Apelon, Inc.
Suite 202, 100 Danbury Road
Ridgefield, CT 06877

Phone: (203) 431-2530
Fax: (203) 431-2523
www.apelon.com

Apelon Distributed Terminology System (DTS)

DTS Web Developers Guide

Table of Contents

Introduction 3

Architecture Overview 3

The Jakarta Struts Framework 3

Servlet Specification and Java Server Pages 6

Prerequisite Software 7

Setting Up a Web Application 9

The web.xml File 9

The struts-config.xml File 11

The server.xml File 11

Connecting the Dots 12

Guidelines for Developing a Web Application 14

Guidelines for Action Classes 14

Scope 14

Naming 14

Responsibilities 15

Guidelines for Developing Forms 17

Scope 17

Responsibilities 17

Guidelines for Writing Java Server Pages 20

Scope 20

Responsibilities 20

Creating Your Own DTS Web Application 22

References 23

© 1999-2009 Apelon, Inc. All Rights Reserved.

Introduction

The DTS Browser is part of the Apelon DTS client/server solution. The DTS Browser is provided as an example of a web application that can be developed using Apelon DTS.

The entire source of the application has been provided to you. You can develop your own application according to your needs by using or extending the DTS Browser application. The purpose of this document is to give you, the developer, the resources needed to develop web applications that use the Apelon DTS or to extend the DTS Browser.

This guide assumes that you understand basic Java development. You also should be familiar with general concepts in developing HTML, XML, Servlets, Java Server Pages and the Apelon DTS API.

Architecture Overview

The Apelon DTS API is written in the Java Programming Language (JDK 1.5). The natural choice for web applications using the DTS API is J2EE - Sun Microsystems' Java Platform 2 Enterprise Edition. Part of the J2EE specification is the Servlet specification, which is a solution for running web applications.

The DTS Browser is based on the Apache Foundation's Struts Web Application Framework (**Struts**). Struts is written using Java's Servlet Specification and also uses Java Server Pages (JSP).

The Jakarta Struts Framework

Struts is an open source framework, developed for encouraging an application architecture based on the Model-View-Controller (MVC) design paradigm. Struts now is part of the Apache Software Foundation (ASF) Jakarta project. Strut's official home on the web is http://jakarta.apache.org/struts.

The Struts mailing list can be found at http://jakarta.apache.org/site/mail.html. An archived mailing list can be found in the Mail Archive (http://www.mail-archive.com - search for "struts").

Struts consists of these four basic components:

1. **ActionServlet** – a single Servlet that controls all requests to the web application. The ActionServlet reads requests, routes the request to be validated (by an ActionForm class), routes the request to be handled (by an Action class), and finally sends the response with any results back to the client (usually a browser).
2. **ActionForm Classes** – the ActionServlet populates ActionForm classes with data from a HTTP request. Once populated with data, the ActionForm provides validation so that only valid data makes it into the system.

© 1999-2009 Apelon, Inc. All Rights Reserved.

If data in the AcitonForm is valid, the ActionServlet sends the ActionForm to an Action for handling. If errors are found during validation, there is a shared mechanism available in the framework for raising and displaying error messages. ActionForms can also be used to populate HTML forms in JSP pages.

