Asynchronous Triggers Modify the Copy

For a number of years, BBj® users have been able to configure triggers on data files to run a BBj program based on the particular action performed on those files. While this is a common feature in most enterprise level database management systems, standard BBj triggers have some limitations. The BBj program they execute runs inside the same Java Virtual Machine (JVM) as the Filesystem Server accessing the files so that the trigger program cannot execute on a remote installation of BBjServices or even locally in a separate JVM. Further, standard triggers must complete execution of the BBj program before the file system operation can finish, blocking that file operation until the trigger code returns. The new asynchronous trigger functionality allows triggers to fire asynchronously on a target as a result of data changes to the source file system.

One good use case for asynchronous trigger jobs is to address an often requested customer wish; "*I want to use replication to keep a copy of my data on a separate machine but I want to massage the data on the way to the replicated target.*" Of course this isn't replication because by massaging the data, you're no longer replicating the data; you're transforming the data! With asynchronous triggers, you can do just that and offload the transformation process to the target machine, leaving your production system largely unaffected.

Creating an Asynchronous Trigger Job

The best way to understand asynchronous triggers is to walk through the process of creating one. Adding an asynchronous trigger to a data file is very simple using a wizard quite similar to that used to setup replication and write auditing jobs. In fact, asynchronous triggers are built upon the replication framework, and as such they require exclusive access to the data files. Therefore, before attempting to create a replication job or an asynchronous trigger, you must first check the 'Exclusive File Access' setting in the Environment section of the BBjServices settings within Enterprise Manager. Then, restart BBjServices for this setting to take effect.

To launch the wizard, open the list of asynchronous trigger jobs by double-clicking the 'Asynchronous Trigger Jobs' node under 'File System' in the Enterprise Manager navigator. Create a new job by clicking [Add], give the job a name, and choose whether to monitor an entire database or a list of manually selected files as shown in **Figure 1**.

