Code Samples

Small Rules Engine

The use case for this is when you have many different inputs that are driving a decision. In this case, the decision was on what PDF forms to display to the end user. Since the form was dependant upon user entered data, I needed a way to evaluate the data, and also allow for future forms to be added easily. Also, the criteria for the form kept changing, so I needed some way to easily update that criteria.

Rule.java

```
import ParameterList;

public interface Rule
{
    public boolean executeRule(ParameterList parameters);
    public Rule or(Rule orRule);
    public Rule and(Rule andRule);
}
```

LogicalRule.java

```
public abstract class LogicalRule implements Rule{
   public Rule or(Rule orRule) {
        return new OrRule(this,orRule);
    }

    public Rule and(Rule andRule) {
        return new AndRule(this,andRule);
    }
}
```

AndRule.java

```
public class AndRule extends LogicalRule {
    private Rule leftHand;
    private Rule rightHand;

    public AndRule(Rule leftHand, Rule rightHand) {
        this.leftHand = leftHand;
        this.rightHand = rightHand;
    }

    public boolean executeRule(ParameterList parameters) {
        return leftHand.executeRule(parameters) && rightHand.executeRule(parameters);
    }
}
```

Form.java

```
public class Form
{
    private final String name;
    private final String descriptiveName;
    private Rule formRule = new AlwaysTrueRule();

    public Form(String descriptiveName, String name)
    {
        this.name = name;
        this.descriptiveName = descriptiveName;
    }

    public void addRule(Rule rule)
    {
        formRule = rule;
    }
    public boolean isApplicable(ParameterList parameters)
    {
     return processRules(parameters);
    }
}
```

FormFactory.java

```
public class FormFactory
{
     protected static Logger logger = Logger.getLogger(FormFactory.class);

    private static List forms = new ArrayList();

     //Static block to share Immutable list with all factories.
    static
    {
        // Application Forms
        Form form = new Form(Form.APPLICATION_FORM_STR, Form.APPLICATION_FORM);
        form.addRule(new FalseRule(new IssueStateRule("MN")
                                   .or(new IssueStateRule("WY"))
                                        .or(new IssueStateRule("SD"))
                                        .or(new IssueStateRule("WI"))));
        forms.add(form);

        Form mnForm = new Form(Form.APPLICATION_FORM_STR, Form.APPLICATION_FORM_MN);
        mnForm.addRule(new IssueStateRule("MN"));
        forms.add(mnForm);

        //Etc Etc Etc for each form
    }
    /**
       * This is called from the service with the passed in model params.
      *//
    public ArrayList getApplicableForms(ParameterList parameters)
    {
        ArrayList applicableForms = new ArrayList();
        for (Iterator iterator = forms.iterator(); iterator.hasNext();)
        {
            Form form = (Form) iterator.next();
            if(form.isApplicable(parameters))
            {
                applicableForms.add(form);
```

```
            }
        }
        return applicableForms;
    }
```

NCAA Bracket Randomizer

This is a small app to randomly assign a person four teams in the NCAA Basketball tourney. It makes sure that each person receives one team in each of the four divisions. In addition it assures that the person gets one team that is ranked 1-4, one that is ranked 5-8, one that is ranked 9-12, and one that is ranked 13-16. Full program is a small Swing app that takes in the People's name and the Bracket.

BracketGUI.java

```
   public class BracketGUI {
private JTextPane seedText;
    private JButton randomlyGenerateBracketAssignmentsButton;
    private JTextField peopleTextField;
    private JTextField bracketsTextField;
    private JButton browseButton;
    private JButton browseButton1;
    private JPanel BracketCreator;
    private JTabbedPane tabbedPane1;
    private JTextPane nameText;
    private JFileChooser fc;
    public BracketGUI() {

        browseButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                //Create a file chooser

                int returnVal = fc.showOpenDialog(seedText);

                if (returnVal == JFileChooser.APPROVE_OPTION) {
                    File file = fc.getSelectedFile();
                    peopleTextField.setText(file.getAbsolutePath());
                }
            }
        });
        browseButton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                int returnVal = fc.showOpenDialog(seedText);

                if (returnVal == JFileChooser.APPROVE_OPTION) {
                    File file = fc.getSelectedFile();
                    bracketsTextField.setText(file.getAbsolutePath());
                }
            }
        });
        randomlyGenerateBracketAssignmentsButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                BracketAssigner ba = new BracketAssigner();
                Loader loader = new Loader();
                Printer printer = new Printer();
                //Read in list of people
                System.out.println("Loading People");
                ArrayList<Person> people = loader.loadPeople(peopleTextField.getText());
                System.out.println("Loading Teams");
                Team[][] teams = loader.loadTeams(bracketsTextField.getText());
                System.out.println("Assigning People to Brackets");
                Person[][][] assignedPeople = ba.assignPeopleToBracket(people);
                seedText.setText(printer.print(teams, assignedPeople,Printer.SORT_BY_SEED));
                nameText.setText(printer.print(teams, assignedPeople,Printer.SORT_BY_NAME));
            }
        });
    }


}
```