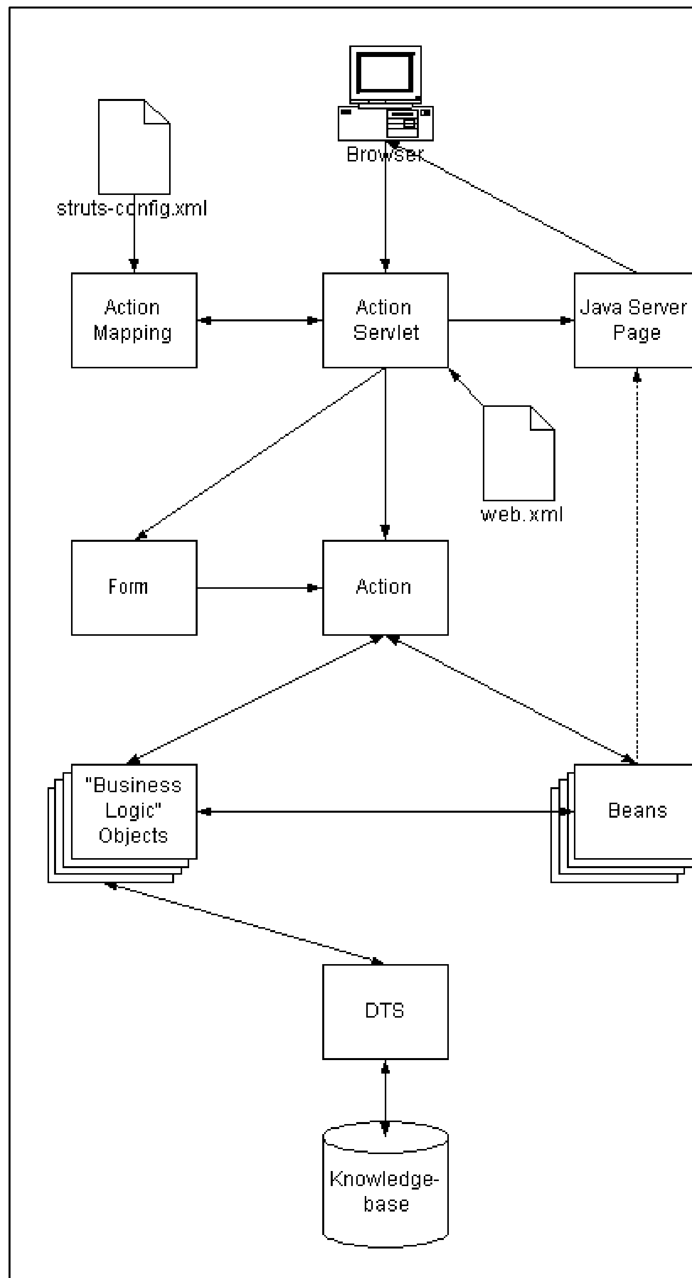
3. **Action Classes** – Actions invoke methods on objects in your application to perform the actual business logic. Actions are best used to focus on error handling and control flow. While an Action can implement business logic of the application, Actions are most effective when they invoke methods on other objects so that the logic of the application is separated from the flow and control of the web application.
4. **Utilities** – Struts provides utilities for parsing XML and also several Tag Libraries for use in JSP pages.

The **Tag Libraries** supplied by Struts allow JSP integration into the Struts framework. While the use of these libraries is not required to use the framework, you may want to explore them. These libraries include the following:

- **`Struts-html` tag** library - used for creating dynamic HTML user interfaces and forms.
- **`Struts-bean` tag** library - provides substantial enhancements to the basic capability provided by `<jsp:useBean>`.
- **`Struts-logic` tag** library - can manage conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.
- **`Struts-template` tag** library - contains tags that are useful in creating dynamic JSP templates for pages, which share a common format.

The code and descriptors that support the Struts tag libraries are distributed with the *struts.jar* archive. You can include these web libraries by modifying the web application descriptor file, *web.xml* (see the section ‘Setting up a Web Application’ for more details).

© 1999-2009 Apelon, Inc. All Rights Reserved.



For more information about Struts Tag Libraries see the *Developer Guide* and *TagLib Documentation* sections on the Struts release page (http://struts.apache.org/1.0.2/index.html).

Using the components described, Struts implements a Model-View-Controller (MVC) design pattern within the Servlet architecture. The following components make up the Model, View and Controller components of the framework.

- Your application logic ("Business Logic" Objects in the diagram above), your data and the Apelon DTS make up the Model Layer of the framework.

© 1999-2009 Apelon, Inc. All Rights Reserved.

- The View Layer contains JSP pages, HTML pages, and Struts Form classes that carry data from the view to the Controller. When errors occur the Form class can also carry data back to the JSP.
- The Struts ActionServlet class coupled with Struts Action classes comprises the Controller Layer. The ActionServlet dispatches incoming requests to an appropriate Action class, which updates the Model Layer components. The dispatch from the controller to the Action class is based on a configuration that is provided by an XML file. Struts Forms are also used to validate and carry data to the Action class. So you could say that Struts Forms are a part of the Controller as well as the View Layer.

The diagram illustrates the relationship between the different components of the Web application framework and how they work together.

Servlet Specification and Java Server Pages

Because Struts uses the Servlet architecture, here are a few points that will help you while developing your application:

1. Servlets are grouped as part of a Web Application called an Application Context.
    a. A Servlet Engine can run multiple web applications; each application has its own Application Context defined by a XML descriptor file called web.xml.
    b. Your application may not be the only application running on the server so be aware of your shared environment.
2. Servlets are Asynchronous – because requests can be sent across an asynchronous TCP/IP network, application state is lost between requests unless explicitly stored in memory or a database.
    a. An HttpSession object is provided in the Servlet architecture for saving state between requests – all Struts Actions have access to this object via the HttpServletRequest object passed to it by the ActionServlet.
    b. Java Database Connectivity (JDBC) and or Enterprise Java Beans (EJB) can be coupled with the Servlet architecture to save state to a database between requests. This is a better solution when you are running the same application on multiple servers. This allows you to share state between servers so any server can handle user requests.
3. There is only one instance of each Servlet in a web application.
    a. Servlets are very efficient because they pass requests to threads that can run simultaneously. When developing an application you need to be aware that your application will be running in a multi-threaded environment sharing a single instance of the ActionServlet and each Action class.

