HDIV

This page is to discuss the possible options of integrating HDIV with Struts.

The original email chain can be found here.

Talking points

SPI or native integration?

Matt Raible: HDIV seems to solve a problem that most web application developers don't know they have. By "natively", I mean it's part of the core and you can't make your application less secure by ripping it out. It is Apache licensed after all. If rolling it into the core isn't an option, it would be nice if it was easier to integrate. Instead of requiring new tag libraries, it'd be nice if tag libraries (and Velocity/FreeMarker macros) were "HDIV aware". If an HDIV JAR /Plugin is on the classpath - use it.

Paul Benedict: I wouldn't want to fork the project because I am not a security expert. I couldn't maintain it well even though I want to integrate it. Also, I don't know if HDIV has aspirations outside of Struts which would make an SPI much more palatable. I am not strongly in favor of belonging to the core. I think the feature should be optional, but I wouldn't also object if it was put of the core with the option to turn on/off.

Martin Cooper: Which security package(s) would you want to work with in addition to HDIV? I firmly believe that you need at least two candidates in order to successfully design an SPI. Otherwise you run a very high risk of designing an SPI that can really only be successfully used by the one candidate you designed it around.

HDIV team: We think it's better to use a SPI. From our point of view the SPI it's an extension point that could be interesting for many projects, not only for security projects. For example if you use webflow in a Struts application you have to add flow execution key parameters by hand and that could be easily solved by implementing a SPI.

Of course, if we use the SPI it's not necessary to extend struts' tld making it easier to integrate HDIV.

In addition to that if you use the SPI you can activate or desactivate updating Struts configuration. By default Struts can use an empty implementation of the SPI.

HDIV can be very useful for many applications but not for others. For example if you have a public web page where it's necessary to be indexable, you shouldn't use HDIV because all links are dynamic and related with web session (the same problem of JSF but in this case you can desactivate it).

About Paul Benedict's comment ("I don't know if HDIV has aspirations outside of Struts which would make an SPI much more palatable"), the target of HDIV is the integration with all web frameworks that need this type of security functionalities. So far we have developed HDIV versions for:

- Struts 1
- Struts 2
- Spring MVC
- JSTL
- WebWork (not published)
- Stripes (not published)
- JSF (not published)

Some of them are still under development, but they will be published in a few months. In consecuence, it will be useful if the SPI it's generic and works with all web frameworks. But thinking about the implementation and library dependencies maybe it would be better if each framework had it's own interface, similar to interceptors concept in many frameworks but in this case related with tag libraries. Anyway, we could create a first release supporting commented frameworks.

Martin Cooper: Please re-read my comment. I am not arguing against an SPI per se. My point is that, without some other security framework in mind in addition to HDIV, it is very unlikely that you will be able to design an SPI that can successfully be used by any other security framework.

Also, given the list of frameworks that you already support, don't you already have an internal API, so that the bulk of your code is common for all frameworks, and you just have a small shim to specialise it for each framework? If that's the case, then how much value is there in having a shared SPI just within Struts? You really need all of the frameworks to implement that SPI, not just Struts, in order to get rid of your per-framework shims, in which case you really want to engage all of the frameworks in the SPI design, not just the Struts team.

HDIV team: From our point of view, although we find interesting to define a common interface for all the frameworks, our primary goal is to add new extension points to Struts' API so that Struts' users can use it in a transparent way, without having to copy any source code.

The real problem is similar to what happens in Struts 1 with the RequestProcessor: when you need to intercept a request's life cycle, many times you are forced to extend the RequestProcessor and copy parts of the existing code next to the extension. This problem was solved in Struts2, because each step of the life cycle is implemented by a different interceptor.

Actually, we want to add extension points to the HTML creation process.

When implementing HDIV for Struts, we had to extend tags and copy parts of the Struts' source code in order to fulfil HDIV's objectives. For us, these are the main reasons why an interface should be created:

1. Avoid having to extend tags and copy source code. Also avoids creating a new version for the extension for each Struts' version.
   2. Provide extension points that could be used by any framework.
   3. Eliminate the direct dependency between Struts and HDIV.

