INTERPRETATION OF MAREK'S DAEMON DEMO

Elliott McCrory, 1 July 2008

INTRODUCTION

The environment for creating daemon processes within the LHC control system is based on the tools that are being used throughout the control system. Primarily, it is suggested that daemons be developed as Spring beans. This is a new concept for Fermilab/LAFS developers, and (likely) for accelerator physicists in general. But this framework, while complicated to understand in its entirety, is easy to use. Marek Misiowiec has created an example for a publishing daemon, and the "business" is entirely contained in one class. This class is easy to extend and to modify.

First, I will describe in full detail how this example works. Then I will describe how I modified the existing example, ignoring a lot of the Spring details, to create a non-trivial daemon.

This document is unnecessary if you want to create your daemon without understanding the underlying Spring-based pieces. In this case, you can probably get away with reading only the last section, A New Publishing Class.

DAEMON EXAMPLE USING SPRING

Marek Misiowiec has created a simple demo of how to implement a simple class of daemons in the CERN AB/CO—LSA controls environment. It is stored in CVS as `lsa/lsa-demo-daemon`.

There are two ways to run the publishing in this package, through `ConcentrationStarter` or through `ParameterPublisherExample`. The former is how Marek does publishing in the BLM concentrators, and the latter is a simplified example of publishing one parameter without "concentration."

For our use for Instrumentation Daemons, I believe that `ParameterPublisherExample` is a better, simpler way to do the job at hand, so I will focus on this.

RUNNING THIS EXAMPLE

1. Check out the code from CVS and configure the project in the normal way.
2. Run `ParameterPublisherExample.java` as a Java application in the background. This publishes a new value every five (5) seconds
3. Run `ParameterMonitor.java` as a Java application. This subscribes to the data published by `ParameterPublisherExample` and shows some output.

KEY ELEMENTS OF THIS EXAMPLE

JMS PUBLISHING

Data are published using JMS, but using a JAPC/LSA wrapper. This wrapper ensures consistent use of JMS across applications, and all JMS publishing should be done this way. In CERN AB/CO, a topic must be created for your publisher. Marek chose to use an existing topic, "LHC_BLM", for this demo. You can see this specified in two places:

1. Primarily, this topic is specified in the file `concentration.parameters`, as the property `demo.concentrated.parameter`.
2. In the file `ConcentrationStarter.java`, line 66: `result.setPropertyName()`. This occurrence only specifies the default value of `demo.concentrated.parameter` if it is not otherwise set.

For the Fermilab daemon processes, it will be necessary to create a new publishing topic. It is not completely clear at this time if one topic for all the daemons, or separate topics for each individual daemon should be created. We should discuss this.

CLASSES IN THE PACKAGE

- **ConcentrationAdapter**. This class implements the two JAPC adapters for receiving parameters. When a new parameter comes in, this class sorts out which one is the one we are interested in and returns it (as an AcquiredParameterValue).
    - Implements cern.japc.group.ParameterGroupValueReceivedAdapter, cern.japc.spi.adaptation.AcquiredParameterValueAdapter
- **ConcentrationStarter**. This is the main class for handling the concentrator-based publishing.
    - Implements java.lang.Runnable
- **ParameterHolder**. Used in the Concentrator publishing example. This becomes the Spring bean that handles the publishing of our custom parameter.
- **ParameterMonitor**. This is a simple class that only subscribes to the parameter that is being published. IT allows you to see that the parameter is actually being published properly.
- **ParameterPublisher**. Becomes the Spring bean that handles the publishing in the simplified publishing example.
- **ParameterPublisherExample**. This is the main class for handling the simplified (non-concentrator) publishing. This class is listed in Appendix 1, PARAMETERPUBLISHEREXAMPLE.JAVA.
    - Implements `java.lang.Runnable`
- **PropertyPlaceholderConfigurerExt**. This is a utility class that extends the Spring class (see bullet below) so that the configuration parameters are read in early in the start-up cycle, and they can be used in the Spring XML file.
    - Extends `org.springframework.beans.factory.config.PropertyPlaceholderConfigurer`
- **ServiceLocator**. This class is used by both publishing examples to return the beans that do the required operations (services).
- **BCTReader.java.** A class copied from an SPS expert that subscribes to the SPS beam current transformer for a particular SPS user. This class is used by …
- **ExtendedParameterPublisher.java.** This class utilizes BCTReader to get the beam intensity from the SPS for a specific user cycle. Then it calculates three new numbers based on this reading and the elapsed time between readings: Total beam intensity over the last 5 minutes; Average beam power over the last five

