



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

The Struts Framework

Version 1.3

Instructor's Guide



Overview

This is a relatively advanced course, and requires strong Java programming skills along with good knowledge of servlets, JSP, and basic XML. Well-qualified students should need little persuading that Struts is a useful tool, having bumped up against many of the frustrations of building J2EE Web applications that started the Struts project in the first place. The Struts framework ought to seem intuitive once the basics are introduced, and a week full of excited discussions of design patterns and best practices can certainly ensue. Don't be surprised to find well-qualified students jumping ahead or anticipating topics not yet covered; experienced developers will see many of the issues normally raised in later chapters as soon as they start building Struts applications in early labs.

However, in practical fact many classes will be populated at least partially with students who either don't have the prerequisites in toto or have them on paper but lack experience and can't code and test very productively. Be prepared to adjust the pace of the class in the early going; Chapters 2 and 3 can carry the class well into Day 2 and, to be certain that everyone completes the labs in the early part of the course, it's often necessary to jettison some of the latter material.

The course can be partitioned into three sections, each expanding concepts and skills to a larger "ring" of completion. The first part would be the first three chapters, and it is this "kernel" of the Struts framework that is absolutely essential to convey thoroughly, even to the relatively inexperienced student. Chapters 4-7 expand to a more complete practical set of skills including tag libraries and validation, and the remaining three chapters are mostly advanced techniques and best practices, including an introduction to Tiles.





Timeline

The following breakdowns are approximate, and every class will vary.

Day 1

2½ hours	Chapter 1
4 hours	Chapter 2

Day 2

3 hours	Chapter 3
2 hours	Chapter 4
2 hours, runs to day 3	Chapter 5

Day 3

1½ hours	Chapter 6
4 hours	Chapter 7
2½ hours, runs to day 4	Chapter 8

Day 4

2 hours	Chapter 9
2 hours	Chapter 10

If time is tight, the best candidates for skipping or skimming are probably Chapters 6, 9 and 10, unless there is strong interest in Tiles in which case Chapters 6, 9 and 8 should be let go, in that order. Lab 6 might also be skipped even if the rest of the chapter is covered.



Ant Build Process

Though most students will be happy to let the **ant** command take care of things, some students (and most instructors) will want to understand the inner workings here a little better. Each web project in the course has its own **build.xml** and **build.properties** files; these rely on the master environment variable **CC_MODULE** to import targets and properties in a central directory, **%CC_MODULE%\Ant**. Information here defines a routine for building a Spring application; we won't document this in detail here, but rather summarize the interesting tasks:

- Clean out and create a **build** directory, which becomes the root of a web application (cleanup is a little tricky since Tomcat likes to lock certain files)
- Compile Java source files to **build\WEB-INF\classes**
- Copy resources from **docroot** to **build** – this may include HTML and JSP, the **web.xml** and Spring configuration files, message resources, and custom tags
- Copy the Spring JAR into **build\WEB-INF\lib**
- Create a Tomcat context file, e.g. **LandUse.xml** for the LandUse application, and copy it into Tomcat's **conf\Catalina\localhost** directory

Not everyone is familiar with this last step, or with this approach to deploying applications to Tomcat. Note that there is no WAR file, nor do we truly deploy our code to Tomcat. Rather, we use the context file to tell Tomcat where to find our application, and to map it to a certain context-root URL. This has the advantage of keeping our build entirely local and letting Tomcat reload as necessary when we change a JSP or class file. It can lead to some confusion, though: once the context XML is in Tomcat's tree, a build failure will leave Tomcat thinking that the application is deployed when it's not even well-formed. This may manifest as exceptions showing in Tomcat's console on a later startup. It's typically harmless, and easy to fix: just rebuild the application completely, from any of its example, demo, or lab steps; the build will overwrite the current context XML to point to itself, and Tomcat will start serving files from that recent build.



Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse WTP 2.0. (See the course Setup Guide for download URLs.) Instructors, use this package on your own initiative and at your own risk. You should have significant experience yourself with Eclipse WTP before using the overlay package in the classroom. This overlay has been tested lightly but are not part of the standard product.

That said, this overlay should make life a lot nicer for those looking for more than the bare-bones environment specified for the course. Eclipse WTP's specialized editors, integrated coding and debugging, and refactoring tools can certainly improve the student's learning experience. See the documentation inside the workspace itself for general notes on usage, and be prepared to walk students through the first few exercises in Eclipse. The notes in this file are for experienced Eclipse users and will not be clear to many students without your help.

