



**Capstone Courseware, LLC**

33 Boylston Street  
Jamaica Plain, MA 02130

877-227-2477  
capstonecourseware.com

# Introduction to ICEfaces

**Version 1.8**

## **Instructor's Guide**



## Overview

Most of the interest in ICEfaces typically gathers around Ajax and UI features. Accordingly, most of this course focuses on those topics. But, to develop a solid understanding of how ICEfaces works, we start with two “backgrounder” chapters: one on custom component architecture and one on Ajax itself.

The risk in this decision is that students will be eager to get started with ICEfaces and impatient with the early material, which includes some demos but no labs. I think it's worthwhile to cover this material first, and it's gone well in classes thus far. That said, if a group is really champing at the bit – or more, obviously, already knows that material – it is perfectly feasible to start right in on Chapter 3, where things get plenty detailed plenty quick and from which point the balance flips to being mostly hands-on exercises without minimal exposition in-between.

Then, Chapter 3 is ostensibly a treatment of architecture and a few cross-cutting features and techniques; even so, it develops as “the Ajax chapter,” with most exercises showing partial submit, partial update, push capabilities, etc. Chapter 4 has a more obvious mission, which is to facilitate study of lots of ICEfaces components and of how to get the most out of ICEfaces in web UI design.





## Timeline

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

### Day 1

1 hour	Chapter 1
1 hour	Chapter 2
3 hours	Chapter 3
2 hours	Chapter 4

Note that this course was initially designed as a follow-on to Course 115 on JSF, and in that configuration it occupies a bit more than a day. It is viable as a standalone one-day, but it's a pretty packed day. Consider covering the first two chapters more lightly to leave lots of time for the latter two. Then it will just be a matter of how JSF-experienced the students are, which will dictate how quickly they can manage the labs.





## Tools Deployed with the Lab Software

This course's software requires separate setups of JDK 6, the Crimson text editor, and/or Eclipse; otherwise it is self-contained, as the lab installer sets up not only the lab software but necessary tools – these will all be found in directories under **c:/Capstone/Tools**:

- **JSF 1.2** reference implementation, in **JSF1.2**. Here you'll find the standard JSF 1.2 RI deployment, plus the 1.2 configuration schema. The schema and older DTDs are with the JAR files in the **lib** directory; the API docs are in **javadocs**. The JSF sample applications are here, too, if you want to take a side trip into showing any of those.
- **JSTL 1.2**, in **JSTL1.2**. The JARs and TLDs for JSTL tags used in many of the course-exercise JSPs are found here.
- **Tomcat 6.0**, in **Tomcat6.0**. This is our target web server for deployment and testing. As mentioned in the coursebook, we need Tomcat 6 to support JSF 1.2 because JSF 1.2 uses the Java-EE-5 Unified EL, and earlier versions of Tomcat don't support that. For this initial release of the course we're using the beta because it's the latest available; we'll migrate to the final when it comes out but we've seen absolutely no issues with the beta in developing the course.
- **Crimson**, a freeware text editor, in **Crimson3.70**. This is made available so that we're sure students have a decent code editor on hand, though they can certainly use their own favorite text editor or IDE. Also, Crimson is primed with support from a second tool, the **XML Validator**, to make a nice little XML editor for working with the **faces-config.xml** file. Hit F9 to save the current file and run an XML parse to check syntax; hit F10 to save and validate against the JSF config schema (though this requires either Internet access or a change to the schema location at the top of the file). Note the mention in the Troubleshooting section about diagnosing and fixing configuration-file syntax and validity errors.



## 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 web application; we won't document this in detail here, but rather summarize the interesting tasks.

In a typical project, Ant will build and deploy a web application to a Tomcat server. For these projects Ant will:

- Undeploy any existing version of the application, by deleting the WAR from Tomcat's **webapps** directory
- Clean out and create a **build** directory, which becomes the root of a web application
- Compile Java source files to **build\classes**
- Create a staging area for assembly of a web application
- Copy resources from **docroot** to that area – this may include HTML and JSP, the **web.xml** and **faces-config.xml** files, message resources, and custom tags – and copy compiled Java classes to its **WEB-INF/classes** folder
- Copy JARs into **WEB-INF\lib**: JSF, JSTL, ICEfaces, etc.
- Build a WAR file from the assembled tree
- Deploy the WAR by copying it to Tomcat's **webapps** directory





## Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse Galileo – also using the ICEfaces plugin. See the course Setup Guide for download URLs and setup instructions. Once installed, the workspace can be found at **c:\Capstone\ICEfaces\Eclipse**.

