



# Struts Controller in Action

---

Gary D. Ashley Jr.

3<sup>rd</sup> Millennium Visions, Inc.

Garyashley at toughguy dot net





# Real Agenda

---

- What are the various configuration elements?
- How is all this configuration loaded?
- How is it used?

## Reasons:

- To show how you can extend the Struts framework to better suit your needs.



# Agenda (along the way)

---

- What are common features of Struts?
- What are some new features in 1.1?
- How is Struts configured?
- What is Digester?
- What is Struts made of?
- What are some pros and cons?
- What's next for Struts?



# What Is Struts?

---

Struts – open source implementation of MVC design pattern for building web applications based on Servlet, JSP, XML, and Java.

- Controller – primary component is a servlet, `ActionServlet`.
- View – one of the primary benefits of 1.0, and even 1.1 is custom tag library.
- Model – only general guidelines are provided by Struts.



# Struts Facts

---

- Originally created by Craig McClanahan, and donated to Apache in May 2000. (also Tomcat and JSF)
- 27 packages (12 top level), and several hundred classes/interfaces.
- Built upon several Jakarta projects (most in Commons):
  - BeanUtils, Collections, Digester, Lang, Logging, Validator, FileUpload, and ORO.



# Common Features

---

- I18N Support (Internationalization)
- Powerful custom tag library
- Form Validation
- Supports various presentation layers (JSP, XML/XSLT, JSF, Velocity, ...)
- Supports various model layers (JavaBeans, EJB, *etc.*)



# New Features to 1.1

---

## Main features:

- Module support
- Plugin Support
- Tiles Integration

## Other important notes:

- JSTL and EL are supported.
- Early Access version of JSF is included.
- Scaffold

<http://jakarta.apache.org/struts/userGuide/release-notes.html#Introduction>



# New Features to 1.2.x

---

- At the writing of this presentation, 1.2 has not yet been released, but it is anticipated that one or more releases will be made prior to the Summit in October.
- Some expected items will include removal of all deprecations since 1.0, and the integration of 1 or more new Jakarta Commons Projects.



# Struts Configuration

---

- There are various elements to configuring Struts:
  - web.xml
  - struts-config.xml
  - tiles-config.xml
  - validation-config.xml
  - messages-text.properties

[http://jakarta.apache.org/struts/userGuide/configuration.html#dd\\_config](http://jakarta.apache.org/struts/userGuide/configuration.html#dd_config)



# Web.xml

---

- ActionServlet instance
  - Init params (one for each module, plus default)
- ActionServlet mappings
- Struts Tag Libraries
- Note: the welcome file list can't be a Struts mapping, but you can use redirect:



# Action Servlet Instance

---

```
<servlet>
```

```
  <servlet-name>action</servlet-name>
```

```
  <servlet-class>
```

```
    org.apache.struts.action.ActionServlet
```

```
  </servlet-class>
```

```
...
```

**WARNING** – Struts **MUST** be a web application wide singleton, do not define more than one `<servlet>` element for an instance of `ActionServlet` (or a subclass).



# Action Servlet init params

- **config** – Context-relative path for the default module. [/WEB-INF/struts-config.xml].
  - This may also be a comma-delimited list of configuration files.
- **config/\${module}** – Context-relative path for the application module that will use the specified prefix (/\${module}).
  - This can be repeated as many times as required for multiple application modules. (Since Struts 1.1)



# Init Params *(Continued)*

---

- **convertNull** – Force simulation of the Struts 1.0 behavior when populating forms. [false]
- **rulesets** – Comma-delimited list of additional org.apache.commons.digester.RuleSet instances that should be added to the Digester that will be processing struts-config.xml files.
- **validating** – Should we use a validating XML parser to process the configuration file
  - (strongly recommended by Struts doc). [true]



# Action Servlet Instance *(Continued)*

---

```
<init-param>
```

```
  <param-name>config</param-name>
```

```
  <param-value>/WEB-INF/struts-default.xml</param-  
  value>
```

```
</init-param>
```

```
<init-param>
```

```
  <param-name>config/module</param-name>
```

```
  <param-value>/WEB-INF/struts-complaint.xml</param-  
  value>
```

```
</init-param>
```



# Action Servlet Mapping

---

```
<servlet-mapping>
```

```
<servlet-name>action</servlet-name>
```

```
<url-pattern>*.3mv</url-pattern>
```

```
</servlet-mapping>
```