Note especially the file **Struts Module Notes.html** within that workspace documentation; this points out specific issues related to using Eclipse WTP with this course.

Again, Capstone Courseware can only offer complete technical support on the standard course, and while we hope this overlay is convenient, it is not as thoroughly tested as the core lab image at this time. If a given exercise is giving trouble, please be certain to build and run it from the command line, using the SDK tools as prescribed in the student guide, before contacting Capstone.





Crimson and the XML Validator

For those classrooms not using Eclipse, it will be worth the trouble to introduce students to a simple XML authoring environment deployed with the course software. The setup guide requires that the Crimson text editor be installed to `c:\Capstone\Tools\Crimson3.70`; then, the lab installer bundles an XML validating tool from Capstone Courseware (just a command-line interface to Xerces, really) and a single file that installs custom hotkeys into Crimson. The end result is that Crimson acts as an integrated editor, parser and schema validator for XML files. Hit **F9** to parse for well-formedness and **F10** to validate against a DTD that's referenced in the XML document.

Students may find this helpful when creating and editing their Struts config files, validation rules, command-chain catalogs, and Tiles definitions. Especially if you sense that students' XML experience is slight, take a few minutes to show them this environment, and suggest that they make a habit of saving with **F10** rather than simply **File|Save**; this will trigger schema validation of their XML files and alert them to any of the common syntax or validity mistakes that can otherwise become frustrating as they don't surface until after the whole application's been run or deployed to Tomcat.

Note however that the standard `<!DOCTYPE>` declarations in these files refer to remote DTD URLs. If the classroom does not have web access it will not be possible to validate with **F10**; still the basic (and probably more valuable) parsing with **F9** will work. If a configuration file is giving some real trouble and it's worth taking a moment to validate it, edit the file so the DTD reference is to the local copy of the Struts 1.3 configuration-file DTD in `c:\Capstone\Tools\Struts1.3\doc\dtts\struts-config_1_3.dtd`.



Teaching Notes

Chapter 1

This chapter serves a few purposes. Ostensibly, it gives a high-level tour of the Struts framework, spending as little as a single page on each of several major features that are covered in their own chapters later in the course. It also provides a dead-simple example to give students a sense of what a Struts application looks like, to introduce the configuration and especially to illustrate the emphasis on XML declaration over Java programming in Struts. Finally, it gives the instructor a chance to prove out most of the environment on students' machines, by working through the simple example application build and test. Once the Housing example is working on a given machine, the only piece of the setup that could be out of whack is MySQL and the JDBC driver config for Tomcat, and this comes up in Chapter 2.

Chapter 2

The foundation of Struts, of course, is the action mapping, and this chapter goes slowly in bringing out this concept. Several simple examples are provided and students get their first chance to build and configure Struts actions. The first example may seem strange, in that it doesn't use any action classes! At first this may seem to undercut the point of this chapter, but a couple important concepts are proved here: (a) there is value in using action mappings even when connecting one view to the next, and (b) actions are not usually called for where no true "control" is being exerted over the model. This latter idea especially will resurface in later chapters as students confront the design choices typical to Struts: what goes in actions vs. views vs. form beans vs. the model?

The latter and smaller section of this chapter covers declarative exception handling, which after actions and action mappings should be straightforward for students to grasp.





Chapter 3

The basic facility of form beans should be evident right away. However the real power of form beans will probably not be obvious to students just yet, especially because they haven't started working with sufficiently complex applications to see the value of form beans as business-tier adapters, and because we have yet to talk about custom tags and validation. Some hand-waving will be required! Try to sell form beans as the basis for a number of more exciting features, such as the Struts HTML tags and the Struts Validator, and students won't have long to wait to see the real power of form beans in practice.