BracketAssigner.java

```
public class BracketAssigner {


    public Person[][][] assignPeopleToBracket(ArrayList<Person> people) {
        Person[][][] brackets = new Person[4][4][4];
        Random rng = new Random();
        for (int i = 0; i < people.size(); i++) {
            Person person = people.get(i);
            for(int quad = 0; quad < 4; quad++)
            {
                int subQuad = rng.nextInt(4);
                System.out.println("Rolled a " + subQuad + " for the subQuad");
                if(personIsDeadlocked(person,brackets[quad]))
                {
                    resetBracketForPerson(person,brackets);
                    quad = -1;
                    continue;
                }
                while(!roomInSelectedSubQuad(person,brackets[quad][subQuad],subQuad))
                {
                    subQuad = rng.nextInt(4);
                }
                int seedInSubQuad = rng.nextInt(4);
                System.out.println("Rolled a " + seedInSubQuad + " for the seed");
                while(brackets[quad][subQuad][seedInSubQuad] != null)
                {
                    System.out.println("\tSeed [" + seedInSubQuad +"] not available, rerolling");
                    seedInSubQuad = rng.nextInt(4);
                }
                System.out.println("\nAssigning " + person.getName() + " to [" + quad + "][" + subQuad + "]["
+ seedInSubQuad +"]\n");
                brackets[quad][subQuad][seedInSubQuad] = person;
                person.getSubquadMembership()[subQuad]=true;
                person.getSlot()[quad] = new Person.Triple(quad,subQuad,seedInSubQuad);
            }
        }
        return brackets;
    }

    private void resetBracketForPerson(Person person, Person[][][] brackets) {
        for(int i = 0; i < person.getSlot().length; i++)
        {
            Person.Triple slot = person.getSlot()[i];
            if(slot != null)
                brackets[slot.getRegion()][slot.getSubRegion()][slot.getSeed()] = null;
        }
        person.reset();
    }

    private boolean personIsDeadlocked(Person person,Person[][] subQuadBracket) {
        boolean deadlocked = true;

        for(int i =0; i < 4; i++)
        {
            List<Person> subQuad = Arrays.asList(subQuadBracket[i]);
            if(!person.getSubquadMembership()[i] && subQuad.contains(null))
            {
                deadlocked = false;
                break;
            }
        }
        return deadlocked;
    }

    private boolean roomInSelectedSubQuad(Person person,Person[] subQuadBracket,int subQuadint) {
        List<Person> subQuad = Arrays.asList(subQuadBracket);
        boolean available = false;
        if(subQuad.contains(null) && !person.getSubquadMembership()[subQuadint])
        {
            System.out.println("\tSubQuad [" + subQuadint + "] is available!");
            available = true;
        }
        else
        {
            System.out.println("\tThis SubQuad [" + subQuadint +"] is full");
```

```
        }
        return available;
    }
}
```

JSTL Taglib

This small tag library is used to get the version number out of the Manifest.MF file in a web app. The version number is set by the build system at build time.

```
public class VersionTag extends SimpleTagSupport {

public void doTag() throws JspException, IOException {
     super.doTag();

     PageContext pctx = (PageContext)getJspContext();
     Properties props = new Properties();
     props.load(pctx.getServletContext().getResourceAsStream("/META-INF/MANIFEST.MF"));

     Set entries = props.entrySet();
     pctx.getOut().print(props.getProperty("Specification-Version"));
}
}
```

Glassfish add ons

These classes were used to enhance the Glassfish Application Server. The first is used to configure JNDI URL references for configuration files. The second is used to achieve a Single Sign On solution with an IBM Access Management system.