minutes (pretty kludgy and probably not perfectly correct); a "peak power" value, which is the measured intensity over an interval of 0.1 seconds. See the next section for discussion of these last two classes.

SPRING COMPONENTS

Much of the work in binding together the objects in this demo is specified in the XML file, `publishing-spring-beans.xml` (see Appendix 1, PARAMETERPUBLISHEREXAMPLE.JAVA). In this file, four (4) beans are created:

1. **`placeholderPublishing`**. This bean is created so that the Spring framework starts properly on your beans. In particular, it causes the properties files to be loaded so that the rest of the Spring beans can utilize the values defined there. For example, a property called `property.one` would be accessed in subsequent beans with the syntax `${property.one}`
    - Class: `cern.lsa.demo.daemons.concentration.PropertyPlaceholderConfigurerExt.` This is a helper class in this package that extends a Spring class, `PropertyPlaceholderConfigurer`, that deals with the configuration of the Spring environment.
    - Location of the property file: `concentration.properties`
    - The property `systemPropertiesModeName` is set to the value `SYSTEM_PROPERTIES_MODE_OVERRIDE`; this means that properties that are already set in the system environment take precedence over the properties set in any other way (for example, from the properties file).
2. **`DemoJmsPublishService`**. This bean configures the JMS Publishing class by setting the property, `jmsTopicPrefix`, to the desired value, `${demo.jms.topic.prefix}` (which is, in this example, set to `CERN.LHC.BLM` in the file `concentration.properties`).
    - Class: `cern.japc.ext.remote.jms.JmsPublishServiceImpl`
3. **`DemoParameterPublisher`**. Creates the bean for doing the publishing
    - Class: `cern.lsa.demo.daemons.concentration.ParameterPublisher`
    - Uses Spring to "inject" objects into this class:
        - The publishing object, `DemoJmsPublishService`, using the setter method, `setJmsPublisher()`.
        - It also specifies the parameter name to be published, `${demo.concentrated.parameter}`, through the setter `setPublishedParameter()`.
4. **`ParameterPublisherExample`**.
    - Class: `cern.lsa.demo.daemons.concentration.ParameterHolder`
    - Injects the previous bean for publishing into the main class.

MAIN CLASS DISSECTED

Java.lang.Runnable.run()

Marek has chosen for `ParameterPublisherExample` to implement `java.lang.Runnable`. This means that all of the meaningful, run-time code is contained in the method `run()`. Here is the code for that method (the entire listing is in Appendix 1).

```
public void run() {
     publisher =
        ServiceLocator.<ParameterPublisher> getSingleton("DemoParameterPublisher");

     System.out.println("Concentration started!");

     while (true) {
         try {
             // sleep a bit
             Thread.sleep(5000);
         } catch (InterruptedException ie) {
             ie.printStackTrace();
         }
         try {
            System.out.println(
                 "publishing dummy value " + (System.currentTimeMillis()/1000));
            publisher.publish(createDummyParameterValue());
         } catch (Exception e) {
             e.printStackTrace();
         }
       }
    }
```

The first statement gets the publishing object, created previously by the Spring environment. You can use this as it is with no modification.

It has been decided for this example that new values will be published once per five seconds. So the publisher sleeps for 5000 milliseconds.

The method `publisher.publish()` takes as it's argument a standard `cern.japc.AcquiredParameterValue` object. This object is created by the utility routine `createDummyParameterValue()`.