Chapter 4

This and the next chapter combine to develop a toolkit of custom tags for developing view components in JSP. As such there is some forward-reference to the JSTL chapter, and the two chapters will feel like a coherent unit in class. The primary focus in this chapter is the HTML tag library, which also continues the case for consistent use of form beans. Seeing how the `<html:form>` and child tags streamline the input and output for an HTML form should solidify students' understanding of the advantages of using form beans in the first place. The exact mechanics of pre-population are tricky, and can be confusing. A page or two and a diagram are provided to clarify this, but don't burn a lot of time on this, even if it doesn't seem to be sinking in. Note that the look-ahead quality of this tag will show itself in the demonstration, as we encounter a fatal error when it can't find the action mapping thanks to a careless refactoring that leaves the **action** attribute in a page-relative form rather than the context-relative form that the tag handler expects. Refer very interested students to the source code for the `<html:form>` tag handler, and also rely on the use of the Periscope at the end of the demo to show the creation of the global FORM and BEAN objects.





Chapter 5

Although the course title is simply, “The Struts Framework,” the scope really includes tools commonly used in Struts development, and JSTL certainly qualifies. This is of course not an in-depth treatment of JSTL; we have a separate two-day course that can be recommended as a companion to this one. Instead, the chapter has a straightforward survey structure: an overview, including the EL, and one example for each of the four libraries Core, Formatting, SQL, and XML. The Core library is far and away the most useful of these, and should be given the most weight – this is reflected in the lab work as well.

Chapter 6

Internationalization and localization are relevant to many kinds of software – not just Web applications, and not just Struts. Still, Struts integrates i18n so thoroughly that it's really not possible to understand application code or configuration files completely without some grounding in locales, resource bundles, etc. So this chapter starts with a quick primer on Java i18n and continues to show how Struts uses localized resources in configuration and Java code. This chapter should not take a long time; for many students the Java i18n part will be review, and if time is tight, Lab 6 is a good candidate for skipping. i18n is quietly threaded into just about every other exercise in the course.



Chapter 7

Validation has become a huge topic within Struts. There are several levels and techniques, plus challenging architectural questions, and so this is one of the longest chapters in the course. Give all the sections of this chapter plenty of time, and encourage discussions of tactics and strategy at many points, especially once the Struts Validator has been covered. Lab 7A in particular illustrates many of the issues: it is just complex enough to exceed the declarative capabilities of the Struts Validator for a coarse-grained bean such as MemberBean, and even multi-page validation doesn't quite solve the problem cleanly.

Notice the odd validator-by-validator behavior of the generated JavaScript when multiple validators are used. In the Ellipsoid JavaScript demo, try entering one good value, one negative value, and leaving one field blank. You'll see a message box about the required value, but not the negative one. Then fill in the missing value with a positive number, and you'll see a message about the negative value. This is not optimal but it's the current state of the art with the `<html:javascript>` tag.

In an early offering of this course, in fact, a student exposed a weakness in the answer code that I wasn't aware of: try navigating as far as **Update.jsp**, deleting a required profile field such as age, submitting – which fails validation and returns to the same page – but then changing the password. The validator will continue to complain that age is missing, even though it's not submitted by **ChangePassword.jsp**! Sort of a deep bug, but enough to make the case that multi-page validation is not sufficient to this problem – which is why I've left it in there. If no one notices, you may want to demonstrate this yourself.

Also, make a point of keeping the Struts validation framework well separated from the Struts Validator in discussions. It's all too easy for students to start thinking that the Validator plug-in is the only way to do validation.

Instructors familiar with earlier versions of Struts may want to review the new locations of some files: **validator-rules.xml** and **chain-config.xml** both live in the Struts core JAR, at internal paths **org/apache/struts/validator/validator-rules.xml** and **org/apache/struts/chain/chain-config.xml**. One really tricky thing about these two is that one uses a leading slash in declaring the former in the Struts Validator plug-in, but does not use a leading slash in **web.xml** when referring to the latter file as a chain configuration.

Finally, one irritation with the Struts "required" validator: it doesn't agree with `<select>/<option>` lists too well, at least not on the client side. The generated JavaScript, running in IE6, will say that the value is required, even when it's been selected. The same page will work correctly in FireFox, but clearly that's not good enough. In **LandUse**, we've skipped validation on comment recommendation fields, for this reason.



Chapter 8

Especially over the last two revisions, Struts has added a number of interesting features, many of which really boil down to ways to simplify the configuration file – things like wildcards and extensions. After a demo of Struts's role-based security, another big theme in this chapter involves ways of extending and customizing Struts: specifying one's own config-object subclasses and using `<set-property>` to configure them, using plug-ins, configuring the request processor, and defining command chains.