© 1999-2009 Apelon, Inc. All Rights Reserved.

4. JSP's are compiled and run in a Servlet Engine as a Servlet.
    a. Don't let the name fool you, JSP pages are not just HTML pages with a fancy name. JSP's are in fact Servlets and can do what ever a Servlet can do.
    b. Because JSP's are Servlets they can do a lot more than just present a view. However, to fit well into the Struts architecture JSP's should only include logic provided by Java Server Page Tag Libraries. This provides a separation of responsibilities so your code can be concentrated in your application, not in a group of web pages.

These few points are important to understand, but there is a lot more to the Servlet and Java Server Pages specifications. If you would like to review more details about Servlets, go to http://java.sun.com/products/servlet/.

You'll find more details about JSP at http://java.sun.com/products/jsp/.

Prerequisite Software

The following software resources are required or recommended when developing DTS Browser source code:

- A Java based Development IDE such as Borland's JBuilder IntelliJIDEA or Eclipse (recommended)
- XML Parser (required) – xerces.jar.
    - Description: XML parsing classes distributed by the Apache Foundation.
    - Used at: Run time
    - Location: **Apelon\DTS\tomcat\common\lib**. However, when compiling the source code, use parser.jar and jaxp.jar since the com.Apelon.common.dom package uses the Sun parser instead. In JBuilder this can be accomplished by including parser.jar and jaxp.jar in the project's "Required Libraries", but excluding them from the "Dependencies" of the project's Web Application component.
- Servlet API Classes (required) – servlet.jar.
    - Description: Java extensions for support of Servlets and Java Server Pages
    - Used at: Compile time and run time
    - Location: **Apelon\DTS\tomcat\common\lib**.

© 1999-2009 Apelon, Inc. All Rights Reserved.

- One of the following:
  - **ojdbc14.jar**
  - **Sprinta2000.jar** (if SQL Server is used)
  - Used at: Compile time and run time.
  - Location: **Apelon\DTS\tomcat\lib**.
- Logging (required) – log4j.jar
  - Description: Application logging utility classes.
  - Used at: Compile and run time.
  - Location: You can find it in the WAR file. The WAR file is located at **Apelon\DTS\tomcat\webapps\dtstreebrowser.war**. If you have run the Tomcat server once, you can find the log4j.jar at **Apelon\DTS\tomcat\webapps\dtstreebrowser\WEB-INF\lib** also.
- Struts Application Framework (required) – struts.jar, commons-beanutils.jar, commons-collections.jar, commons-digester.jar
  - Description: Struts is a web application framework from the Apache Foundation.
  - Used at: Compile and run time.
  - Location: You can find it in the WAR file. The WAR file is located at **Apelon\DTS\tomcat\webapps\dtstreebrowser.war**. If you have run the Tomcat server once, you can find the struts.jar at **Apelon\DTS\tomcat\webapps\dtstreebrowser\WEB-INF\lib** also.
  - Notes: DTS Browser source code is extended from classes found in this archive. All of these jar files should be placed in JBuilder's "Required Libraries" list.
- Tomcat 5.5 or higher (recommended) – This is a web server to run the DTS Browser. You can download it at http://jakarta.apache.org/site/binindex.html.
- Ant (recommended) – This is used to do the build and make the WAR file. You can download it at http://jakarta.apache.org/site/binindex.html.

In order to build your [dtstreebrowser].war file you need to create the *build.properties* file and *build.xml* file and run Ant. To see a detail discussion, go to http://struts.apache.org/1.0.2/installation.html. After you get your [DTSBrowser].war file built, you can install it in a Tomcat Server. For installation information refer to the "DTS Browser Installation Guide".

© 1999-2009 Apelon, Inc. All Rights Reserved.

Setting Up a Web Application

The web.xml File

A single ActionServlet will handle all requests that are made to your web application. In the case of the DTS Browser this class is `com.apelon.struts.common.ApelonActionServlet.java` that extends the Struts ActionServlet. Before being able to use Struts, you must set up a web application to run inside a Servlet Engine (also referred to as a JSP container). The Servlet Engine must be configured so that it knows to map all requests for your web application to the ActionServlet for handling. The configuration of the web application is done using an XML descriptor called *web.xml.*

The Servlet Engine reads the *web.xml* file at start up. The *web.xml* file allows you to tell the Servlet Engine the following:

1. The class that implements the ActionServlet
2. The start up parameter values needed when the ActionServlet is initialized
3. An URL extension mapping to the ActionServlet
4. The location of any Tag Libraries that you will use

In the case of the DTS Browser, the *web.xml* maps all requests with an URL ending in ".do" to the ApelonActionServlet. The following example displays how this is done:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
    <servlet>

            <!-- The next two lines define the action servlet to the Servlet Engine --
>

            <servlet-name>action</servlet-name>
            <servlet-class>com.apelon.struts.common.ApelonActionServlet</servlet-
class>

            . . .