If we integrate HDIV natively in the tags, without using an interface, we eliminate an extension point that could be required by other frameworks or by Struts itself. Nowadays, we don´t know who and how will require this interface (the automatization of the webflow's flow execution key could be an example), but from a designer point of view it is always better to define an interface that could change from version to version.

There should be two different implementations of this interface: the one provided by HDIV and the one provided by Struts. Struts' implementation would keep the current tag behaviour and HDIV's one would add security functionalities.

In our opinion, HDIV's implementation should be configured by default in order to make the application safer. If the developers don't require HDIV they could modify the configuration and use the empty implementation provided by Struts.

So, although we find using an interface a better solution, we don´t have any problem if the final decision is to include it natively in the code.

Performance

Ted Husted: It's unusual that a feature such as this comes without penality. If HDIV were native, what would be the performance cost? Complexity cost?

HDIV team: In our opinion the performance offered by HDIV is acceptable and it could be activated by default, but we have to take into account that it generates a problem with the pages indexation (all links are dynamic and related with web session). Consequently we think it's better if the HDIV activation it's optional.

Another discussion is if it has to be activated by default or not. From a security point of view it's better if HDIV is activated by default. We agree with Matt Raible when he says that HDIV "solves a problem that most web application developers don't know they have". In our experience we have seen that even experimented developers don't know they have.

For more information about HDIV performance see hdiv-performance.pdf

Validation

Martin Cooper: How does HDIV's editable content validation interact with the validation mechanisms that we already have built into Struts? Is someone using HDIV with Struts going to be writing their validations using HDIV's format, Commons Validator's format, or both?

HDIV team: Editable data validation offered by HDIV it's integrated with struts' validator. HDIV creates validation errors within HDIV's validation filter and they are added within HDIV RequestProccessor. The errors generated by HDIV are generated with the same format of Struts and they are visualized in the same way as usual errors using Struts tags (html:errors,logic:messagesPresent, etc.).

For more information about it see HDIV reference (chapter-7.1.2.4.2).

Usability

Martin Cooper: How much of the functionality of HDIV is only available for people using JSP with tag libraries? If I'm using Velocity, or not using server-side presentation at all, how much of HDIV do I lose?

HDIV team: The core API of HDIV (hdiv-core) is not related with a concrete technology and it's possible to use it with any technology. Although we haven't implemented it for Struts 1 the integration of Velocity and Freemarker it's included in Struts 2 version, see Struts 2 example application, within (struts2-showcase-2.0.x) ui-tags section.

Martin Cooper: If I am not using any server-side presentation technology at all, which parts of HDIV do I lose? Which can I still use?

HDIV team: If no presentation framework is used (or the one used is not supported by HDIV), the only way to use HDIV is to consume the base API offered by HDIV core (hdiv-core). That is, the program which generates the HTML code should consume the methods offered by the IDataComposer interface.

Obviously, in this case the use of HDIV is not transparent as it is when we use the extended tags. Anyway, the validation phase made by the HDIV filter doesn't change in any case.

SPI (ParameterProcessing) Integration

HDIV team: As we mentioned earlier, from our point of view it is more adecuate to integrate HDIV with Struts via SPI due to the following reasons:

- We avoid a direct dependency between Struts and HDIV.
- Offers an extension point for Struts tags that can be used for other projects apart from the ones related with security or HDIV.

The new extension point provided by the SPI offers these functionalities:

- Add new parameters to forms and links
- Modify the value of the parameters
- Modify the name of the parameters

The SPI could be used for both Struts 1 and Struts 2. Being defined in a so high level it could be used for any web framework, although this would be a secondary objective.

In order to support the SPI option, we have developed the following software components:
1. SPI
2. Struts 1.3.8 tags' version using the SPI
3. Two implementations of the SPI: empty and HDIV's implementation
4. Web application using these new components: struts-examples

The SPI, the new version of SPI-consuming tags and the empty SPI implementation would be part of Struts, without any HDIV dependency. In the case of Struts 2, a new version of tags must be implemented. We could implement it if necessary. As an additional information, we describe more deeply each of the developed components:

1.SPI

It is consumed by the HTML tags in order to offer extension points and extend tag's core behaviour. Thus, we avoid extending each tag separately. The name of the SPI is ParameterProcessing and this is its definition:

```
package org.apache.struts.config;

import java.util.Map;
import javax.servlet.http.HttpServletRequest;

/**
 * <p>
 * It is consumed by the HTML tags in order to offer extension points and extend
 * tag's core behaviour.
 * </p>
 * <p>
 * Struts will distribute one implementation of this interface maintaining
 * present behaviour: org.apache.struts.config.impl.EmptyParameterProcessing. It
 * doesn't process any form or link parameter, leaving values as they are
 * received as parameters.
 * </p>
 * <p>
 * By default, Struts will be configured to use HDIV's implementation
 * (org.hdiv.config.impl.HDIVParameterProcessing) which guarantees data integrity
 * and confidentiality.
 * </p>
 *
 * @author hdiv.org
 */
public interface ParameterProcessing {

        /**
         * Init method.
         *
         * @param request The servlet request we are processing
         */
        public void init(HttpServletRequest request);

        /**
         * <p>
         * It is called by each form of the html page returned by the server. The
         * value returned by this method will be the value assigned to the form's
         * "action" attribute.
         * </p>
         *
         * @param action Action value.
         * @return Returns the value of form´s "action" attribute
         */
        public String processStartForm(String action);

        /**
         * <p>
         * It is called each time the "name" attribute of the form parameter needs
         * to be rendered. The value returned by this method will be the value
         * assigned to the form's "name" attribute.
         * </p>
         *
         * @param name HTTP parameter name
         * @param type parameter's type (select, radio, hidden, etc.). Most of the
         * times the value for this parameter is the value of the tag's "type"
         * attribute. Sometimes it is needed to explicitly indicate the type
         * parameter, because some HTML tags lack it.
         * @return Value for the "name" attribute for the <code>name</code>
         * parameter.
         */
        public String processFormParameterName(String name, String type);

/**
         * <p>
         * It is called each time the "value" attribute of the form parameter needs
         * to be rendered. The value returned by this method will be the value
         * assigned to the form's "value" attribute.
         * </p>
         * <p>
         * In the default implementation configured by Struts, it generates a new
         * encoded value for the parameter <code>name</code> and the value
         * <code>value</code> passed as parameters. The returned value guarantees
         * the confidentiality in the cipher and memory strategies if
         * confidentiality is activated.
         * </p>
         *
         * @param name HTTP parameter name
         * @param value parameter's value
         * @param type parameter's type (select, radio, hidden, etc.). Most of the
         * times the value for this parameter is the value of the tag's "type"
         * attribute. Sometimes it is needed to explicitly indicate the type
         * parameter, because some HTML tags lack it.
         * @return Value for the "value" attribute of the form's <code>name</code>
         * parameter
         */
        public String processFormParameterValue(String name, String value, String type);

        /**
         * <p>
         * It is invoked before the form is closed and it returns the extra
         * parameters, the ones not defined using HTML tags, that must be added to
         * the form. For instance, parameters for controlling the application's
         * flow, random token to avoid CSRF attacks, etc.
         * </p>
         * <p>
         * In the default implementation configured by Struts, the HDIV parameter
         * that guarantees data integrity and random token to avoid CSRF attacks are
         * returned. If our application uses Spring Web Flow, we can add the
         * <code>_flowExecutionKey</code> parameter to the forms and links
         * automatically by using the hdiv-webflow-x.x library. See HDIV's
         * repository: <a href="http://repo1.maven.org/maven2/org/hdiv/">HDIV
         * (Central Repository)</a>
         * </p>
         *
         * @return Returns parameters' map that we need to add to form as new
         * fields.
         */
        public Map getExtraFormParameters();

        /**
         * <p>
         * It is invoked each time the "href" attribute of a url generated by the
         * <code>link</code>, <code>frame</code>, <code>redirect</code> and
         * <code>forward</code> tags needs to be rendered.
         * </p>
         * <p>
         * In the default implementation configured by Struts, it generates new
         * encoded values for the <code>url</code> parameters only if
         * confidentiality is activated, and adds HDIV's parameter and a random
         * token parameter to avoid data tampering and CSRF attacks respectively.
         * </p>
         *
         * @param url request url
         * @param charEncoding character encoding
         * @return url Processed url
         */
        public String processURL(String url, String charEncoding);

}
```