This last feature is one of the most interesting, especially when set against the older (and more suspect) practice of chaining actions. The mechanism of the CoR request processor is really very simple, but it takes a while to put all the moving pieces into view and for students to separate out the possible uses. We focus on two scenarios: (1) you want to add pre- and post-processing commands to your handling chain, and (2) you want to modify fundamentally the default request-handling chain. The Flow demo illustrates both of these, and generally it's easiest to talk about (1) before getting into (2). The second lab, on auditing, extends the security demo and focuses on scenario (1).

The more advanced the group, the more likely they are to seize on this segment, which only makes sense given that the `ComposableRequestProcessor` was introduced in response to problems involved in some specific advanced uses of Struts 1.1 and 1.2. You could toy around with the demo on this subject almost ad infinitum: for starters, try plugging **ExecuteCommand** in elsewhere in the processing chain – perhaps before form-bean population.

Also, a note about the plug-in strategies discussed in this chapter: static fields vs. application-scope attributes and so on. The final approach listed in the coursebook is probably the best one, if for no other reason than that it would support clustering where statics won't work very well in that situation.



Chapter 9

This is the inevitable catch-all chapter! shining a flashlight in some of the darker corners of the Struts API. We get a clearer sense of the term “global object” that’s been used throughout the coursebook, and note that Java and JSP code can interact with such an object, if the developer knows its key and scope. Canned **Action** subtypes are worth some discussion, and we provide a quick demonstration of **LookupDispatchAction**. The section on **BeanUtils** ought to be interesting, for the utility itself and for the form-bean-as-adapter techniques it discusses. The adapter solution in Lab 9A is not the most elegant, but it’s a major step forward and should suggest to students the sorts of things they could do in building their own base or delegate form-bean classes as part of a larger development effort. The last section revisits validation and essentially leaves students with the assertion that there is no one perfect architecture here; the Struts Validator is a clever but limited tool and application-specific design choices will almost always be necessary.

Chapter 10

The Tiles tag library has been held out of the Struts-tags chapter and deferred until now, and in a way this chapter is of more of a piece with Struts-tags and JSTL than with anything that precedes it more directly. It is possible to cover this chapter right after JSTL, too, if you prefer. The lab starter code includes developments from Chapters 7-9, but these should be transparent for the moment and shouldn’t interfere with lab work. Tiles has been held for last in the book simply because most Struts developers are not using Tiles; thus for many classes it will be an optional chapter. It also introduces a fairly broad new framework of its own – on a par in complexity with the Struts Validator – and is not integral to Struts development as shown in Chapters 6-9. It seems easiest to take a deep breath after Chapter 10 and then dive into this new world of view management.

In my view, the real advantages in using Tiles are (a) basic reuse of content, à la `<jsp:include>` but with the clever inversion of who’s doing the inserting and controlling reuse, and (b) the ability to progressively classify view types using definitions. This chapter focuses on those “wins” but stops short of another that has its proponents: using definitions as forward targets in action mappings. I personally don’t see this as an attractive choice in most cases – it seems overly centralized and awkward. There are different schools of thought on this, and I’ve seen highly dynamic page-building applications that forward to definitions files and even use the Tiles actions to redefine or reload those definitions on the fly. If you do like this technique, by all means consider extending the chapter or exercises in some way to show that. We’d be interested in hearing any feedback in this area, or in seeing examples you may cook up.



Quizzes

You may find the quizzes on the following page to be useful learning devices. We've had good feedback on them, the first when presented soon after covering Chapter 3, and the second when presented after Chapter 7. They offer a way to reinforce concepts. Present them not as evaluations of the students but rather as one more way to get everyone thinking about the subject and reminding themselves what they know and perhaps don't yet know as well as they want to. Give everyone a few minutes to take their best shots at all the questions, and then review as a group and hand out the answer sheet.

The following pages are the blank quizzes, each one followed by answer sheets. You may print these out and distribute them to the class at whatever times seem to make sense for your specific schedule – shooting for first thing in the morning or after lunch is often good.





Quiz: Struts Action Mappings and Form Beans

Answer the following questions briefly before discussing with the whole class.

