Example - Brad Harvey

Spring Integration

Sample Implementation

> This is a slightly modified version of what we've implemented at my company. I haven't tested my modifications, so I apologise if it doesn't work exactly but it should give you an idea of what can be done. It should work with quickfixj 1.0.0/Spring 1.2.5.
>
> Brad Harvey

This was all written before MINA 0.9 was being used. MINA 0.9 has its own spring integration classes, and (at least in theory in my head) it would be really nifty if quickfixj just hooked into these as it would open up some extra customisation possibilities such as SSL Filters.

> Brad, thanks for adding this information. This is very similar to an approach I've been thinking about. Barry Kaplan also prototyped an somewhat different approach and I've added his example configuration files for discussion purposes. I'd like to create an abstraction for the session settings so an application could either use an existing QF settings file, create the settings directly in a Spring configuration, or plug in an external session configuration provider (possibly supporting dynamic creation and deletion of sessions). I'll look into the Mina 0.9 Spring integration classes and see how those could be incorporated as well. I'm thinking of restructuring the QFJ project to allow a sibling quickfixj-spring subproject that would be released separately from the core quickfixj code. I also want to try using some of the new Spring 2.0 features to make the configuration file easier to manage. I'll add a page soon with ongoing experiments that I'm doing as well. Thanks again. Steve Bate
>
> ---
>
> No worries. I agree that being able to supply other implementations of SessionSettings would be useful. It'd also be nice to have a common interface for initiator/acceptor.
>
> I think in some ways SessionSettings is a bit anti-spring since you're not making use of dependency injection for most of the configuration. Moving away from SessionSettings takes you towards Example - Barry Kaplan which is more flexible at the expensive of a steeper learning curve for quickfix and spring newbies.
>
> I guess it depends whether you want quickfixj powered by Spring internally or just Spring friendly for those who want to use it. Presumably the latter if you're leaning towards an optional subproject, which I think is a good idea.
> Brad Harvey

Application Context

Here's the basic application context. You supply an Application implementation and quickfix will start up once the application context is loaded. Aside from your Application you also need sample.QuickFixBean which does most of the work.

quickfixContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans default-lazy-init="false" default-dependency-check="none"
default-autowire="no">

 <bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigu
rer">
  <property name='locations'>
   <value>config.properties</value>
  </property>
  <!-- defaults - you can override this in config.properties or by
specifying them as system properties -->
  <property name="properties">
   <props>
```

```
    <!-- path to sessions configuration file -->
    <prop key="quickfix.config">/path/to/quickfix.cfg</prop>
    <!-- true = initiator, false = acceptor.  Haven't actually tried as an
acceptor. -->
    <prop key="session.client">true</prop>
    <!-- haven't tried threaded=true -->
    <prop key="session.threaded">false</prop>
   </props>
  </property>
 </bean>

 <!-- Change your.Application to your Application class -->
    <bean id='application'
        class='your.Application'>
    </bean>

    <bean id='sessionSettings' class='quickfix.SessionSettings'>

<constructor-arg><value>${quickfix.config}</value></constructor-arg>
    </bean>

    <bean id='quickfix' class='sample.QuickFixBean'>
        <property name='application'><ref bean = 'application'/></property>
        <property name='sessionSettings'><ref
bean='sessionSettings'/></property>
        <property name='client'><value>${session.client}</value></property>
        <property
name='threaded'><value>${session.threaded}</value></property>
        <property name='logFactory'><ref bean='logFactory'/></property>
        <property name='messageStoreFactory'><ref
bean='messageStoreFactory'/></property>
    </bean>

    <bean id='logFactory' class='quickfix.SLF4JLogFactory'>
        <constructor-arg><ref bean='sessionSettings'/></constructor-arg>
    </bean>

    <bean id='messageStoreFactory' class='quickfix.FileStoreFactory'>
        <constructor-arg><ref bean='sessionSettings'/></constructor-arg>
```

```
    </bean>
</beans>
```

We also add in the following to get some basic JMX admin going. It requires an MBean server to be already be running. An easy way to do this is to use java5 and add -Dcom.sun.management.jmxremote to your startup parameters. You can then use jconsole ($JAVA_HOME/bin) to view your MBean.

```
<!-- Poor man's session management bean -->
    <bean id="sessionExporter"
class="org.springframework.jmx.export.MBeanExporter">
        <property name="beans">
            <bean id="quickfix.sessions"
class="org.springframework.beans.factory.config.PropertyPathFactoryBean"/>
        </property>
        <property name="assembler">
          <bean
class="org.springframework.jmx.export.assembler.MethodNameBasedMBeanInfoAs
sembler">
            <property name="managedMethods">

<value>isEnabled,getExpectedSenderNum,getExpectedTargetNum,isLoggedOn,isSe
ssionTime,logon,logout,

disconnect,enable,sentLogon,receivedLogon,sentLogout</value>
            </property>
          </bean>
        </property>
    </bean>
```

sample.QuickFixBean

```
package sample;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.BeanCreationException;
import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.util.StringUtils;

import quickfix.Acceptor;
import quickfix.Application;
import quickfix.ConfigError;
```

```
import quickfix.DefaultSessionFactory;
import quickfix.Initiator;
import quickfix.MemoryStoreFactory;
import quickfix.MessageStoreFactory;
import quickfix.SLF4JLogFactory;
import quickfix.Session;
import quickfix.SessionFactory;
import quickfix.SessionID;
import quickfix.SessionSettings;
import quickfix.SocketAcceptor;
import quickfix.SocketInitiator;
import quickfix.ThreadedSocketAcceptor;
import quickfix.ThreadedSocketInitiator;