**NOTE:** Commonly used is \*.do



# Welcome-File redirect

---

```
<welcome-file>index.jsp</welcome-file>
```

In jsp:

```
<%@taglib uri="/tags/struts-logic.tld" prefix="logic" %>  
<logic:redirect forward="index"/>
```

In struts-config:

```
<action path="/index" ...> ... </action>
```



# Struts Tag Libraries

---

- Be sure to setup each Struts tag library you plan to use in web.xml
  - `struts-bean.tld`
  - `struts-html.tld`
  - `struts-logic.tld`
  - `struts-template.tld`
  - `struts-nested.tld`
  - `struts-tiles.tld`



# Tag Library Setup

---

- Setup tag library in web.xml

```
<taglib>
  <taglib-uri>
    /tags/struts-html.tld
  </taglib-uri>
  <taglib-location>
    /WEB-INF/struts-html.tld
  </taglib-location>
</taglib>
```



# Struts-config.xml

---

- This is the central configuration file(s) to Struts, and any modules.
- Each defines various elements for:
  - Controller – flow & control
  - Message Resources – i18n files (key=value pairs)
  - Plugins – extension points
  - Data Sources – basic db config support
  - Form Beans – html form data/query string transport
  - Global Forwards – points to jsp/tile definitions, go here.
  - Global Exceptions – points to forward by exception type
  - Action Mappings – what code should run, based on url mapping.



# Struts-config.xml layout

---

```
<struts-config>  
  <data-sources />  
  <form-beans />  
  <global-exceptions />  
  <global-forwards />  
  <action-mappings />  
  <controller />  
  <message-resources />  
  <plug-in />  
</struts-config>
```



# Form Beans

```
<form-beans>
```

```
<form-bean name="myForm"  
  type="org.apache.struts.validator.DynaValidatorForm">
```

```
  <form-property name="oid" type="long" initial="-1"/>
```

```
  <form-property name="name" type="java.lang.String"/>
```

```
  <form-property name="values" type="java.lang.String[]"/>
```

```
</form-bean>
```

```
...
```

```
</form-beans>
```

NOTE: this form is a DynaBean class. See Jakarta Commons BeanUtils for more info.



# Global Exception

---

```
<global-exceptions>
```

```
<exception path="failure"  
  key="pageAlert.failure.validation"  
  type="biz.threemv.struts.action.ValidationException" />
```

```
</global-exceptions>
```

NOTE: I utilize the path as a mapping to forward and the key as message-text entry.



# Action Mapping

---

```
<action-mappings>
  <action path="/logon"
    type="org.apache.struts.x.LogonAction"
    name="logonForm"
      scope="request"
      input="/logon.jsp"
      unknown="false"
      validate="true" />
</action-mappings>
```



# Controller

---

```
<controller  
  processorClass="o.a.s.action.RequestProcessor"  
  debug="0"  
  contentType="text/html"/>
```

- This defines which RequestProcessor will be utilized by this module. The RequestProcessor is the key object in ‘controlling’ a Struts application.
- NOTE: There are several other configurable attributes.

<http://jakarta.apache.org/struts/userGuide/configuration.html>



# Message Resources

```
<message-resources
```

```
  parameter="biz.threemv.config.messages-common"  
  key="commonBundle"  
  null="false"/>
```

- This loads a text file of key=value pairs for internationalizing messages, or just managing text messages in general.
- NOTE: set null="false" to suppress throwing error messages when not found. It will display ???key??? instead to avoid nagging errors during development.
- WARNING: key="xxx" will save a bean in Application scope under that name.



# Tiles-config.xml

---

```
<tiles-definitions>
  <definition>
    <put />
    <putList>
      <add />
      <item />
    </putList>
  </definition>
</tiles-definitions>
```

NOTE: check out 'tiles-documentation' example that comes with Struts for many examples of what you can do here.



# Validation-config.xml

---

```
<form-validation>
  <global>
    <constant />
  </global>
  <formset>
    <constant />
    <form name="myForm">
      <field property="firstName"
        depends="required,mask,minlength">
        <arg0 />
      </field>
    </form>
  </formset>
</form-validation>
```



# What Points to What?

---

- For those new to Struts, one of the difficult things is the complexity of parts.
- `<action name="xx" />` points to `<form-bean name="xx"/>`

This maps any HTML form data or query strings into the `ActionForm` instance defined by the form bean and passes that form into the Action associated to the path.