1. How does a servlet container know to route a request to the Struts ActionServlet?
2. How does ActionServlet find the struts-config.xml file?
3. Can struts-config.xml be renamed? Can it be relocated? If so, what would have to change with it, and are there any constraints on new names or locations?
4. What's the required base class for a Struts action?
5. What's the required base class for a Struts form bean?
6. Can you have multiple <action>s with the same path attribute?
7. Can you have multiple <action>s with the same type attribute?
8. Can you have multiple <action>s with the same <forward>s?
9. We've seen <action> mappings forwarding to JSPs. Could you forward to an HTML page?

Extra credit (!):

10. How about forwarding to a URL outside the application -- could I define a forward to `http://java.sun.com`?
11. What would happen if you changed the <url-pattern> in web.xml from `*.do` to just `*?` (For instance in the LoveIsBlind case study.)





Struts Action Mappings and Form Beans – Answers

1. How does a servlet container know to route a request to the Struts ActionServlet?

`<url-pattern>*.do</url-pattern>` in `web.xml` dictates which URLs will be mapped to the ActionServlet. It maps to a servlet name, and elsewhere in `web.xml` that name maps to the ActionServlet class.

2. How does ActionServlet find the `struts-config.xml` file?

In the `<servlet>` element for ActionServlet, there is an `<init-param>` that identifies the config file.

3. Can `struts-config.xml` be renamed? Can it be relocated? If so, what would have to change with it, and are there any constraints on new names or locations?

It can be renamed and/or relocated. The `<init-param>` exists to allow this to be determined by the developer. The path can't be `/WEB-INF/web.xml!` or anything else that would collide with an existing file. It also shouldn't be outside the `/WEB-INF` tree; it's a config file and should be hidden from outside addressing, which is what `WEB-INF` is for.

4. What's the required base class for a Struts action?

`org.apache.struts.action.Action`

5. What's the required base class for a Struts form bean?

`org.apache.struts.action.ActionForm`

6. Can you have multiple `<action>`s with the same path attribute?

No. Paths have to be unique as keys, or Struts' ActionServlet won't be able to decide which controller to use for a given request.

7. Can you have multiple `<action>`s with the same type attribute?

Yes. This means many URLs could resolve to to the same controller -- that's fine.

8. Can you have multiple `<action>`s with the same `<forward>`s?

Yes. This means multiple paths and/or multiple controllers resolving to the same view -- also fine.





9. We've seen `<action>` mappings forwarding to JSPs. Could you forward to an HTML page?

Yes. Any URI in the application will work. The only no-no is something that would mistakenly be trapped by the `<url-pattern>` in `web.xml`! Remember that a forward is resolved from the front of the system, meaning the Web container sees it and maps it through `web.xml` to `ActionServlet`, if the `<url-pattern>` says so. This can be used to advantage in chaining one controller to another, but typically it's not what you want to do.





Quiz: Custom Tags, JSTL, and Validation

Answer the following questions briefly before discussing with the whole class.

1. Say you write a page with an `<html:form>` in it, and when you test, most or all of your form fields are missing from the page. What would you suspect is the problem? How would you confirm it? How would you fix it?
2. Say you have a sequence of pages 1, 2, and 3, with HTML forms on them; form bean A is used at request scope for the transition between pages 1 and 2, and form bean B is used between 2 and 3; form beans A and B have no overlapping property names. What form bean is used to populate the form fields on page 2?
3. Can you interact with the form bean used by `<html:form>`, for instance in your own JSP expressions and custom tags? What name would you use to get the bean? At what scope would you expect to find it? At what times (or at what points in the JSP) could you access this object?
4. Refactor this Struts custom tag to use JSTL:

```
<logic:empty name="myBean" property="myCollectionProp" >
  <p>No data found!</p>
</logic:empty>
```
5. Say you're using a form bean and Struts HTML tags for form-building. You want to modify a value provided on page 1 before it's echoed to page 2: for instance you want to push all the letters in an email address to lowercase. Where would you make the conversion -- what's your best hook into the process?
6. The Struts Validator is a plug-in to Struts itself. When it's plugged in, application of its XML-declared validation rules are triggered by a call by the ActionServlet to what method on what class?





7. Say you implement validation using the Struts Validator, and when testing you don't see the rules being applied. Where would you look to fix the problem? There are several possible missing pieces ...

8. Say you implement validation using the Struts Validator, and the rules are applied, but you don't see the error messages that you were expecting. Where would you look to fix the problem?