URLResourceFactory.java

```
public class URLResourceFactory implements ObjectFactory {

        /**
         * @see javax.naming.spi.ObjectFactory#getObjectInstance(java.lang.Object,
         *      javax.naming.Name, javax.naming.Context, java.util.Hashtable)
         */
        public Object getObjectInstance(Object obj, Name name, Context nameCtx, Hashtable<?, ?> environment)
throws Exception {
                Reference ref = (Reference) obj;
                RefAddr addr = ref.get("url");
                if (addr == null) {
                        throw new NamingException("Missing parameter with key 'url'");
                }

                String urlName = (String) addr.getContent();
                if(urlName == null)
                {
                        throw new NamingException("Url attrbute not set.");
                }
                return new URL(urlName);
        }
}
```

WebsealAuthModule.java

```
public class WebsealAuthModule implements ServerAuthModule {

     private static final String AUTHORIZED_GROUPS = "iv-groups";
     private static final String AUTHORIZED_USER = "iv-user";
     private static final String DEBUG_OPTIONS_KEY = "debug";
     private static final String GROUP_PROPERTY_NAME = "defaultGroup";

     private boolean debug;

     private Level debugLevel;

     private CallbackHandler handler;

     private static final Logger LOG = Logger.getLogger(WebsealAuthModule.class.getName());
```

```
    private static Class[] supportedMessageTypes = new Class[] {
javax.servlet.http.HttpServletRequest.class, javax.servlet.http.HttpServletResponse.class };

    private MessagePolicy requestPolicy;

    private String defaultGroup[] = null;
/**
     * Initialize this module with request and response message policies to
     * enforce, a CallbackHandler, and any module-specific configuration
     * properties.
     * <p/>
     * <p>
     * The request policy and the response policy must not both be null.
     *
     * @param requestPolicy
     *            The request policy this module must enforce, or null.
     * @param responsePolicy
     *            The response policy this module must enforce, or null.
     * @param handler
     *            CallbackHandler used to request information.
     * @param options
     *            A Map of module-specific configuration properties. throws
     *            AuthException If module initialization fails, including for
     *            the case where the options argument contains elements that are
     *            not supported by the module.
     */
    public void initialize(MessagePolicy requestPolicy, MessagePolicy responsePolicy, CallbackHandler
handler, Map options) throws AuthException {
            this.requestPolicy = requestPolicy;
            this.handler = handler;
            if (options != null) {
                    if (options.containsKey(DEBUG_OPTIONS_KEY)) {
                            try {
                                    debug = Boolean.valueOf((String) options.get(DEBUG_OPTIONS_KEY));
                            } catch (Exception e) {
                                    debug = false;
                            }
                    } else {
                            debug = false;
                    }

                    if(options.containsKey(GROUP_PROPERTY_NAME))
                    {
                            defaultGroup = new String[]{(String)options.get(GROUP_PROPERTY_NAME)};
                    }
            }

            debugLevel = (LOG.isLoggable(Level.FINE) && !debug) ? Level.FINE : Level.INFO;
    }
    public AuthStatus validateRequest(MessageInfo messageInfo, Subject clientSubject, Subject
serviceSubject) throws AuthException {
            HttpServletRequest request = (HttpServletRequest) messageInfo.getRequestMessage();
            HttpServletResponse response = (HttpServletResponse) messageInfo.getResponseMessage();
            debugRequest(request);