Thus, in the `run()` method, there are only two lines that would need to be changed—how long you sleep between publishing and how you create the `AcquiredParameterValue`. For the foreseen LAFS daemons, the timing will be wholly determined by the reception of data from the front ends, so there will likely not be a need to `sleep()`.

Method createDummyParameterValue()

There is a great deal of flexibility built into JAPC parameters. Consequently, it takes several steps to create a parameter from scratch. Here is the method, adding line numbers for clarity:

1	`private AcquiredParameterValue createDummyParameterValue() {`
2	`Map<String, SimpleParameterValue> valuesToReturn =` `new HashMap<String, SimpleParameterValue>(1);`

3	`valuesToReturn.put(`
	`"crateNames",`
	`ParameterValueFactory.newParameterValue(new String[]{"dummy"}));`

4	`MapParameterValue resultValue =` `ParameterValueFactory.newParameterValue(valuesToReturn);`
5	`AcquiredParameterValueImpl result = new AcquiredParameterValueImpl();`

6	`result.setHeader(`
	`new ValueHeaderImpl(System.currentTimeMillis()*1000, // Acq stamp`
	`0, // cycleTimestamp`
	`ParameterValueFactory.newSelector(null)));`

7	`result.setParameterName(` `System.getProperty("demo.concentrated.parameter", "LHC_BLM/Demo"));`
8	`result.setValue(resultValue);`
9	`return result;` `}`

Line 1 and 2: The method returns an `AcquiredParameterValue`, which simply holds a `Map<String, SimpleParameterValue>`, as seen here.

Line 3: The key to the value (which is the "property") is named "crateNames" here, and the value is "dummy".

Line 4: An object of type `cern.japc.MapParameterValue` is created based on the Map created in line 3.

Line 5: A new `cern.japc.AcquiredParameterValue` is created, using the "`Impl`" class. (This pattern is seen quite frequently in the code produced by AB/CO/AP.) This will be the return value.

Line 6: The header for the result is created. The time stamp is taken from the system clock; there is not cycle stamp and there is no timing selector.

Line 7: The name of the parameter is set, based on the name specified in the configuration. This step utilizes the flexibility of the Spring system, but it may be argued that your daemon will know, a priori, the name of its parameter(s), so going to the configuration file might not be necessary.

Line 8: The value is set!

Line 9: The parameter is returned, ready to be published.

A NEW PUBLISHING CLASS

I have modified the example from Marek to create a somewhat more realistic daemon. This new daemon is contained in the class ExtendedPublisherExample.java, as described above. This class utilizes BCTReader to get the beam intensity from the SPS for a specific user cycle. Then it calculates three new numbers based on this reading and the elapsed time between readings: Total beam intensity over the last 5 minutes; Average beam power over the last five minutes (pretty kludgy and probably not perfectly correct); a "peak power" value, which is the measured intensity over an interval of 0.1 seconds.

Other than this new class, the only other change I made was to the file `concentration.parameters`—I changed the value of `demo.concentrated.parameter` to "`LHC_BLM/Demo2`" so as not to conflict with the example Marek wrote.

Here is a sample of the output of the publisher/daemon, `ExtendedParameterPublisher.java`:

```
17 Intensity: 2.6169000244140625E12, time=Tue Jul 01 15:20:50 CEST 2008
Time interval: 288.001 seconds. Total charge=2.2203998565673828E11
    Ave power=55.58534638095629 Watts,
    peak power=1886732.1620556451 Watts.
Publishing LHC_BLM/Demo2
18 Intensity: 2.6803399658203125E12, time=Tue Jul 01 15:21:38 CEST 2008
Time interval: 287.999 seconds. Total charge=2.632760009765625E12
    Ave power=659.0880148069377 Watts,
    peak power=1932471.0808883954 Watts.
```

```
Publishing LHC_BLM/Demo2
19 Intensity: 0.0, time=Tue Jul 01 15:22:26 CEST 2008
Time interval: 288.0 seconds. Total charge=2.9816799926757812E12
    Ave power=746.4344324199812 Watts,
    peak power=0.0 Watts.
Publishing LHC_BLM/Demo2
20 Intensity: 2.8547998046875E12, time=Tue Jul 01 15:23:14 CEST 2008
Time interval: 288.0 seconds. Total charge=2.8547998046875E12
    Ave power=714.6712179439095 Watts,
    peak power=2058253.1076784593 Watts.
Publishing LHC_BLM/Demo2
```