# This Points to That...

---

- `<forward name="" />` is used inside Action class to determine where to forward the request to.
- `<forward path="/xxx.jsp" />` points to a jsp.
- `<forward path="xxx" />` points to (tiles)  
`<definition name="xxx" />`



# ... and That to That

---

- `<action name= "myForm" />`

**points to** `<form name= "myForm" />` ( in validation-xxx.xml)

**IF** `<form-bean name= "myForm"`

`type=" DynaValidatorForm or ValidatorForm "`

- `<action name= "myForm" path= "/fooPath" />`

**points to** `<form name= "/fooPath" />` ( in validation-xxx.xml)

**IF** `<form-bean name= "myForm"`

`type=" DynaValidatorActionForm or ValidatorActionForm "`



# Modules

---

- Modules allow the application to be divided up. This is important for a team environment.



# Module Configuration

---

Inside web.xml file.

```
<init-param>  
  <param-name>config</param-name>  
  <param-value>/WEB-INF/struts-default.xml</param-value>  
</init-param>
```

```
<init-param>  
  <param-name>config/module1</param-name>  
  <param-value>/WEB-INF/struts-module1.xml</param-value>  
</init-param>
```

<http://jakarta.apache.org/struts/userGuide/configuration.html#d>



# Caveats to Modules

---

- At the time of this writing, there are still several 'un' implemented aspects to modules.
  - For instance: can't have 2 form beans named same in 2 different modules because form bean names are NOT module aware yet.



# Plugins

---

- PlugIns – provides an extension point to add or enhance functionality.
- Several exist in Struts release:
  - TilesPlugin – Tiles Framework Extension
  - ValidatorPlugin – Validation Framework
  - ModuleConfigVerifier – set of verification tests to ensure module is configured properly.



# Other Plugin Ideas

---

- Business Model Hooks
  - Allow to initialize model components
  - Hooks to JMX, JNDI, JAAS, EJB Sessions, *etc.*
- Custom Tag Libraries
  - Allow for startup configurations
- Others:
  - Encryption/Decryption support (*i.e.*, passwords, credit cards, SSNs)



# Other Plugin Examples

---

- Some of my own include:
  - CaseManagementPlugin – business domain specific.
  - WorkflowEnginePlugin – implementation of WfMC business process workflow engine
  - PluginTagsPlugin – custom tag library that leverages external XML configurations (and digester). Used for things like TabMenus, DropDownMenus, Tables, *etc.* that can leverage custom XML configurations.



# Plugin Configuration

---

Inside struts-xxx.xml file.

```
<plug-in
  className="org.apache.struts.tiles.TilesPlugin">

  <set-property
    property="definitions-config"
    value="/WEB-INF/tiles-xxx.xml"/>

</plug-in>
```



# Tiles

---

- Tiles is a very powerful templating engine that allows you to assemble presentation pages from component, called "Tiles".



# Tiles Features

---

- Screen Definitions
- Layouts
- Dynamic Page Building
- Reuse of Tiles (components)
- Internationalization
- Multi-Channels



# Tile Screen Definitions

---

- Allow you to create a page of parts:
  - Header, Footer, TopNav, SideNav, and Body
- Definitions can be in:
  - One or more tiles-xxx.xml files
  - Directly in JSP pages
  - In Struts Actions
- Definitions have inheritance mechanism.



# Tiles Definition Examples

In tiles-default.xml file

A master template for use throughout the site.

```
<definition name="master" path="/main.jsp" >  
  
    <put name="title"           value="Default Title" />  
    <put name="banner"         value="/banner.jsp" />  
    <put name="topnav"         value="/topNav.jsp" />  
    <put name="body"           value="" />  
    <put name="formAction"     value="/nothing.do" />  
    <put name="footer"        value="/footer.jsp" />  
  
</definition>
```



# Tiles Definition Examples *(Continued)*

---

In tiles-xxx.xml file

Extend the master definition to provide page specific details.

```
<definition name="myPage" extends="master">  
    <put name="title" value="New page title"  
    />  
    <put name="body" value="/myPage.jsp"  
    />  
    <put name="formAction" value="/myAction.do" />  
</definition>
```



