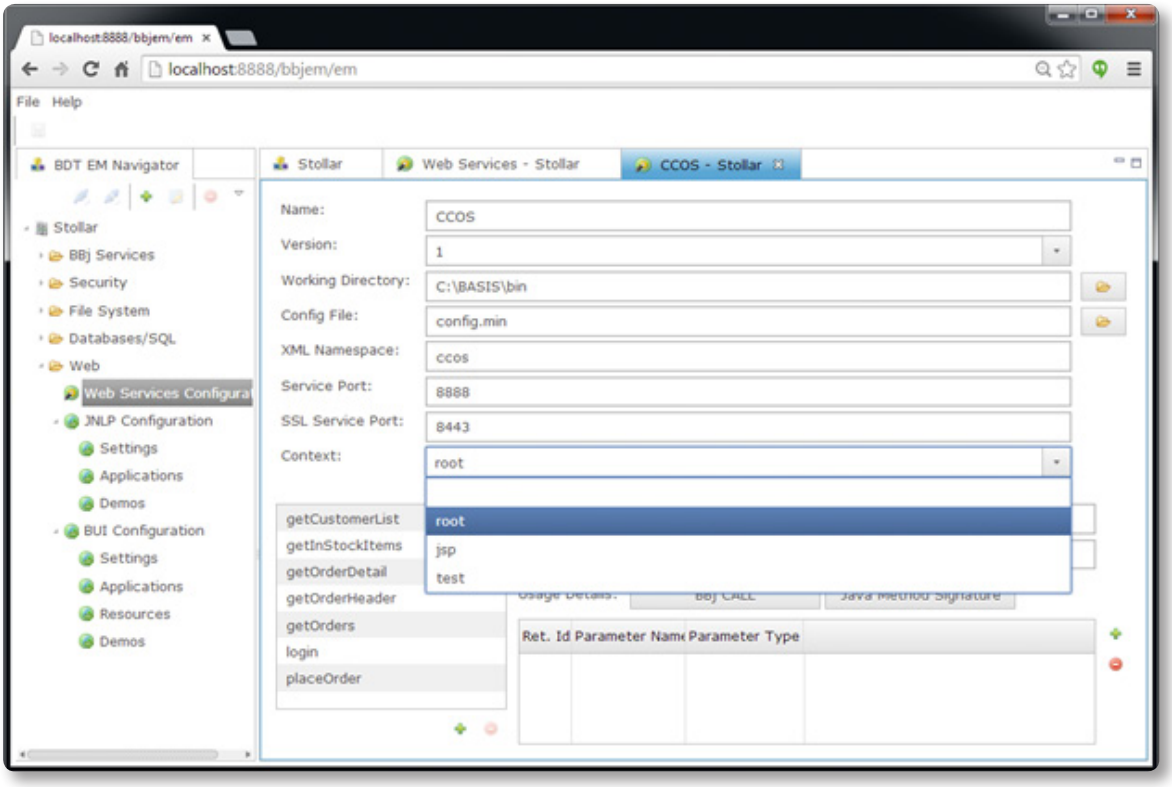


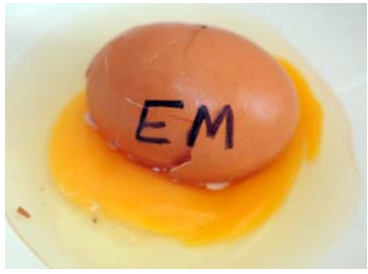
Figure 4. Set the context for your applications

Why did we go to all That Trouble?

We listen to the newsgroups and recently received the question, *"Is there any way to turn off the browser EM access without disabling our JNLP access?"* The problem was that everything was accessible on the server and you couldn't do much about it. Unless you took special action by installing another web server like an Apache server to filter requests, all your applications as well as EM were accessible.

Besides allowing you to restrict access to your applications, another significant benefit relates to BBj Servlets. Servlets are by design stateless but you almost certainly need to maintain some kind of user state that you would achieve through the underlying session. The session ID cookie identifies the client's session with each request but with all your servlets on the same URL (**http://jupiter:8888/servlet/***), then there is no real way to have some servlets use a different session from other servlets. Using contexts allows each session to maintain a unique session ID and thus unique session data.

Keep Enterprise Manager Safe



One of the applications that you may want to take control over is EM. You don't want someone trying to hack into your EM that resides on the same server as your public application, do you? Keep that employee you had to let go last week from acting out his need to mess with your server or delete your database.

Ask yourself two simple questions, "*Is my Enterprise Manager accessible over the Internet?*" and *"Do I still have **admin123** as my admin password?"* For many of you, the answer to both of these questions will be, "*Yes*!"