Source code: struts-core-1.3.8-ParameterProcessing-sources

2.Struts 1.3.8 tags' version using the SPI

We have modified Struts 1.3.8 HTML and LOGIC (redirect and forward) tags by adding calls to the SPI methods. We have implemented tags so that each time the attributes "name" or "value" need to be rendered they call SPI methods which process these values and obtain the processed value. Let's see an example with the tag "hidden". Keep in mind that this tag extends BaseFieldTag class which renderizes the HTML of the hidden field.

```
/**
 * Custom tag for input fields of type "hidden".
 *
 * @version hdiv.org
 */
public class HiddenTag extends BaseFieldTag {

...
/**
 * @return Returns the value to be rendered in the “name” attribute.
 * @throws JspException if a JSP exception has occurred
 * @version hdiv.org
 */
    public String processName() throws JspException {
        return processFormParameterName(prepareName(), this.type);
    }

        /**
         * @return Returns the value to be rendered in the “value” attribute.
         * @throws JspException if a JSP exception has occurred
         * @version hdiv.org
         */
    public String processValue() throws JspException {
            return processFormParameterValue(prepareName(), prepareValue(), this.type);
    }

...
}
```

```
/**
 * Convenience base class for the various input tags for text fields.
 *
 * @version hdiv.org
 */
public abstract class BaseFieldTag extends BaseInputTag {

...

    /**
     * Renders a fully formed <input> element.
     * @throws JspException
     */
    protected String renderInputElement() throws JspException {
        StringBuffer results = new StringBuffer("<input");
        prepareAttribute(results, "type", this.type);

        prepareAttribute(results, "name", processName());  <---- new implementation

        prepareAttribute(results, "accesskey", getAccesskey());
        prepareAttribute(results, "accept", getAccept());
        prepareAttribute(results, "maxlength", getMaxlength());
        prepareAttribute(results, "size", getCols());
        prepareAttribute(results, "tabindex", getTabindex());

        prepareAttribute(results, "value", this.formatValue(processValue()));  <---- new implementation

        results.append(this.prepareEventHandlers());
        results.append(this.prepareStyles());
        prepareOtherAttributes(results);
        results.append(this.getElementClose());
        return results.toString();
    }
        /**
         * @return Returns the value to be rendered in the “value” attribute.
         * @throws JspException if a JSP exception has occurred
         * @version hdiv.org
         */
public abstract String processValue() throws JspException;
        /**
         * @return Returns the value to be rendered in the “name” attribute.
         * @throws JspException if a JSP exception has occurred
         * @version hdiv.org
         */
        public abstract String processName() throws JspException;
        ...
}
```

```
/**
 * Base class for tags that render form elements capable of including JavaScript
 * event handlers and/or CSS Style attributes. This class does not implement the
 * doStartTag() or doEndTag() methods. Subclasses should provide appropriate
 * implementations of these.
 *
 * @version hdiv.org
 */
public abstract class BaseHandlerTag extends BodyTagSupport {

...

public String processFormParameterValue(String name, String value, String type) {

ParameterProcessing paramProcessing = TagUtils.getInstance().getParameterProcessing(pageContext);
        return paramProcessing.processFormParameterValue(name, value, type);
}

public String processFormParameterName(String name, String type) {

ParameterProcessing paramProcessing = TagUtils.getInstance().getParameterProcessing(pageContext);
return paramProcessing.processURL(url, charEncoding);
}


        ...
}
```

```
/**
 * Base class for tags that render form elements capable of including JavaScript
 * event handlers and/or CSS Style attributes. This class does not implement the
 * doStartTag() or doEndTag() methods. Subclasses should provide appropriate
 * implementations of these.
 *
 * @version hdiv.org
 */
public class TagUtils {

...

    /**
     * <p>Returns the ParameterProcessing class on the specified module.</p>
     *
     * @param pageContext PageContext we are operating in
     * @return Returns the ParameterProcessing class on the specified module.
     * @author hdiv.org
     */
    public ParameterProcessing getParameterProcessing(PageContext pageContext) {

        ModuleConfig config = getInstance().getModuleConfig(pageContext);
      String paramProcessingClass = config.getControllerConfig().getParameterProcessingClass();

       return (ParameterProcessing) pageContext.getAttribute(paramProcessingClass,
                    PageContext.REQUEST_SCOPE);
}
```