            if (debug && clientSubject != null) {
                    LOG.log(Level.FINE, "ClientSubject = " + clientSubject);
            }
            try {
                    if (requestPolicy.isMandatory()) {
                            if (debug) {
                                    LOG.log(Level.FINE, "REQUEST IS MANDATORY");
                            }
                            String groups = request.getHeader(AUTHORIZED_GROUPS);
                            if (groups != null || defaultGroup != null) {
                                    String groupArray[] = getGroups(groups);

                                    String user = request.getHeader(AUTHORIZED_USER);
                                    // create Subject with principals from name
                                    CallerPrincipalCallback callerPrincipalCallback = new
CallerPrincipalCallback(clientSubject, user);

                                    GroupPrincipalCallback groupPrincipalCallback = new
GroupPrincipalCallback(clientSubject, groupArray);
                                    handler.handle(new Callback[] { callerPrincipalCallback,
groupPrincipalCallback });
```

```
                                if (debug && clientSubject != null) {
                                        LOG.log(Level.INFO, "ClientSubject = " + clientSubject);
                                }
                        } else {
                                response.setStatus(HttpServletResponse.SC_FORBIDDEN);
                                response.sendError(HttpServletResponse.SC_FORBIDDEN, "Resource requested
needs authentication to access.");
                                return AuthStatus.SEND_FAILURE;
                        }
                }
            } catch (IOException e) {
                    AuthException ae = new AuthException();
                    ae.initCause(e);
                    throw ae;

            } catch (UnsupportedCallbackException e) {
                    AuthException ae = new AuthException();
                    ae.initCause(e);
                    throw ae;

            }
            return AuthStatus.SUCCESS;
    }

    protected String[] getGroups(String groups) {
            ArrayList<String> groupList = new ArrayList<String>();
            if(groups != null && groups.trim().length() > 0)
            {
                    String groupArray[] = groups.split(",");
                    for (int i = 0; i < groupArray.length; i++) {
                            groupList.add(groupArray[i].substring(1, groupArray[i].length() - 1));
                    }
            }
            if(defaultGroup != null)
            {
                    groupList.add(defaultGroup[0]);
            }
            return groupList.toArray(new String[]{});
    }

    private void debugRequest(HttpServletRequest request) {

            if (debug || LOG.isLoggable(Level.FINE)) {
                    StringBuffer sb = new StringBuffer();
                    sb.append("\n");
                    try {
                            sb.append("Request: " + request.getRequestURL() + "\n");
                            sb.append("UserPrincipal: " + request.getUserPrincipal() + "\n");
                            sb.append("AuthType: " + request.getAuthType() + "\n");
                            sb.append("Headers:" + "\n");
                            Enumeration names = request.getHeaderNames();
                            while (names.hasMoreElements()) {
                                    String name = (String) names.nextElement();
                                    sb.append("\t" + name + "\t" + request.getHeader(name) + "\n");
                            }
                            LOG.log(debugLevel, sb.toString());

                    } catch (Throwable t) {
                            LOG.log(Level.WARNING, "jmac.servlet_debug_request", t);
                    }
            }
    }
```