    <servlet>

    <!-- maps all requests with extension of *.do to the Action Servlet -->
    <servlet-mapping>
            <servlet-name>action</servlet-name>
            <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    . . .

</web-app>
```

The discussion in the next section relates to how the ApelonActionServlet processes **.do** requests. More examples are provided in the *Guidelines for Developing Web Applications* section later in the guide.

© 1999-2009 Apelon, Inc. All Rights Reserved.

The *web.xml* file also defines to the Servlet Engine the location of Tag Libraries that can be used in JSP pages. In the case of the DTSBrowser, three Tag Libraries are used. Displayed is a snippet from the DTSBrowser web.xml file:

```
. . .
<web-app>
  . . .

  <!-- the DTS Browser uses the bean, html and logic Tag Libraries from Struts -->
  <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
  </taglib>
</web-app>
```

In the above example, the `<taglib-location>` tag references a local file for the Tag Library Definition files (*.tld). In this case, the tag library definition files used by the DTS Browser have been extracted from the struts.jar file for convenience.

Now that the web.xml file references the Tag Library, you can use this library in a JSP. To reference a Tag Library, you need to include a taglib directive at the top of your JSP. Each Tag Library that the JSP will reference needs its own taglib directive. Note an example of a tag-lib directive:

```
<%@ taglib uri="WEB-INF/struts-logic.tld" prefix="logic" %>
```

If the taglib directive above is included in the JSP, the Struts JSP Logic Tag Library can be included in the page. Here is an example of a call to the logic Tag Library:

```
<logic:iterate id="concept" name="concepts" scope="request">
   . . . Code to execute here . . .
</logic:iterate>
```

The example illustrates a call to the iterate tag in the Logic Tag Library. A full description of the Logic Tag Library can be found at http://struts.apache.org/1.0.2/struts-logic.html.

© 1999-2009 Apelon, Inc. All Rights Reserved.

The struts-config.xml File

The ActionServlet can also be configured with a XML descriptor of its own. The *web.xml* file specifies as one of the parameters to the ActionServlet a parameter called ***config***. The *config* parameter points to the *struts-config.xml* file, which configures the ActionServlet. When the Servlet Engine initializes the ActionServlet, the ActionServlet reads the *struts-config.xml* file. The relationships are stored in memory for optimal performance as an ActionMapping object.

The *struts-config.xml* file provides the ActionServlet with the following:

1. Mappings of URLs to the Action class that will handle the request.
2. Mappings of Action classes to their required Form class
3. Mapping of names to locations where Actions forward control

In short, the *`struts-config.xml`* file is really the glue of the entire framework. While *`web.xml`* defines where a request should go upon arriving, *`struts-config.xml`* provides the mappings so the ActionServlet can determine exactly what should happen to it.

The advantage of using a single configuration file is a modular system that is easier to maintain. It prevents the hard coding of URLs to be called within a component. Changes can be made by updating the configuration file without having to recompile or re-deploy applications.

When the ActionServlet starts up, it reads the *struts-config.xml* file and creates an ActionMapping object. This ActionMapping object is then used by the ActionServlet to look up locations for Actions, ActionForms and JSPs that were defined in *struts-config.xml*. The ActionServlet keeps a reference of the ActionMapping object until it is stopped. So if you update the struts-config.xml file, you have to stop and restart the ActionServlet before those changes take effect at runtime. In most cases this means you'll need to stop and restart your Servlet Engine after you make a change.

The server.xml File

One of the key elements in configuration of the server.xml file is the **context** tag for adding new web applications. View the *Context Container* discussion at the following link for details:
http://jakarta.apache.org/tomcat/tomcat-5.5-doc/config/context.html

View the Overview discussions at this link for general server.xml file information:
http://jakarta.apache.org/tomcat/tomcat-5.5-doc/config/index.html

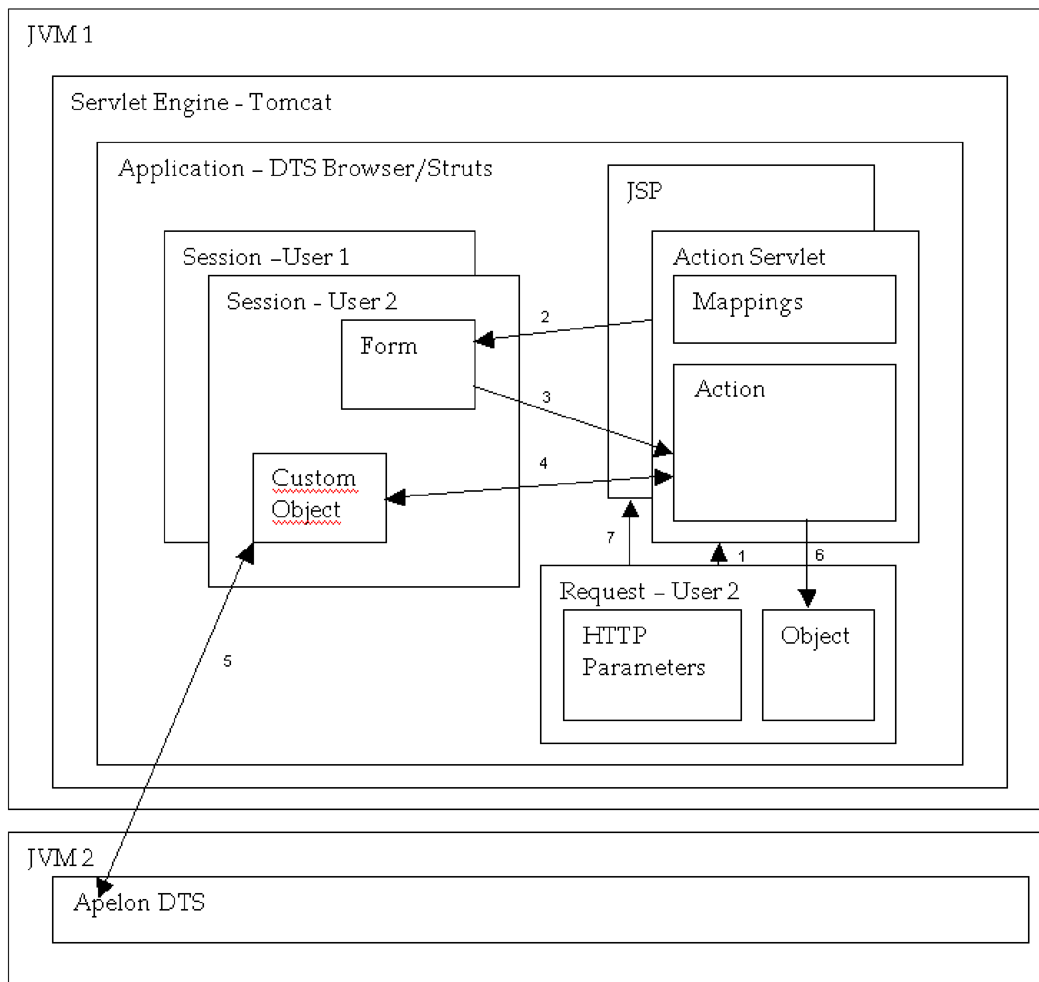
© 1999-2009 Apelon, Inc. All Rights Reserved.

Connecting the Dots

Now that we've reviewed the components of the framework and how to set the system up, let's look at what happens to a single request in the framework. The following diagram represents the DTS Browser application running inside the Java Virtual Machine (JVM).

The DTS Server runs in a separate JVM process. Communication between the two processes is done via XML over socket connections. In the diagram, the DTS Browser with the Struts framework is running inside the Servlet Engine on JVM 1. The DTS Server is running on JVM 2, which could even be on a different machine.

The diagram assumes that the Servlet Engine (Tomcat in this case) is up and running, previous requests have already been handled and several sessions have already been created for two users.



© 1999-2009 Apelon, Inc. All Rights Reserved.

The following steps correspond to the steps numbered on the diagram.

1. A user generates a request from a web page and the Servlet Engine routes the Request to the ActionServlet for handling.
2. The ActionServlet populates the ActionForm from the HTTP Parameters in the request.
3. The ActionServlet validates the form and sends it to the Action for processing along with references to the Mapping object, the Request Object, and the Response Object (not shown).
4. The Action sends messages to a custom object that handles the business logic of the application.
5. The custom object communicates with the Apelon DTS via a socket connection.
6. The result of the Action's processing is stored in the Request as an object (also referred to as a Bean).
7. The Request is forwarded by the ActionServlet to the appropriate JSP for display back to the user's web browser.

© 1999-2009 Apelon, Inc. All Rights Reserved.

Guidelines for Developing a Web Application

Once the environment is configured, the web application needs the logic and presentation to make it work. This section provides some guidelines that will help you develop a web application using Apelon DTS.

Guidelines for Action Classes

By the time a request gets to the Action Class, the ActionServlet has already parsed the request, all form validation has already been handled by the Form Class and the request has already been routed to the right place. An Action Class then sends the request's message to your application, handles any errors and directs the request to the next location.

The Struts framework does not limit the amount or type of code that you place in an Action. In fact, you can place all your application logic in Action Classes if you want to. However, if you want to separate your business logic from your Web interface, Action Classes are best used to invoke another object to perform the actual business logic. This lets the Action focus on error handling and control flow rather than business logic.

Scope

Actions should have a very specific purpose. Optimally, an Action should support a single type of action or a few related actions in an application. Actions are the messengers from a view to object containing the application logic. Actions should only have a single method implemented, the perform(…) method.

In this scenario, the application that is being acted upon should provide an interface of objects and methods that would allow any type of user interface to interact with it, not just a web interface. Actions should be unaware of anything about the application except for what it is passing to the application and the objects it gets back as a result.

Here are some examples of good uses of Actions:

1. Log a user into the system.
2. Save data entered into a form.
3. Update a list of items.

Naming

Action names should reflect the purpose of their existence. Here are examples of names:

1. LoginAction - Logs a user into the system
2. SaveConceptsAction - Saves a list of concepts selected by a user to a database.
3. UpdateConceptListAction - Updates the application's concept list with a new concept.

© 1999-2009 Apelon, Inc. All Rights Reserved.

Responsibilities

Actions can have any of the following responsibilities in the perform method:

1. Retrieve the data from a form for this request.
2. Retrieve any business logic objects from the Application, Session or Request scope that require action.
3. If needed, create any new business logic objects that the application may need.
4. Call methods on the business logic objects to complete the action.
5. If needed, store the business logic objects in the Application object, Session object, or a database for future requests.
6. If needed, store the results of the action in the Application context, Session object or Request object for the view.
7. Handle any errors that may occur in the application.
8. Determine the view object to forward to and return a reference to that view as an ActionForward object.

When limited to these responsibilities, Action Classes have the effect of separating the logic of an application from its web interface.

Example

Let's say that we are working on an application. This example has the following struts-config.xml that we are working on:

```
<struts-config>
 <!-- ========== Form Bean Definitions =================================== -->
  <form-beans>
   <form-bean name="saveForm"
              type="com.apelon.testapp.SaveForm"/>
   </form-beans>
   <!-- ========== Global Forward Definitions ============================== -->
   <global-forwards>
     <forward name="error" path="/Error.do"/>
   </global-forwards>
   <!-- ========== Action Mapping Definitions ============================== -->
   <action-mappings>
    <action path="/Error"
            type="com.apelon.testapp.ErrorAction"
            scope="request">
     <forward name="error" path="/error.jsp"/>
    </action>

    <action path="/Save"
            type="com.apelon.testapp.SaveAction"
            scope="request">
      <forward name="success" path="/confirm.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

This *struts-config.xml* refers to a global forward. The type of forward is available to any Action. The global forward called "error" is overridden by a local forward called "error" defined with the ErrorAction.

© 1999-2009 Apelon, Inc. All Rights Reserved.

This means that if ErrorAction's code refers to the string "error" the ActionServlet will map the forward to "/error.jsp". However, if SaveAction's code refers to the string "error", the ActionServlet will map it to "/Error.do", another Action.

In the example, the SaveAction is our first action (see the source below). The application is an editing application and has a business logic object called "workingDoc" stored in the user's session. The workingDoc is an instance of a SecureDocument that will be edited by the user during this session.

The SaveAction's design is to send a message to the document to save itself. You will see that the SaveAction only sends the message to the SecureDocument object. The SaveAction does not actually do any of the saving.

Once the save action is complete, the action stores a key in the request for the view to display. So in this example there are no inputs from a form, and there is a single output: the key of the document stored.

If errors occur, the Action stores the error message an error object and sends control to an error view.

If no errors occur, the Action stores the key and sends it to "success" or "/confirm.jsp".

```
public class SaveAction extends Action {

  public ActionForward perform(ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response)
    throws IOException, ServletException{

       // Reference to an errors object that may be needed if we have exceptions.
       ActionErrors errors = null;

       // get a reference to this user's session.
       HttpSession session = request.getSession();

       // get the business logic object needed for this action.
       SecureDocument doc = (SecureDocument)session.getAttribute("workingDoc");

        // Document Key object of the document we will save so we can open it later.
        Key key = null;

       // the document is sent the message to save itself returning a key.
        try{
          key = doc.save();
        }
        catch (Exception e){

          // the application had some problems executing so handle that here.
          errors = new ActionErrors();
          errors.add(ActionErrors.GLOBAL_ERROR,
                     new ActionError("error.savefailed",
                     e.getMessage()));
          saveErrors(request, errors);

          // display the error view
          return (mapping.findForward("error"));
        }

        // store the key of the document in the request for use in the next context.
        request.setAttribute("docKey", key);

        // display the success view.
        return (mapping.findForward("success"));
    }
```

© 1999-2009 Apelon, Inc. All Rights Reserved.