We always invoke methods from the SPI to obtain the values of the attributes "name" and "value" using processName() and processValue() methods. This is the main rule that we have followed for all the tags. As we can see in the definition of the SPI, there are methods to be invoked in other cases, for example in the beginning and in the end of a form or when generating an url; For these cases "form" and "link" tags have been implemented. Let's see the implementation of the tag "form":

```
public class FormTag extends TagSupport {

        ...

        /**
```

```
         * Renders the action attribute
         */
protected void renderAction(StringBuffer results) {

        String calcAction = (this.action == null ? postbackAction : this.action);
        HttpServletResponse response = (HttpServletResponse) this.pageContext.getResponse();

        String url = response.encodeURL(TagUtils.getInstance().getActionMappingURL(calcAction, this.
pageContext));

        results.append(" action=\"");
        results.append(processStartForm(url)); <---- new implementation
        results.append("\"");
}

public String processStartForm(String action) {

ParameterProcessing paramProcessing = TagUtils.getInstance().getParameterProcessing(pageContext);
        return paramProcessing.processStartForm(action);
}

public int doEndTag() throws JspException {
        renderExtraParams();
        ...
}

/**
 * Adds fields to form
 * @throws JspException
 * @author hdiv.org
 */
public void renderExtraParams() throws JspException {

ParameterProcessing paramProcessing = TagUtils.getInstance().getParameterProcessing(pageContext);
        Map extraParams = paramProcessing.getExtraFormParameters();

        if (extraParams != null) {
                for (Iterator iter = extraParams.entrySet().iterator(); iter.hasNext();) {
                        Map.Entry entry = (Map.Entry) iter.next();
                        String key = (String) entry.getKey();
                        String val = (String) entry.getValue();
                        TagUtils.getInstance().write(pageContext, renderField(key, val));
                }
        }
}

        /**
         * Renders an HTML <b>&lt;input&gt;</b> element of type hidden.
         *
         * @param name hidden parameter name
         * @param value value
         * @return HTML <b>&lt;input&gt;</b> element of type hidden
         * @author hdiv.org
         */
        public String renderField(String name, String value) {
                StringBuffer sb = new StringBuffer();
                sb.append("<input type=\"hidden\"");
                renderAttribute(sb, "name", name);
                renderAttribute(sb, "value", value);
                sb.append(">\n");

                return sb.toString();
        }

        ...
}
```

In this case, we invoke **processStartForm** method to get the value for the "action" attribute in the form. And before the end of the form, we invoke **getExtraFormParameters** method to get the extra parameters to be added to the form. For instance, parameters for controlling the application's flow like *flowExecutionKey in Spring Web Flow, random token to avoid CSRF attacks, and _HDIV_STATE* parameter to guarantee integrity and confidentiality.

Source code: struts-taglib-1.3.8-ParameterProcessing-sources

3. Two implementations of the SPI

- empty (EmptyParameterProcessing): it maintains the Struts' present behaviour.
- HDIV (HDIVParameterProcesing): it guarantees data integrity and confidentiality.

In order to configure the SPI implementation, we must define the name of the class in the parameterProcessingClass attribute of the controller. Thus, the SPI implementation to use is defined for each module, making it possible to define different implementations for different modules. **By default**, if we don't define any value for this new attribute, the **HDIV’s implementation will be used**, which guarantees data integrity and confidentiality.

- struts-config-1_3.dtd file has been modified by adding the new "parameterProcessingClass" attribute.

See the example in point 4 to get more information about this configuration.

ActionServlet: The servlet has been modified to make it create a new instance of the SPI for each request made using the class defined in the controller.

4. Web application using these new components: struts-examples

We have configured the struts-examples application (struts-examples-1.3.8.war)provided with the Struts 1.3.8 distribution to make it use different SPI implementations.

- Default SPI configuration: No implementation has been configured for “/exercise”, “/upload” and “/hdiv” modules. This means that SPI’s default implementation (HDIVParameterProcessing) will be used, which guarantees data integrity and confidentiality.

- We have configured for “/validation” and “/attacks” modules SPI's empty implementation to maintain Struts' present behaviour.

```
<controller parameterProcessingClass="org.apache.struts.config.impl.EmptyParameterProcessing">
```