Custom Tags, JSTL, and Validation – Answers

1. Say you write a page with an `<html:form>` in it, and when you test, most or all of your form fields are missing from the page. What would you suspect is the problem? How would you confirm it? How would you fix it?

Most likely you forgot the `<%@ taglib %>` directive. See p.171 of the coursebook. You can diagnose this by looking at the raw HTML source: if it shows the `<html:form>` and `<html:text>`, etc., then it's not being processed on the server side. Add the directive and retest.

2. Say you have a sequence of pages 1, 2, and 3, with HTML forms on them; form bean A is used at request scope for the transition between pages 1 and 2, and form bean B is used between 2 and 3; form beans A and B have no overlapping property names. What form bean is used to populate the form fields on page 2?

Form bean B! This is counter-intuitive, but remember that `<html:form>` always looks ahead. See p.167 of the coursebook. Always ask yourself, what form bean would be involved if the user were to click Submit in this form (that is, on page 2)? It will be bean B, and that same bean is used "in advance" by `<html:form>` to populate the form with any data that might be available. This is potentially useless! but only if the user works forward through the page sequence with no problems. If there's a validation error, then bean B can be useful in feeding information back to page 2.

3. Can you interact with the form bean used by `<html:form>`, for instance in your own JSP expressions and custom tags? What name would you use to get the bean? At what scope would you expect to find it? At what times (or at what points in the JSP) could you access this object?

Semi-trick question, with two answers. The form bean is already published, whether `<html:form>` is involved or not. It has whatever name and scope you've declared for it in your Struts configuration. `<html:form>` publishes an additional reference to the form bean, for use by its child components. The name is defined by `org.apache.struts.Globals.FORM_BEAN_KEY`, which is currently "org.apache.struts.BEAN". It is published at request scope, regardless of the declared scope of the form bean itself. Further, it only exists during the processing of `<html:form>`.

**4. Refactor this Struts custom tag to use JSTL:**

```
<logic:empty name="myBean" property="myCollectionProp" >
  <p>No data found!</p>
</logic:empty>

<c:if test="\${empty myBean.myCollectionProp}" >
  <p>No data found!</p>
</c:if>
```

5. Say you're using a form bean and Struts HTML tags for form-building. You want to modify a value provided on page 1 before it's echoed to page 2: for instance you want to push all the letters in an email address to lowercase. Where would you make the conversion -- what's your best hook into the process?

```
Form bean setEmail (String value) { actualEmailField = toLower (value); }
validate (...) { actualEmailField = toLower (actualEmailField); }
<c:toLower value="\${bean.email}" />
MyAction.execute ()
```





- 6. The Struts Validator is a plug-in to Struts itself. When it's plugged in, application of its XML-declared validation rules are triggered by a call by the ActionServlet to what method on what class?**

In the form bean set method, like:

```
setEmail (String value)
{
    actualEmailField = value.toLowerCase ();
}
```

Or, use a validator:

```
validate (...)
{
    actualEmailField = value.toLowerCase ();
}
```

Maybe don't convert it as application state, but always render it differently in the JSP.

Finally, you might apply the fix in a controller:

```
execute (...)
{
    myBean.setEmail (myBean.getEmail ().toLowerCase ());
}
```

Any of these is fine; the first is probably best.

- 7. Say you implement validation using the Struts Validator, and when testing you don't see the rules being applied. Where would you look to fix the problem? There are several possible missing pieces ...**

Did you add the <plug-in> code to the configuration? Add the validation.xml and validator-rules.xml to the application? Make your action class extend ValidatorForm, or declare one of the dynamic validator form types for your form bean?

- 8. Say you implement validation using the Struts Validator, and the rules are applied, but you don't see the error messages that you were expecting. Where would you look to fix the problem?**

Probably you're missing the string resources for keys that you're using in your validation rules. Be sure there's a <message-resources> element in your config; that the file it points out is actually deployed; and that it includes all the necessary keys.