/**
 * The QuickFixBean gives us a convenient way to boostrap quickfix from
Spring.
 */
public class QuickFixBean implements DisposableBean, InitializingBean {

    private Application application;

    private SessionSettings sessionSettings;

    private MessageStoreFactory messageStoreFactory;

    private quickfix.LogFactory logFactory;

    private boolean client = true;

    private boolean threaded = false;

    private Map sessions;

    private HashMap jmxSessions;

    private static Log log = LogFactory.getLog(QuickFixBean.class);

    public void afterPropertiesSet() throws ConfigError {
        validateProperties();
        SessionFactory sessionFactory = new
DefaultSessionFactory(application, messageStoreFactory, logFactory);
        if (client) {
            Initiator initiator;
            if (threaded) {
                initiator = new ThreadedSocketInitiator(sessionFactory,
sessionSettings);

            } else {
                initiator = new SocketInitiator(sessionFactory,
sessionSettings);
            }
            initiator.start();
```

```
        } else {
            Acceptor acceptor;
            if (threaded) {
                acceptor = new ThreadedSocketAcceptor(sessionFactory,
sessionSettings);

            } else {
                acceptor = new SocketAcceptor(sessionFactory,
sessionSettings);
            }
            acceptor.start();
        }
        lookupSessions();
        visitSessions(new SessionVisitor() {

            public void visitSession(SessionID sessionId, Session session)
{
                log.info("Started " + sessionId);

            }});
    }

    private void validateProperties() {
        if (application == null) {
            throw new BeanCreationException("Must specify an application");
        }
        if (sessionSettings == null) {
            throw new BeanCreationException("Must specify session
settings");
        }
        if (logFactory == null) {
            logFactory = new SLF4JLogFactory(sessionSettings);
        }
        if (messageStoreFactory == null) {
            messageStoreFactory = new MemoryStoreFactory();
        }
    }

    public void destroy()  {

        // ask them all to logout
        visitSessions(new SessionVisitor() {

            public void visitSession(SessionID sessionId, Session session)
{
                session.logout();
            }});

        // wait for them to do so
        visitSessions(new SessionVisitor() {

            public void visitSession(SessionID sessionId, Session session)
{
```

```
                // bit of a hack...
                int waitCount = 1;
                while (waitCount < 5 && session.isLoggedOn()) {
                    log.info("Waiting for " + sessionId + " to logout");
                    try {
                        Thread.sleep(200 * waitCount);
                    } catch (InterruptedException e) {
                        break;
                    }
                    waitCount++;
                }
                if (session.isLoggedOn()) {
                    log.warn(sessionId + " didn't log out, shutting down
anyway.");
                }
            }});
    }

    // at the time of writing acceptor.getSessions() didn't work...
    private void lookupSessions() {
        sessions = new HashMap();
        jmxSessions = new HashMap();
        for (Iterator iter = sessionSettings.sectionIterator();
iter.hasNext();) {
            SessionID sessionId = (SessionID) iter.next();
            Session session = Session.lookupSession(sessionId);
            if (session != null) {
                sessions.put(sessionId, session);
                // can't have : in the name but the sessionId looks like
FIX4.4:LocalCompId->RemoteCompId
                // prefix should be configurable
                jmxSessions.put("quickfixj:name=" +
StringUtils.replace(sessionId.toString(), ":", ""), session);
            }
        }
    }


    /**
     * This is just to avoid duplicating the loop code.
     * No it isn't really the Visitor pattern.  No it probably isn't
necessary.
     * @param sessionVisitor Does something to each session.
     * @see SessionVisitor#visitSession(SessionID, Session)
     */
    private void visitSessions(SessionVisitor sessionVisitor) {
        for (Iterator iter = sessions.entrySet().iterator();
iter.hasNext();) {
            Entry entry = (Entry) iter.next();
            SessionID sessionId = (SessionID) entry.getKey();
            Session session = (Session) entry.getValue();
            if (sessionId != null && session != null) {
                sessionVisitor.visitSession(sessionId, session);
```

```
            }
        }
    }

    private interface SessionVisitor {
        /**
         * Do something to the given session.
         * @param sessionId
         * @param session
         */
        public void visitSession(SessionID sessionId, Session session);
    }

    public Map getSessions() {
        return jmxSessions;
    }

    public Application getApplication() {
        return application;
    }

    public void setApplication(Application application) {
        this.application = application;
    }

    public boolean isClient() {
        return client;
    }

    public void setClient(boolean client) {
        this.client = client;
    }

    public quickfix.LogFactory getLogFactory() {
        return logFactory;
    }

    public void setLogFactory(quickfix.LogFactory logFactory) {
        this.logFactory = logFactory;
    }

    public MessageStoreFactory getMessageStoreFactory() {
        return messageStoreFactory;
    }

    public void setMessageStoreFactory(MessageStoreFactory
messageStoreFactory) {
        this.messageStoreFactory = messageStoreFactory;
    }

    public SessionSettings getSessionSettings() {
        return sessionSettings;
    }
```

```
public void setSessionSettings(SessionSettings sessionSettings) {
    this.sessionSettings = sessionSettings;
}

public boolean isThreaded() {
    return threaded;
}

public void setThreaded(boolean threaded) {
    this.threaded = threaded;
}
```

```
}
```

Starting it up

Now you just need to load your application context - put quickfixContext.xml on your classpath and do something like this:

```
new ClassPathXmlApplicationContext("quickfixContext.xml");
```

Or, for extra kicks put it in a servlet using a spring ContextLoaderListener in web.xml.