By Jeff Ash
Software Engineer

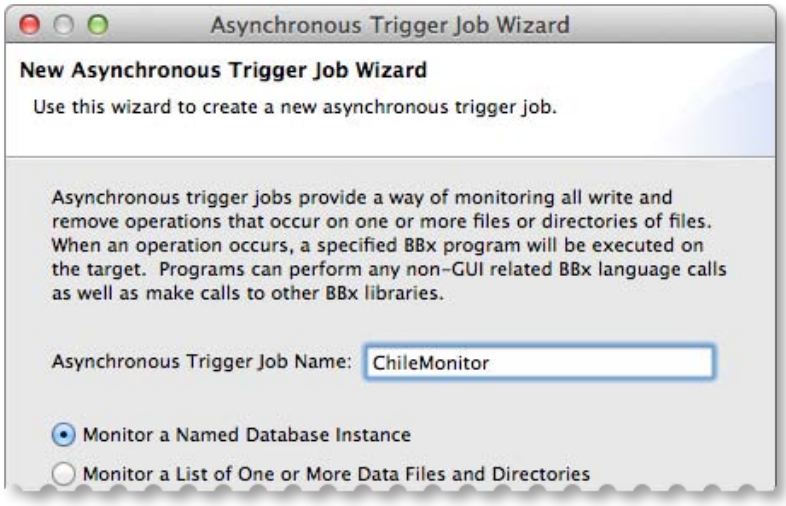

Figure 1. Choose the type of trigger job to create and name it

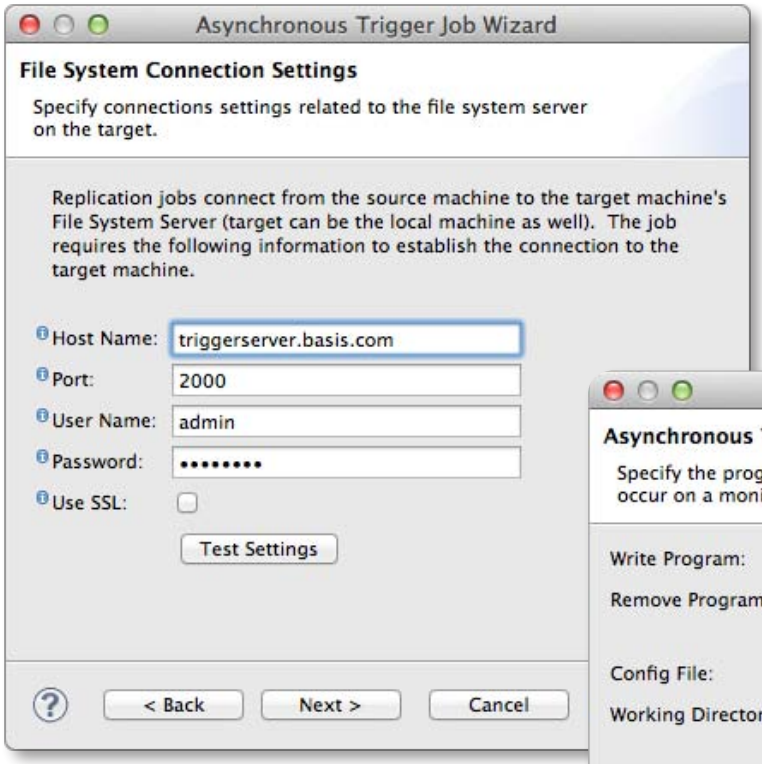

Figure 2. Specify the server to run the trigger programs

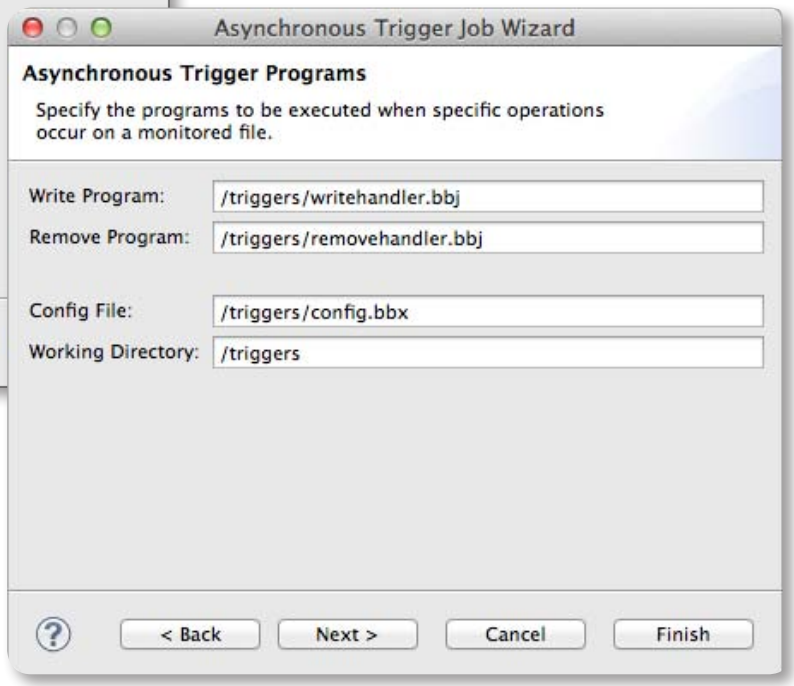

Figure 3. Specify the runtime details

The next step is to indicate to the server where the triggers should run, which can be `localhost`, and then enter the 'User Name' and 'Password' for that server (**Figure 2**).

Next, configure the BBj programs to run when a write or remove operation occurs (see **Figure 3**). Anytime a write operation occurs on a monitored file or directory, the 'Write Program' runs. The same is true for removing a record from a file with respect to the 'Remove Program'.

Finally, configure the list of files and directories to include in the job (**Figure 4**), and optionally, the list to exclude from the job. Use the subsequent wizard page to specify exclusions.