LDAP Querying

Given a set of input, create the proper LDAP query message and return the results. This uses Spring Ldap. Also shows an example of using a parameter object to encapsulate the input.

LdapDao.java

```
public interface LdapDao {
    public List getAllNames();
    public List getPersons(LdapQuery query);
    public List getAllGroups();
}
```

LdapDaoImp.java

```
public class LdapDaoImp implements LdapDao {
    private LdapTemplate ldapTemplate;
    private LdapTemplate prodLdapTemplate;

    public void setLdapTemplate(LdapTemplate ldapTemplate) {
           this.ldapTemplate = ldapTemplate;
    }

    public void setProdLdapTemplate(LdapTemplate prodLdapTemplate) {
           this.prodLdapTemplate = prodLdapTemplate;
    }

    public List getAllNames() {
           EqualsFilter filter = new EqualsFilter("objectclass", "person");
           return ldapTemplate.search(DistinguishedName.EMPTY_PATH, filter
                         .encode(), new PersonAttributesMapper());
    }

    public List getPersons(LdapQuery query) {
           AndFilter filter = new AndFilter();
           filter.and(new EqualsFilter("objectclass", "person"));
           if (StringUtils.isNotEmpty(query.getLastName())) {
                  filter.and(new WhitespaceWildcardsFilter("sn", query.getLastName()));
           }
           if (StringUtils.isNotEmpty(query.getFirstName()))
           {
                  filter.and(new EqualsFilter("givenname",query.getFirstName()));
           }
           if (StringUtils.isNotEmpty(query.getUserID()))
           {
                  filter.and(new EqualsFilter("uid",query.getUserID()));
           }
           if (StringUtils.isNotEmpty(query.getBudgetCenter()))
           {
                  filter.and(new EqualsFilter("fblbudgetcenter",query.getBudgetCenter()));
           }
           if (StringUtils.isNotEmpty(query.getEmployeeNum()))
           {
                  filter.and(new EqualsFilter("employeenumber",query.getEmployeeNum()));
           }
           if(query.isUseProd())
           {
                  return prodLdapTemplate.search("", filter.encode(),new PersonAttributesMapper());
           }
           else
           {
                  return ldapTemplate.search("", filter.encode(),new PersonAttributesMapper());
           }
    }

    public List getAllGroups() {
           EqualsFilter filter = new EqualsFilter("objectclass", "group");
           return ldapTemplate.search(DistinguishedName.EMPTY_PATH, filter
                         .encode(), new PersonAttributesMapper());
    }

    private class PersonAttributesMapper implements AttributesMapper {
           public Object mapFromAttributes(Attributes attrs)
                         throws NamingException {
                  Person person = new Person();
                  person.setFullName(getAttribute(attrs, "cn"));
                  person.setFirstName(getAttribute(attrs, "givenname"));
                  person.setLastName(getAttribute(attrs, "sn"));
                  person.setMiddleInitial(getAttribute(attrs, "initials"));
                  person.setPhoneNum(getAttribute(attrs, "telephonenumber"));
                  person.setDepartment(getAttribute(attrs, "departmentnumber"));
                  person.setEmployeeType(getAttribute(attrs, "employeetype"));
                  person.setLocation(getAttribute(attrs, "l"));
                  person.setEmail(getAttribute(attrs, "mail"));
                  person.setUserID(getAttribute(attrs, "uid"));
                  person.setState(getAttribute(attrs, "st"));
                  person.setTitle(getAttribute(attrs, "title"));
                  person.setEmployeeID(getAttribute(attrs,"employeenumber"));
                  person.setManager(getAttribute(attrs,"manager",1));
                  person.setSupervisorIndicator(getAttribute(attrs,"supervisorindicator"));
```

```
                person.setBudgetCenter(getAttribute(attrs,"fblbudgetcenter"));
                return person;
        }

        private String getAttribute(Attributes attrs, String attribute) throws NamingException {
                return getAttribute(attrs,attribute,0);
        }

        private String getAttribute(Attributes attrs, String attribute,int whichValue) throws
NamingException
        {
                Attribute attr = attrs.get(attribute);
                String stringVal = "";
                if (attr != null) {
                        stringVal = (String) attr.get(whichValue);
                }
                return stringVal;
        }
    }
}
```

Webservice with Metro

This is an implementation of a service which calls an Imaging system to get a document, and return the document or document meta-data to the caller.

```
@MTOM
@WebService(serviceName="RetrieveDocument", portName="RetrieveDocumentPort",
targetNamespace="http://www.fbfs.com/RetrieveDocument")

public class RetrieveDocumentPortTypeImpl {
Logger logger = Logger.getLogger("RetrieveDocumentPortTypeImpl");


    public DataHandler retrieveDocument(@WebParam(name="documentID")long docID) throws InvalidDataException,
SystemUnavailableException
    {

        StopWatch watch = new StopWatch();
        watch.start();
        FileNetConnector connector = new FileNetConnector();
        try {
                DocumentInfo docInfo = connector.getDocumentInfo(docID);
                if(docInfo == null)
                {
                        throw new InvalidDataException("Error - No results found.");
                }
                ByteArrayOutputStream baos = connector.getDocument(docInfo);
                ByteArrayDataSource ds = new ByteArrayDataSource(baos.toByteArray(), "application/pdf");
                return new DataHandler(ds);
        }
        finally
        {
                watch.stop();
                logger.log(Level.INFO,"retrieveDocument took ["+watch.toString()+"]");
        }
    }
    @WebResult(name="documents")
    public Documents search(@WebParam(name="policyNumber")long policyNumber) throws
InvalidDataException,SystemUnavailableException
    {

        StopWatch watch = new StopWatch();
        watch.start();
        FileNetConnector connector = new FileNetConnector();
        Documents docs = new Documents();
        try {
                docs.setDocInfo(connector.getDocumentsForPolicy(policyNumber));
        }
        finally
        {
                watch.stop();
                logger.log(Level.INFO,"search took ["+watch.toString()+"]");
        }
        return docs;
    }
```

```
}
```

PDF Form Binding Engine

I extended the Spring Frameworks Bean classes to implement a framework that takes model data, and binds it to a PDF form field. This allowed me to rapidly respond to changes that were required by the business and by regulators. Since forms are always changing, this made sense to make the binding a configuration.