Revision History

Version 1.3 overhauls the course for Struts 1.3. All code builds to an environment including Java SE 5.0, Tomcat 6, and Struts 1.3 itself. Instructors may want to consult the Struts migration guides (listed in the coursebook's Appendix A) to get a sense of the general types of code changes that were necessary for Struts 1.3. Other major changes:

- Chapter 2 now builds more gradually from the **Housing** and **Ellipsoid** examples through a demonstration using the **Calculator** application – in which students do just a little bit of configuration work to get the application running – and then to the first lab. This lab, still a long one, seemed to be a bit too high a wall for many students to climb without some smaller hands-on work leading to it.
- There is also a new **Flow** example series that illustrates control flow and the sequence in which Struts creates and calls actions, form beans, and (later in the course) commands.
- We still use a MySQL database for the main case study, but this is now nearly transparent. We bundle MySQL 5.0 with the lab software, alleviating the pain usually felt in the classroom as the result of differing independent setups, and database creation is now scripted using Ant. All steps of the **LoveIsBlind** case study now use the database; there is no **LoveIsBlind_XML** but rather a longer **LoveIsBlind** series.
- Discussion of form beans in Chapter 3 is simplified and hopefully more to the point.
- A new case study, **LandUse**, provides some alternate techniques and illustrates a few best practices not applicable in **LoveIsBlind** (or not usable due to the very incremental approach taken in that case study). Instructors may note that this case study is the only one implemented in all of Capstone's Struts, Spring, and JSF courses, and as such it can provide a good apples-to-apples comparison for those students interested in other frameworks.
- With the database configuration mostly transparent, and because `<data-source>`s have been removed from the Struts framework, Chapter 4 on relational data in Struts applications has been removed entirely. This was never a popular chapter! and it is now irrelevant. The **LoveIsBlind** database is set up with minimal fuss in Chapter 2, and we never look back.





- What were Chapters 9 and 10 have been reworked almost totally and are now Chapter 8 on “Advanced Configuration” and Chapter 9 on “Under the Hood” information. Much of the old Chapter 10 seemed to go over like a lead balloon, and in any event later versions of Struts provide many new techniques that will be more interesting to most groups.
- Especially, Chapter 8 brings almost all new material: wildcards, extensions, role-based security, etc. It puts more focus on plug-ins, de-emphasizes subclassing of ActionServlet, and introduces the CoR request processor and command chains.

Version 1.1.3 is a maintenance release focused on simplifying classroom setup and getting the software more in line with Capstone’s courseware standards as they’ve evolved since Version 1.1.2. In particular, most supporting tools are now bundled with the lab image, so that classroom setup involves many fewer steps and is less error prone; also the course now sports an Eclipse WTP 2.0 overlay.

Version 1.1.2 is a maintenance release with minor fixes and enhancements.

Version 1.1.1 fixes various minor defects including typos and minor code glitches, as well as various design enhancements.

Version 1.1 is the initial public release.





Troubleshooting

If you run into any trouble with code exercises, the first and best thing to do is to double-check that the classroom machines have been set up precisely according to the course setup guide. Especially, the wrong version of a tool can cause significant problems; don't wander off-book in this way unless absolutely sure you can support the software that you prefer and that we haven't tested. Check environment variable settings carefully, too; these are the cause of a great many classroom glitches.

Below are some specific pitfalls that have come up in previous offerings of the course:

- If you set the **CC_MODULE** environment variable in an individual command console (and not globally, e.g. through the Windows Control Panel), you must be sure to start Tomcat from that console or from one that also has the environment variable defined correctly. This is necessary because some Java classes running in the deployed web applications will attempt to resolve the value of that variable, as a way of finding resources outside of their own web contexts. A good strategy is to set your environment in the console where you'll do your Ant builds; **start** another console, which inherits that environment, and change to the Tomcat **bin** directory in that second console. Use the second console to **startup** and **shutdown** Tomcat as necessary.





Errata

At this time there are no errata for this version of the course.





Feedback

We very much appreciate whatever feedback we can get on our courseware – especially from the instructor's perspective. Naturally, the more specific, the better, and we strongly encourage you to make notes on issues you may encounter in the classroom, whether they're typos, missing files, or suggestions for clearer language to explain a concept. We can't guarantee that we'll act on every suggestion, but we're aggressive about stamping out problems and try to be highly responsive. Hopefully this means that when you give us good feedback, you get a better course the next time you need to teach it.

Please direct all courseware feedback to

Will Provost
Capstone Courseware
<mailto:provost@capcourse.com>
877-227-2477

For anyone who's interested, we have a very informal defect-tracking system, based in Excel spreadsheets with columns to capture defect location, nature, status, and author feedback. Ultimately, feedback goes into these sheets, so if you want a template, we'll be happy to provide one, to facilitate the reporting process.