This example did the following:

1. Retrieved an application object from the session.
2. Executed a method on that object.
3. Handled any errors that may have resulted.
4. Saved the resulting object in the request.
5. Sent control of the web application to the next location
    a. If errors occurred, forwarded to a forward location called “error”
    b. If successful, forwarded to a forward location called “success”

Guidelines for Developing Forms

Forms are simply JavaBeans. Forms consist of the following:

1. Private String properties to hold parameter data passed from an HTTP request.
2. Setter methods for setting property values.
3. Getter methods for getting property values.
4. Validation method – to validate the properties before the Action can use them.
5. Reset method – to reinitialize the form for reuse by subsequent requests. (Forms are reused to save on memory in the Servlet Engine.)

That’s all there is to a Form. The ActionServlet first populates the Form with data from the request by using the public setter methods. The ActionServlet then calls the validate method on the Form. If valid, the Form is sent to the Action who can then get any of the properties on the Form.

Scope

A single Form class can be mapped to several Action classes. While you can map a single form to a single Action, there is no reason why a Form class cannot be more general. For instance, your application may have several different types of searches. Each search has its own Search Action such as “ConceptSearchAction”, “RoleSearchAction” and “PropertySearchAction”. If the parameters sent to each of these Actions are the same, only one Form is needed: “SearchForm”. SearchForm can then be mapped to each of the search actions in the *struts-config.xml*. This consolidates code and takes complexity out of your application.

Responsibilities

The only responsibility that is needed in a Form is the validation method. The validation method should confirm that all the Strings sent from the client in the request are valid. If they are not valid for your application, then you have two choices: either update the value to some default and continue or send an error back to the user.

© 1999-2009 Apelon, Inc. All Rights Reserved.

Example

Let's continue the example from the Actions section discussed earlier. Here is the struts-config.xml file again, but the save action has been updated to now use a form.

```
<struts-config>
 <!-- ========== Form Bean Definitions =================================== -->
  <form-beans>
   <form-bean name="saveForm"
              type="com.apelon.testapp.SaveForm"/>
   </form-beans>
   <!-- ========== Global Forward Definitions ============================== -->
   <global-forwards>
     <forward name="error" path="/Error.do"/>
   </global-forwards>
   <!-- ========== Action Mapping Definitions ============================== -->
   <action-mappings>
    <action path="/Error"
            type="com.apelon.testapp.ErrorAction"
            scope="request">
     <forward name="error" path="/error.jsp"/>
    </action>

    <action path="/Save"
            type="com.apelon.testapp.SaveAction"
            name="saveForm"
            scope="request">
      <forward name="success" path="/confirm.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

Save form has been defined as such:

```
public class SaveForm extends ActionForm {

      private static final String ENCRYPTION = "encrypted";
      private static final String CLEARTEXT = "cleartext";
      private static final DEFAULT_VALUE = CLEARTEXT;

      private String type = null;

      public String getType(){return type;}

      public void setType(String value){type = value;}

      public void reset(ActionMapping mapping, HttpServletRequest request){
        super.reset(mapping, request);
        type = DEFAULT_VALUE;
      }

      public ActionErrors validateForm(ActionMapping mapping,
                                   HttpServletRequest request){

        ActionErrors errors = new ActionErrors();
```

© 1999-2009 Apelon, Inc. All Rights Reserved.

```
      // if no value or value is not expected set to default.
      if (this.type == null ||
          (!type.equalsIgnoreCase(ENCRYPTION) &&
           !type.equalsIgnoreCase(CLEARTEXT)))
          this.type = DEFAULT_VALUE;
      }

      return errors;
    }
```

So SaveForm is very simple. It has one property called type. When the SaveForm is validated, if type is not set to a value or it is set to a value we don't expect, type is set to the default value. This ensures that any Action that uses a SaveForm will always have a valid value for property type.

Here's the updated Action class that will now use this Form (updates in **bold**):

```
public class SaveAction extends Action {

  public ActionForward perform(ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response)
    throws IOException, ServletException{

      // Reference to an errors object that may be needed if we have exceptions.
      ActionErrors errors = null;

      // get the type of document to save entered by the user
      String type = ((SaveForm) form).getType();

      // get a reference to this user's session.
      HttpSession session = request.getSession();

      // get the business logic object needed for this action.
      SecureDocument doc = (SecureDocument)session.getAttribute("workingDoc");

      // Document Key object of the document we will save so we can open it later.
      Key key = null;

      // the document is sent the message to save itself returning a key.
       try{
         key = doc.save(type);
       }
       catch (Exception e){

         // the application had some problems executing so handle that here.
         errors = new ActionErrors();
         errors.add(ActionErrors.GLOBAL_ERROR,
                    new ActionError("error.savefailed",
                    e.getMessage()));
         saveErrors(request, errors);

         // display the error view
         return (mapping.findForward("error"));
       }

      // store the key of the document in the request for use in the next context.
      request.setAttribute("docKey", key);