AbstractPdfBinder.java

```
public abstract class AbstractPdfBinder implements PropertyEditorRegistry{


    private SimpleTypeConverter typeConverter;
    private PropertyEditorRegistrar customPropertyEditorRegistrar;
    private Resource pdfFormFolderLocation;
    private String formName;
    private Map fields;


    public PropertyEditor findCustomEditor(Class requiredType, String propertyPath) {
            return getPropertyEditorRegistry().findCustomEditor(requiredType, propertyPath);
    }

    public void registerCustomEditor(Class requiredType, PropertyEditor propertyEditor) {
            getPropertyEditorRegistry().registerCustomEditor(requiredType, propertyEditor);
    }

    public void registerCustomEditor(Class requiredType, String field, PropertyEditor propertyEditor) {
            getPropertyEditorRegistry().registerCustomEditor(requiredType, field, propertyEditor);
    }
    /**
     * Return this binder's underlying SimpleTypeConverter.
     */
    protected SimpleTypeConverter getSimpleTypeConverter() {
            if (this.typeConverter == null) {
                    this.typeConverter = new SimpleTypeConverter();
            }
            return this.typeConverter;
    }
    private PropertyEditorRegistry getPropertyEditorRegistry() {
            return getSimpleTypeConverter();
    }
    public void setCustomPropertyEditorRegistrar(PropertyEditorRegistrar customPropertyEditorRegistrar) {
            this.customPropertyEditorRegistrar = customPropertyEditorRegistrar;
    }

    public PropertyEditorRegistrar getCustomPropertyEditorRegistrar() {
            return customPropertyEditorRegistrar;
    }

    public PdfBindingResult bind(Object model, OutputStream outStream) throws MalformedURLException,
IOException, DocumentException
    {

            initBinder(this);
            PdfBindingResult bindingResults = getModel(model);
            doBind(bindingResults,outStream);
            return bindingResults;

    }

    protected void initBinder(AbstractPdfBinder binder) {
            getCustomPropertyEditorRegistrar().registerCustomEditors(binder);
    }
    /**
     * @return Returns the pdfFormFolderLocation.
     */
    public Resource getPdfFormFolderLocation() {
            return pdfFormFolderLocation;
    }
    /**
     * @param pdfFormFolderLocation The pdfFormFolderLocation to set.
```

```
     */
    public void setPdfFormFolderLocation(Resource pdfFormFolderLocation) {
            this.pdfFormFolderLocation = pdfFormFolderLocation;
    }

    /**
     * @return Returns the formName.
     */
    public String getFormName() {
            return formName;
    }
    /**
     * @param formName The formName to set.
     */
    public void setFormName(String formName) {
            this.formName = formName;
    }

    /**
     * @return
     */
    protected File getForm(){
            String formName;
            try {
                    formName = getPdfFormFolderLocation().getFile().getAbsolutePath() + File.separator
+getFormName();
            } catch (IOException e) {
                    throw new FormNotFoundException("Form wasn't found.",e);
            }
            return new File(formName);
    }
    /**
     * @return Returns the fields.
     */
    public Map getFields() {
            return fields;
    }
    /**
     * @param fields The fields to set.
     */
    public void setFields(Map fields) {
            this.fields = fields;
    }

    protected String getFieldAsString(Object fieldValue, Class fieldClass, String fieldPath) {
            if(fieldValue == null)
            {
                    return "";
            }

            PropertyEditor editor = getSimpleTypeConverter().findCustomEditor(fieldClass, fieldPath);
            if(editor == null)
            {
                    editor = getSimpleTypeConverter().getDefaultEditor(fieldClass);
            }

            if(editor == null && (String.class.equals(fieldClass) ||
ClassUtils.isPrimitiveOrWrapper(fieldClass)))
            {
                            // We can stringify any primitive value...
                            return fieldValue.toString();
            }

            editor.setValue(fieldValue);
            return editor.getAsText();

    }
    public abstract void doBind(PdfBindingResult bindingResults, OutputStream outstream) throws IOException,
DocumentException;
    public abstract PdfBindingResult getModel(Object model);
}
```