# Tiles – One Approach

---

- One approach is to use a 3 deep pattern. This is useful to manage complexity of Tiles.
  - Master Definition – defines elements common to entire application. (no path defined)
  - Style / Format Definitions – extends Master Definition to provide different styles utilized. *i.e.* InfoPage, EditorPage, HelpPage (path="xxxLayout.jsp")
  - Page Definitions – extends a specific Style Definition and provides specific items (various body jsps, menu, title, actionMapping to submit forms to, *etc.*)
    - addressSummaryView, addAddressForm, editAddressForm



# Tiles Layouts

---

- Define common layouts across application.
- Define menu layouts
  - I found using Tiles XML to define menus to be difficult and confusing, especially with junior developers on team. I use XML'ized custom tag plus Struts plugin mechanism instead. More recent work in this area may be better.
- Use existing layouts



# Tiles *(Continued)*

---

- Tiles are dynamic for each page reload, and can be changed,
  - *i.e.*, From inside Action
- Can reuse tiles in different locations.
  - *i.e.*, address.jsp
- Internationalization – can load different tiles according to Locale



# Enabling Tiles

---

- Setup tag library in web.xml

```
<taglib>  
  <taglib-uri>  
    /WEB-INF/struts-tiles.tld  
  </taglib-uri>  
  <taglib-location>  
    /WEB-INF/struts-tiles.tld  
  </taglib-location>  
</taglib>
```



# Enabling Tiles *(Continued)*

---

- In any jsps using Tiles:

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

Or if servlet 2.3 (doesn't need web.xml entry):

```
<%@ taglib uri="http://jakarta.apache.org/struts/tags-  
tiles" prefix="tiles" %>
```



# Enabling Tiles

---

- Tiles Plugin entry required in Struts config (see plugin slide for example)
  - Note for Modules – I recommend using a tiles-default.xml that is configured in struts-default.xml for any definitions that are common throughout application, *i.e.* Master and Style definitions. Then each module can have a comma delimited list of tiles-xxx.xml files that inherit from various layout definitions.



# Tiles, editorLayout.jsp

---

## setup

```
<%@ page language="java"%>
```

```
<%@ taglib uri="/tags/struts-tiles.tld"  
    prefix="tiles" %>
```

```
<tiles:useAttribute  
    name="formAction"  
    classname="java.lang.String" />
```



# Tiles, editorLayout.jsp

---

Head section

```
<html:html locale="true">
  <head>
    <title>

      <tiles:getAsString name="title" />

    </title>
  ...
</head>
```



# Tiles, editorLayout.jsp

---

Start body

```
<body>
```

```
  <table>
```

```
    <tiles:insert attribute="banner" />
```

```
    <tiles:insert attribute="topnav" />
```

```
  ...
```



# Tiles, editorLayout.jsp

---

Body cont.

```
<html:form action="<%=formAction%>">
```

```
<tiles:insert attribute="body" flush="true"/>
```

```
</html:form>
```



# Tiles, editorLayout.jsp

---

Finish body

```
<tiles:insert attribute="footer" />
```

```
</body>
```

```
</html:html>
```



# Tiles *(Continued)*

---

- Logging
  - Tiles uses Jakarta commons logging framework (which can hook to JDK1.4, Avalon, Log4J, *etc.*)
- There are a host of references on how to utilize all the capabilities of Tiles.
  - Book – *Developing Applications with Tiles* (by lead developers, Cedric Dumoulin and Ted Husted)



# How Is All This config Loaded?

---

- As you can see, there is quite a large collection of configuration information in multiple files.
- One of the nice things about Struts is that you can extend the configuration to suit your own needs by adding new elements and attributes.
- This is done using Jakarta Commons Digester.



# Digester

---

- A common component in Jakarta used to configure XML to Java object mappings.
  - *i.e.*, used in Tomcat and Jasper Reports
- Triggers Rules when XML pattern is recognized.
- A very rich set of predefined rules is built in.