You can take control over the EM context and assign it to a specific hostname, ***localhost*** for example, and in doing so, ensure that the EM is only accessible on the physical machine or to a user logged in with remote desktop or a VNC connection. **Figure 5** shows a modified entry in `jetty.xml` that limits EM access to localhost.

```
<bbjem path="/bbjem" >
    <host>localhost</host>
</bbjem>
```

Figure 5. Restrict Enterprise Manager access

The example in **Figure 5** tells Jetty to allow access to the EM only through the specified hostname, `localhost`. You cannot even access it using the IP address unless you add another host entry such as `<host>192.168.0.99</host>`. How cool is that?! In addition, the hostname can be a subdomain like `bbjem.juniper.com`.

OK, so it's clear that you can set a hostname for your application contexts but...

What is the Path all About?

In addition to the hostname, a context has a path that really means the top of the context. We can create several contexts that are all accessible through the same hostname but have different path entries. Look at **Figure 6** and examine this more closely.

```
<context name="myapp" path="/" docbase="\var\www\myapp">
  <host>myapp.juniper</host>
  <welcome-file>index.html</welcome-file>
  <servlet name="myservlet"
           program="MyServlet.bbj"
           config="/usr/bbj/cfg/config.bbx" />
</context>
```

Figure 6. Contexts with a path

These context entries are the standard entries created automatically for you. Because none of them have a specific `<host>` entry, they are available to all hosts but the ***path*** is different providing `http://juniper:8888/files`, `http://juniper:8888/documentation`, and so on.

Earlier, you saw how **Figure 5** showed adding a hostname to the EM configuration to limit access to a specific hostname but the EM also has a path that is /bbjem, but again it doesn't have to be and it is easy to place EM on `http://em.juniper:8888/em` by using `path="/"` instead of `path="/bbjem"`.

Using Context Parameters and Attributes

When you start using contexts to control your applications, you may find the need to store context specific data. There are two types here, *parameters* and *attributes*. Parameters are defined in `jetty.xml` and are read-only to your servlets Whereas attributes can be created, edited, and destroyed on the fly within your servlet code. **Figure 7** shows how to define parameters in `jetty.xml`.

Your servlets have access to these parameters and attributes through the new BBjJettyContext object. The use cases are beyond the scope of this article but the examples in **Figure 7** are fairly self-explanatory.

```
<context name="myapp" path="/">
  <param name="database" value="myDatabase" />
  <param name="currency_code" value="USD" />
  <param name="currency_symbol" value="$" />
</context>
```

Figure 7. Context parameters in jetty.xml

The BBjJettyContext class provides access to the parameters and attributes within the context. You can obtain a BBjJettyContext class from the HttpSession, and the code sample in **Figure 8** demonstrates how to read a parameter in your BBj Servlet.

Remember that parameters can only be read by your servlets and if you need to store information in the context, then you should use attributes. Attributes stored in the BBjJettyContext are available to all servlets in your application, and are shared between requests and sessions. That means that the attributes are globally available to all visitors of the web application whereas session attributes are just available to a single user.

```
request! = p_event!.getBBjHttpRequest()
session! = request!.getSession()
context! = session!.getContext()
currencySymbol$ = context!.getInitParameter("currency_symbol")
currencyCode$ = context!.getInitParameter("currency_code")
```

Figure 8. Reading application context parameters

Using Java Elements

We have opened up the Java world too as Java provides some features that you may find useful. To incorporate Java elements into your application context, create a session classpath through the EM in the usual way and specify the classpath in the context by adding a classpath element as shown in **Figure 9**.

```
<context name="myapp" path="/" classpath="myappClasspath" >
  … other elements …
</context>
```

Figure 9. Adding a classpath to the Context

You can add third party Java servlets, context listeners, and request filters directly into your Jetty Server without having to use another application server. **Figure 10** shows an example of how to setup Java classes.

```
<context name="myapp" path="/" classpath="myappClasspath" >
  <j-listener class="com.acme.SystemInitializationListener" />
  <j-filter class="com.acme.Filter" filter="/*" />
  <j-servlet class="com.acme.BusinessProcess" mapping="*.bp" />
</context>
```

Figure 10. Adding Java elements to the Context

These are common elements of a Java web application. The specifics about how to write them and what you would use them for are well beyond the scope of this article. But when the need arises, they can be incorporated seamlessly into a custom context. Add a `<j-servlet>`, `<j_filter>`, or `<j_listener>` entry to define the components.

Sometimes a Java servlet needs initialization parameters and these can easily be added to the <j-servlet> tag as shown in **Figure 11**.

```
<j-servlet class="com.acme.BusinessProcess" mapping="*.bp">
  <param name="com.acme.dateFormat" value="yyyy MMM dd" />
</j-servlet>
```

Figure 11. Passing initialization parameters to a Java servlet

Summary

By introducing contexts, BASIS now allows you to take better control of how you deploy your Jetty Web Server applications by supporting multiple hosts and providing ways to restrict access to applications. Nevertheless, the good news is that you don't have to do anything unless you feel the need. Leave `jetty.xml` alone and everything will continue working the same as it has always done up until now. However, even if all you do after reading this article is restrict access to your EM, then I will consider this article to have been a great success. ■

Download and run the code samples