SimplePdfBinder.java

```
public class SimplePdfBinder extends AbstractPdfBinder{
    public void doBind(PdfBindingResult bindingResults, OutputStream outstream) throws DocumentException,
IOException
```

```
    {
            ConfigurablePropertyAccessor wrapper = bindingResults.getPropertyAccessor();
            File theForm = getForm();
            PdfReader reader = new PdfReader(theForm.getPath());
            PdfStamper stamper = new PdfStamper(reader,outstream);
            AcroFields formFields = stamper.getAcroFields();
            for(Iterator iterator = getFields().entrySet().iterator();iterator.hasNext();)
            {
                    Map.Entry fieldEntry = (Map.Entry)iterator.next();
                    String fieldPath = (String)fieldEntry.getValue();
                    Object fieldValue = wrapper.getPropertyValue(fieldPath);
                    Class fieldClass = wrapper.getPropertyType(fieldPath);
                    int fieldType = formFields.getFieldType((String)fieldEntry.getKey());
                    String fieldName = (String)fieldEntry.getKey();
                    if(fieldType == AcroFields.FIELD_TYPE_TEXT ||fieldType ==
AcroFields.FIELD_TYPE_RADIOBUTTON || fieldType == AcroFields.FIELD_TYPE_CHECKBOX)
                    {
                            if(fieldType == AcroFields.FIELD_TYPE_CHECKBOX)
                            {
                                    registerCustomEditor(Boolean.class,fieldPath,new CustomBooleanEditor("On",
"Off", true));
                            }
                            String fieldStringValue = getFieldAsString(fieldValue,fieldClass,fieldPath);

    validateFieldValue(formFields,fieldStringValue,fieldName,fieldType,fieldPath,bindingResults);
                            formFields.setField(fieldName,fieldStringValue);
                    }
                    else
                    {
                            bindingResults.rejectValue(fieldPath, "unsupported.fieldType");
                    }

            }
            stamper.setFormFlattening(true);

            stamper.close();
            reader.close();

    }

    protected void validateFieldValue(AcroFields formFields, String fieldStringValue, String fieldName, int
fieldType,String fieldPath,PdfBindingResult bindingResults) {
            if(fieldType == AcroFields.FIELD_TYPE_RADIOBUTTON)
            {
                    String[] validValues = formFields.getAppearanceStates(fieldName);
                    List listValues = Arrays.asList(validValues);
                    if(!listValues.contains(fieldStringValue))
                    {
                            bindingResults.rejectValue(fieldPath, "invalid.value");
                    }
            }
    }
    public PdfBindingResult getModel(Object model) {
            return new PdfBindingResult(model,"model");
    }
}
```

PdfPropertyEditorRegistrar.java

```
public class PdfPropertyEditorRegistrar implements PropertyEditorRegistrar {

    public void registerCustomEditors(PropertyEditorRegistry registry) {
            registry.registerCustomEditor(Date.class, new
CustomDateEditor(SimpleDateFormat.getDateInstance(SimpleDateFormat.LONG), true));

            // Form 432-129 - Section A
            registry.registerCustomEditor(Integer.class, "lifeApplicationUIModel.CTobaccoType", new
DomainsPropertyEditor(Domains.tobaccoUsage));
            registry.registerCustomEditor(Integer.class, "lifeApplicationUIModel.CTobaccoFrequency", new
DomainsPropertyEditor(Domains.tobaccoFrequency));
            registry.registerCustomEditor(Date.class, "insured.DBirth", new
CustomDateEditor(SimpleDateFormat.getDateInstance(SimpleDateFormat.SHORT), true));
            registry.registerCustomEditor(String.class, "insured.ISsnTin", new
CustomMaskPropertyEditor(CustomMaskPropertyEditor.SOC_NUM_FORMAT));
            registry.registerCustomEditor(Integer.class, "insuredAddress.CState", new
DomainsPropertyEditor(Domains.stateAbreviation));
```

```
            registry.registerCustomEditor(String.class, "insuredHomePhone.IPhoneNbr",new
CustomMaskPropertyEditor(CustomMaskPropertyEditor.PHONE_FORMAT));
            registry.registerCustomEditor(String.class, "insuredWorkPhone.IPhoneNbr",new
CustomMaskPropertyEditor(CustomMaskPropertyEditor.PHONE_FORMAT));
            registry.registerCustomEditor(String.class, "insuredCellPhone.IPhoneNbr",new
CustomMaskPropertyEditor(CustomMaskPropertyEditor.PHONE_FORMAT));
            registry.registerCustomEditor(BigDecimal.class,"insured.AGrossIncome",new
CustomNumberEditor(BigDecimal.class, NumberFormat.getCurrencyInstance(), false));

//Additional editors are registered as needed for formatting of model data
    }
}
```