And here is a sample of the output of the program (`ParameterMonitor.java`, which is not modified) that is monitoring these virtual parameters:

```
++ got value for LHC_BLM/Demo2 Cycle=[0] AcqStamp=[1214918450]
    Average Power: 55.58534638095629
       Peak Power: 1886732.1620556451
     Total Charge: 2.2203998565673828E11
++ got value for LHC_BLM/Demo2 Cycle=[0] AcqStamp=[1214918498]
    Average Power: 659.0880148069377
       Peak Power: 1932471.0808883954
     Total Charge: 2.632760009765625E12
++ got value for LHC_BLM/Demo2 Cycle=[0] AcqStamp=[1214918546]
    Average Power: 746.4344324199812
       Peak Power: 0.0
     Total Charge: 2.9816799926757812E12
++ got value for LHC_BLM/Demo2 Cycle=[0] AcqStamp=[1214918594]
    Average Power: 714.6712179439095
       Peak Power: 2058253.1076784593
     Total Charge: 2.8547998046875E12
```

APPENDIX 1, ParameterPublisherExample.java

```
package cern.lsa.demo.daemons.concentration;
import java.util.HashMap;
import java.util.Map;

import cern.japc.AcquiredParameterValue;
import cern.japc.MapParameterValue;
import cern.japc.SimpleParameterValue;
import cern.japc.factory.ParameterValueFactory;
import cern.japc.spi.AcquiredParameterValueImpl;
import cern.japc.spi.ValueHeaderImpl;

/**
 * This class starts the LHC BLM concentrator server. Command "exit" is used to stop
the server.
 *
 * @version $Revision: 1.1 $, $Date: 2008/06/30 15:56:30 $, $Author: emccrory $
 *
 * to run needs : -DserverId=LhcBlmServer
 */
public class ParameterPublisherExample implements Runnable {
```

```
    private static Thread serverThread;

    private ParameterPublisher publisher;

    public void run() {
        this.publisher = ServiceLocator.<ParameterPublisher>
getSingleton("DemoParameterPublisher");

        System.out.println("Concentration started!");

        while (true) {
            try {
                // sleep a bit
                Thread.sleep(5000);
            } catch (InterruptedException ie) {
                ie.printStackTrace();
            }
            try { // and then prepare a dummy value to publish it using custom virtual
parameter publisher:
                System.out.println("publishing dummy value " +
(System.currentTimeMillis()/1000));
                this.publisher.publish(createDummyParameterValue());
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }


    private AcquiredParameterValue createDummyParameterValue() {
        Map<String, SimpleParameterValue> valuesToReturn = new HashMap<String,
SimpleParameterValue>(1);
        valuesToReturn.put("crateNames",
                           ParameterValueFactory.newParameterValue(new
String[]{"dummy"}));
        MapParameterValue resultValue =
ParameterValueFactory.newParameterValue(valuesToReturn);
        AcquiredParameterValueImpl result = new AcquiredParameterValueImpl();
        result.setHeader(new ValueHeaderImpl(System.currentTimeMillis() *
1000,      // acqTimestamp
                                             0,    // cycleTimestamp
                                             ParameterValueFactory.newSelector(null)))
;
        result.setParameterName(System.getProperty("demo.concentrated.parameter",
"LHC_BLM/Demo"));
        result.setValue(resultValue);
        return result;
    }

    public static void main(String[] args) {
        ParameterPublisherExample ppe = new ParameterPublisherExample();
        serverThread = new Thread(ppe);
        serverThread.start();
    }


    /**
     * @return the publisher
     */
    public ParameterPublisher getPublisher() {
      return publisher;
    }
```

```
    /**
     * @param publisher the publisher to set
     */
    public void setPublisher(ParameterPublisher publisher) {
      this.publisher = publisher;
    }
}
```