# Digester Advanced Uses

---

- Plugin in custom pattern matching engine.
- Optional namespace-aware processing
- Encapsulate Rules into RuleSets



# Example of How to Use Digester

---

- Create an XML document.

```
<tabMenus>
```

```
  <tabMenu name="mainMenu">
```

```
    <images selected="tab_h.gif" default="tab.gif"/>
```

```
    <tabs>
```

```
      <tab name="books"
```

```
        displayKey="tab.books"
```

```
        action="/booksHome.3mv"
```

```
        module="/book" />
```

```
    </tabs>
```

```
  </tabMenu>
```

```
</tabMenus>
```



# Digester Example *(Continued)*

---

- Create Java Objects to map into.
  - TabMenusConfig
  - TabMenuConfig
  - ImagesConfig
  - TabConfig



# Digester Example *(Continued)*

---

- Let's look at Objects

- TabMenuConfig

- ✓ void addTabMenu(TabMenuConfig tabMenuConfig);

- TabMenuConfig

- ✓ private String name;

- ✓ void addTab(TabConfig tabConfig);

- ✓ void setImages(ImageConfig imageConfig);

- TabConifg (fields all have basic getters/setters)

- ✓ private String name;

- ✓ private String displayKey;

- ✓ private String action;

- ✓ private String module;



# Digester Example *(Continued)*

---

- Let's look at mappings
  - If this mapping is found:  
    /tabMenus/tabMenu  
    then create a new TabMenuConfig object add it to stack.
  - If this mapping is found:  
    /tabMenus/tabMenu/tabs/tab  
    then create a new TabConfig objects add it to parent TabMenu.



# Digester Example *(Continued)*

---

- So, first create a rule set:

```
public class ConfigRuleSet extends RuleSetBase {
```

- Then, implement method:

```
public void addRuleInstances(Digester digester) {
```

- Then add 3 lines (basic stuff) for each object to map:

```
digester.addObjectCreate(pattern, clazz);
```

```
digester.addSetProperties(pattern);
```

```
digester.addSetNext(pattern, methodName, paramType);
```



# Digester Example *(Continued)*

---

- Important Notes:
  - When processing Rules, Digester uses a basic Stack. (can peek, pop, *etc.*)
  - `addXxx` adds a rule to Digester object, it is NOT parsing XML until `.parse()` is called.
- `digester.addObjectCreate(pattern, clazz)`
  - Adds a rule so will create a new instance of `clazz` when the pattern is found and place on top of stack.



# Digester Example *(Continued)*

---

- `digester.addSetProperties(pattern)`
  - Adds a rule that sets all properties on the object on top of the stack using attributes with corresponding names.
- `digester.addSetNext(pattern, methodName, paramType)`
  - Adds a rule that calls method on top-1 object (2<sup>nd</sup> from top), and passes the object at top of stack as argument)
  - Used to establish parent child relationship as conveyed in XML layout.



# addRuleInstances(digester)

```
//TabMenuConfig settings
```

```
digester.addObjectCreate(  
    "/tabMenus/tabMenu",  
    TabMenuConfig.class);
```

```
digester.addSetProperties(  
    "/tabMenus/tabMenu");
```

```
digester.addSetNext(  
    "/tabMenus/tabMenu",  
    "addTabMenu",  
  
    "biz.threemv.TabMenuConfig"  
);
```

```
//TabConfig settings
```

```
digester.addObjectCreate(  
    "/tabMenus/tabMenu/tabs/tab",  
    TabConfig.class);
```

```
digester.addSetProperties(  
    "/tabMenus/tabMenu/tabs/tab")  
;
```

```
digester.addSetNext(  
    "/tabMenus/tabMenu/tabs/tab",  
    "addTab",  
    "biz.threemv.TabConfig");
```



# Use Digester

---

```
Digester digester = new Digester();  
digester.setUseContextClassLoader(true);  
digester.addRuleSet(new ConfigRuleSet());
```

```
InputStream xmlFile = getResourceAsStream(xmlPathName);  
TabMenusConfig config = new TabMenusConfig();
```

```
digester.push(config);  
digester.parse(xmlFile);
```

```
//that's it. Config is now loaded with TabMenuConfigs mapped  
    from your XML file.
```



# Struts and Digester

---

- Uses Init-Params to find XML documents.
- ActionServlet creates an instance of Digester during init() and uses o.a.s.config.ConfigRuleSet
- all struts-xxx.xml files, tiles, validation, *etc.* are 'digested' at init. (hint. you can use plugin to do same thing for your app specific stuff.)
- Stores a ModuleConfig instance for each module configured in web.xml in ServletContext (uses a Constant + prefix as key)



# Struts and Digester

---

- You can add more rule sets and provide your own ModuleConfig implementation to completely customize your apps configuration.
- To try and minimize getting 'out of sync' with Struts I follow few guidelines.
  - Extend existing config objects where possible, and provide additional properties.
  - Use Plugin classes to load new config constructs that are outside scope of base Struts implementation. *i.e.*, for building navigation menus.