Spring Configuration

We define our parent bean to extend from at first. Then each unique form gets defined, extending the parent bean. The fields are a key-value map with the key being the PDF form name, and the value being the bean path in the model to the data.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd"
xmlns:util="http://www.springframework.org/schema/util">
    <!--
            ********************************************************************************
            ************************** PDF Beans Configuration**********************
            ********************************************************************************
    -->
    <bean id="customPropertyEditorRegistrar" class="com.fbfs.lf.lec.pdf.PdfPropertyEditorRegistrar"/>

    <bean id="parentPDFBinder" abstract="true" class="com.fbfs.lf.lec.pdf.SimplePdfBinder">
            <property name="pdfFormFolderLocation" value="/WEB-INF/Forms"/>
            <property name="customPropertyEditorRegistrar" ref="customPropertyEditorRegistrar"/>
    </bean>
<!-- Summary of Coverage Home Office -->
 <bean id="form433-133HO" parent="parentPDFBinder">
    <property name="formName" value="433-133_11-09HO.pdf"/>
    <property name="fields">
            <map>
                    <entry key="IPreferWithdrawDay" value="lifeApplicationUIModel.IEftWithdrawDay"/>
                    <entry key="DDesiredIssue" value="lifeApplicationUIModel.DDesiredIssue"/>
                    <entry key="CTermPeriod" value="lifeApplicationUIModel.CTermPeriod"/>
                    <entry key="CPremMode" value="lifeApplicationUIModel.CPremMode"/>
                    <entry key="IPolicyNBR" value="lifeApplicationUIModel.IPolicyNbr"/>
                    <entry key="AModalPrem" value="lifeApplicationUIModel.AModalPrem"/>
            </map>
    </property>
</bean>
<!-- Generic Replacement Form -->
 <bean id="form434-176-A" parent="parentPDFBinder">
    <property name="formName" value="434-176A(07-00).pdf"/>
    <property name="fields">
            <map>
                    <entry key="ReplacePremiumRadio" value="replacement.CDiscontinuePayment"/>
                    <entry key="ReplacePolicyRadio" value="replacement.CUseExistingFunds"/>
                    <entry key="NReplacedCompany" value="replacedCompany"/>
                    <entry key="IReplacedPolNBR" value="replacement.IReplacedPolicyNbr"/>
                    <entry key="NLastInsuredNameReplacing" value="replacedInsuredName"/>
                    <entry key="CPolicyReplaceFinance" value="replacement.CPolicyReplaceFinance"/>
                    <entry key="NReasonforReplace" value="replacement.NReasonTerminated"/>
                    <entry key="NInsuredFirstMiddleLast" value="uiModel.insured.fullName"/>
                    <entry key="NAgentFirstMiddleLast" value="uiModel.agentSignatureUIModel.TSignature"/>
                    <entry key="TApplicantEsignature" value="uiModel.insuredSignatureUIModel.TSignature"/>
                    <entry key="CSignGUIDApplicant" value="uiModel.insuredSignatureUIModel.CGuid"/>
                    <entry key="TDateFieldApplicantEsign" value="uiModel.insuredSignatureUIModel.DSign"/>
                    <entry key="TAgentEsignature" value="uiModel.agentSignatureUIModel.TSignature"/>
                    <entry key="CSignGUIDAgent" value="uiModel.agentSignatureUIModel.CGuid"/>
                    <entry key="TDateFIeldAgentEsign" value="uiModel.agentSignatureUIModel.DSign"/>
                    <entry key="TInitialsApplicantEsign" value="readAloudInitials"/>
            </map>
    </property>
</bean>
<!—Additional forms added as needed →
</beans>
```