       // display the success view.
       return (mapping.findForward("success"));
    }
```

© 1999-2009 Apelon, Inc. All Rights Reserved.

Guidelines for Writing Java Server Pages

As discussed earlier in this document, Java Server Pages are Servlets and can do anything a Servlet can do. However, this does not always make for the best design of a system.

Scope

JSPs in the Struts architecture are used as simple views that place dynamic content into HTML pages. The content comes from objects that are saved either in the Application Context, the Session or a Request. The JSP should not be required to perform any logic against these objects. If logic is required, it should be placed either in the application or if necessary a Struts Action class. By keeping the scope of the JSP to the View Layer only, code is kept in your application making your project more manageable.

Responsibilities

JSPs should have the following simple responsibilities:

1. Retrieve objects saved in the Session, Request or Application contexts.
2. Insert dynamic content into a page
    a. Iterate over collections of objects (using the Struts logic Tag Library)
    b. Print values of object properties to the page.
3. Post data back to the server using the HTTP protocol
    a. This can be done using plain HTML, but Struts also provides the html Tag Library to help integrate HTML forms into the framework.

Example

Here is an example of the confirm.jsp that is part of the above example code. If you recall, the SaveAction forwards to a location called "success". The *struts-config.xml* maps "success" to the confirm.jsp page. Confirm.jsp shows a user that the document has indeed been saved and will display the key number of the document to the user (the SaveAction placed a key object into the request for the JSP to use).

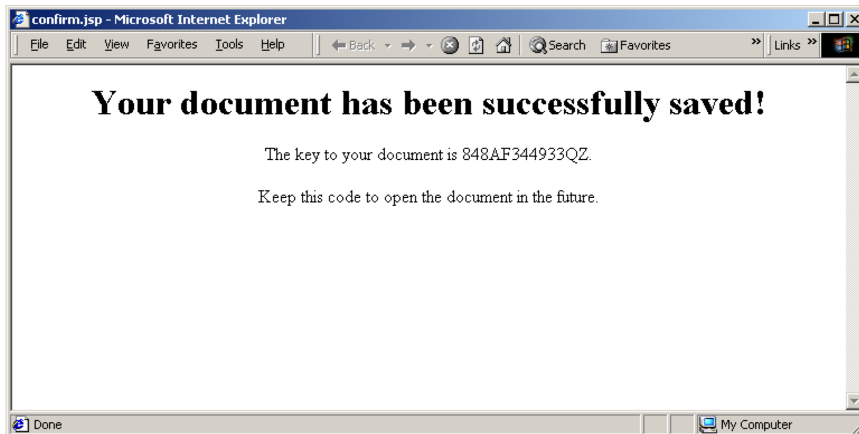
```
<%@ taglib uri="/WEB-INF/struts-bean.tld"  prefix="bean" %>
<html>
 <title>confirm.jsp</title>
  <body>
    <center>
     <h1>Your document has been successfully saved!</h1>
     <p>
       The key to your document is <bean:write name="docKey" property="code" />.
     </p>
     <p>
       Keep this code to open the document in the future.
     </p>
    </center>
  </body>
</html>
```

© 1999-2009 Apelon, Inc. All Rights Reserved.

The Key class has a single property called code that is accessed by the write tag of the bean Tag Library. Here's the source for the Key class:

```
public class Key {

  private String code;

  public Key(){
    this.code = CodeGenerator.createCode();
  }

  public String getCode(){
    return code;
  }

}
```

Here's what the confirm.jsp page prints out to the browser:



© 1999-2009 Apelon, Inc. All Rights Reserved.

Creating Your Own DTS Web Application

Before beginning your own application or extending the DTS Browser, the following references will help you when coupled with this document:

1. The DTS Browser JavaDoc
2. The DTS Browser source code
3. Struts documentation
    a. Home page: http://jakarta.apache.org/struts/
    b. User Guide: http://struts.apache.org/1.0.2/userGuide/index.html
    c. Java Doc: http://struts.apache.org/1.0.2/api/index.html

The Struts home page has excellent Developer Guides for each of the Tag Libraries. Look for links on the User Guide page.

© 1999-2009 Apelon, Inc. All Rights Reserved.

References

Web Sites

Struts: http://jakarta.apache.org/struts/

Servlets: http://java.sun.com/products/servlet/.
Java Server Pages: http://java.sun.com/products/jsp/.

Books

Core J2EE Patterns: Best Practices and Design Strategies by Deepak Alur, John Crupi and Dan Malks, Prentice Hall.

Java Servlet Programming by Jason Hunter with William Crawford, O'Reilly.

Web Development with Java Server Pages by Duane K. Fields and Mark A. Kolb, Manning.

Back to Top

© 1999-2009 Apelon, Inc. All Rights Reserved.