# Action Mapping Example

```
<action-mappings
  type="biz.threemv.struts.action.HelperMapping">
  <action
    path="/basicInfoPage"
    type="biz.threemv.struts.action.HelperAction"
    name="myForm"
    helper="biz.threemv.actions.MyActionHelper"
    beans="makeXxxBean, makeYyyBean"
    defaultForward="success"
    roles="myRole"
    <forward name="success" path="displayMyPage" />
  </action>
```

- Key is providing a type="" in action-mappings (global) or a className in <action> element



# Let's Look "under the hood"

---

- Once Struts gets 'started' *via* `ActionServlet.init()`, and 'digests' all the XML configurations as `ModuleConfig` instance, it uses:  
`org.apache.struts.action.RequestProcessor`
- `doGet()` and `doPost()` both call `process()`, which in turn finds the `ModuleConfig` mapped to the URI, and gets the `RequestProcessor` from it. In turn, calls a `process()` method there.



# Recap - Struts-config.xml

---

- This is the central configuration file(s) to Struts, and any modules.
- Each defines various elements for:
  - Controller – flow & control
  - Message Resources – i18n files (key=value pairs)
  - Plugins – extension points
  - Data Sources – basic db config support
  - Form Beans – html form data/query string transport
  - Global Forwards – points to jsps/tile definitions, go here.
  - Global Exceptions – points to forward by exception type
  - Action Mappings – what code should run, based on url mapping.



# RequestProcessor

---

- The `o.a.s.a.RequestProcessor` handles most of the core processing for the servlet.
  - NOTE: A current design issue is the number of subclasses for popular extensions, and the incompatibilities between them.
- Has a set (currently) of 15 helper methods.



# RequestProcessor

---

- processPath
- processLocale
- processContent
- processNoCache
- processPreprocess
- processMapping
- processRoles
- processActionForm
- processPopulate
- processValidate
- processForward
- processInclude
- processActionCreate
- processActionPerform
- processForwardConfig



# Request Process methods

---

- `processPath`
  - Determine the path that invoked us. This will be used later to retrieve an `ActionMapping`.
- `processLocale`
  - Select a locale for this request, if one hasn't already been selected, and place it in the request.
- `processContent`
  - Set the default content type (with optional character encoding) for all responses if requested.



# Request Process methods

---

- processNoCache
  - If appropriate, set the following response headers: "Pragma", "Cache-Control", and "Expires".
- processPreprocess
  - This is one of the "hooks" the RequestProcessor makes available for subclasses to override.
- processMapping
  - Determine the ActionMapping associated with this path.



# Request Process methods

---

- processRoles
  - If the mapping has a role associated with it, ensure the user has the specified role. If they do not, raise an error and stop processing of the request.
- processActionForm
  - Instantiate (if necessary) the ActionForm associated with this mapping (if any) and place it into the appropriate scope.
- processPopulate
  - Populate the ActionForm associated with this request, if any.



# Request Process methods

---

- processValidate
  - Perform validation (if requested) on the ActionForm associated with this request (if any).
- processForward
  - If this mapping represents a forward, forward to the path specified by the mapping.
- processInclude
  - If this mapping represents an include, include the result of invoking the path in this request.



# Request Process methods

---

- `processActionCreate`
  - Instantiate an instance of the class specified by the current `ActionMapping` (if necessary).
- `processActionPerform` \*\*\*
  - This is the point at which your action's `execute` method will be called.
- `processForwardConfig`
  - Finally, the `process` method of the `RequestProcessor` takes the `ActionForward` returned by your `Action` class, and uses it to select the next resource (if any).

# Soccer Club Registration

## Demo

- Goals:
  - Create a plugin that uses digester to help build tabMenu
  - Create new ActionMapping used to run jasper report.
- Demo:
  - Basic welcome, login, register pages.
  - Provide tabMenu over registration process.
  - Provide PDF 'receipt' of my registration.
  - Running on Jboss 3.0/Tomcat 4.1.24
  - Using Eclipse 2.1.1 as IDE



# What's Next?

---

- Composeable Request Processors
- Pattern matching for Action Mapping paths.
- JSF
- DB support for MessageText
- Better tools to manage complexity (but hopefully still leverage power of flexibility).



# The End.

---

Thank you for coming.

**Gary D. Ashley Jr.**

3<sup>rd</sup> Millennium Visions, Inc.

garyashley at toughguy dot net