Writing the Trigger Handlers

Remember the 'Write Program' and the 'Remove Program' – `writehandler.bbj` and `removehandler.bbj` – that we specified in **Figure 3**? They need to exist for the asynchronous trigger job to work. Fortunately, writing a trigger handler is quick and simple. These special programs support the complete functionality of BBx® with one limitation – they cannot include any user interface-related operations since the programs run inside the server, often in a headless environment.

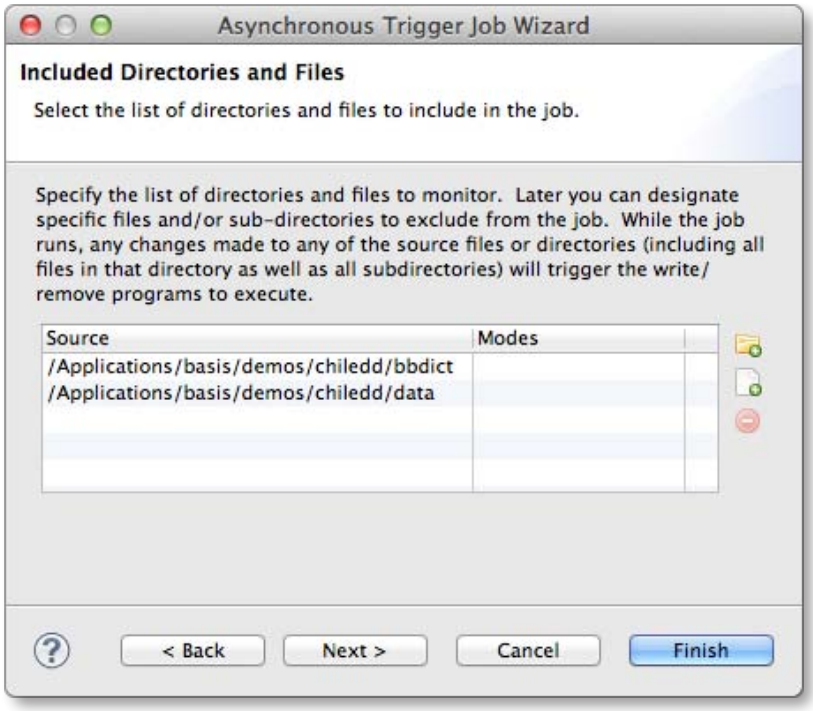

Figure 4. Specify the files and directories to monitor

Another important difference is that BBj treats trigger handlers in a special way, making a special object available to the environment that provides valuable information about the file operation that triggered the programs execution**. Figure 5** displays a short example showing some of the special information available to a trigger handler.

```
REM Get a Trigger object from the BBj File System
tdata! = BBjAPI().getFileSystem().getTriggerData()

REM Get the user who performed the operation
user$ = tdata!.getUser()

REM Get the name of the file
filename$ = tdata!.getFilename()

REM If a write handler, get the record written to the file
writeBuffer$ = tdata!.getWriteBuffer()

REM If a remove handler, get the key value specified for the remove
key$ = tdata!.getKey()
```

Figure 5. An example program showing some of the information available to a trigger handler

Summary

Asynchronous triggers provide a seamless and efficient way to execute specific BBj code whenever write and remove operations occur on a list of monitored directories and files – all without blocking the file operation. The beauty of this new feature is that you can configure your triggers to execute the BBj code on any BBjServices server available on the network, remote or local. Using the BBjTriggerData object available to the trigger handlers, you have all the information necessary about the file and operation necessary to provide powerful processing. This new 15.0 feature is available for preview today so why not install it and try it out? ■

- See asynchronous triggers in action on YouTube
- Refer to BBjTriggerData in the online documentation
- Download and run the code sample

Stimulate Your Brain!

30-minute webinars that make a difference!

links.basis.com/javabreak