Instructors, use this package on your own initiative and at your own risk. You should have significant experience yourself with Eclipse web development before using the overlay package in the classroom. The workspace and projects have been tested lightly with the course but are not part of the standard product.

That said, this overlay should save a good deal of work for those who wish to use Eclipse instead of the text editor and command-line tools that are standard for the course. See the documentation inside the workspace itself for general notes on usage: when you first open the workspace you'll see a **ReadMe.txt** that directs you to deeper, HTML-based documentation. 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 on their own.

See also the module notes file in each workspace – e.g. **ICEfaces Module Notes.html** – for specific matters 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.





## Teaching Notes

### Chapter 1

Described in the overview of this document as a “backgrounder,” this chapter does indeed try to provide just a bit of orientation to what JSF custom components are, and how they are used, without getting in too deep. Ideally this chapter will bring quicker understanding of the architecture of ICEfaces when the time comes in Chapter 3. The only example in this chapter is deliberately feature-poor – a custom text field that styles the text uniquely and does some client-side validation – so as to focus attention on the processes and artifacts involved in custom component development and in integrating such components into JSF applications.

If it turns out that there is actually deeper interest in this topic, more adventurous examples can be found in Course 115A, which includes labs on building and using custom components. Please feel free to contact Capstone and request supplemental material drawn from this course.

### Chapter 2

Another so-called backgrounder, this chapter introduces Ajax terminology and key concepts, and for almost its entire length it is independent of ICEfaces. The chapter moves through three parallel examples of Ajax alternatives: “raw” Ajax with XHR, RMI-style Ajax with DWR, and JSF-enabled Ajax with, aha, ICEfaces.

It’s been my experience that even developers with some Ajax experience or savvy will find this chapter interesting – perhaps because Ajax is such a fuzzy term and can imply so many different things. We’re trying here to break down some of the key challenges in building Ajax applications, largely so that we can show in the next chapter how neatly ICEfaces meets those challenges.

One note for reference in later chapters: the HTTPSniffer tool, while generally pretty robust for typical request/response usage, will not function well if you ask it to pass traffic for an ICEfaces application that uses Ajax push. It’s not ready to handle the keep-alive requested by the server and will crash, also causing the application to fail. So, use it sparingly in later chapters!





## Chapter 3

This is the chapter where the rubber hits the road. We start in with ICEfaces more properly now, having seen one quick example at the end of the previous chapter: where to get it, how to set it up, etc. The first demo gives a good sense of the impact of bringing ICEfaces into a JSF project – even at the bare minimum of using a single `<ice:form>` in a single page. Students will often need some amplification at this point of the idea that, after the initial HTTP request, all traffic between the browser and the server is going over the Ajax Bridge. You may want to discuss now, if it didn't come up in the previous chapter, some of the implications of this: how the back button can get goofed up, how URLs are now harder to control, bookmarking, etc.

After the initial demo we start to see much shorter cycles of a few pages of new information followed by examples, demos, or labs. In this chapter there's a lot of exercise in basic JSF/Ajax implementation, and the latter half of the chapter considers the two main techniques for server-side push.

One interesting bug that's latent in the answer to the first lab: you'll notice that the Clear button calls a JavaScript function that clears all the values from the form fields. But, no mention of this is made to the server side of the application! This can pass unnoticed, but try this: type a value in for Department, hit Enter, and see results; click Clear; type the same value, hit Enter ... and see no results. What's happening? When the second request comes in, the results are calculated, but the response is identical to the previous one, and – since it is unaware of any changes since then – ICEfaces optimizes this information out of its partial response. The simple fix is to have Clear invoke an action method on the server side that clears all properties – and this is generally the right JSF practice for such a button. So, if nobody unearths this little gem, you might ask about it when reviewing the lab answer.

Also, a note on the Poll application involved in the second lab: for those with interest in other Ajax strategies, you might point out that the whole Q&A part of the application is actually a pure servlet/JSP approach, using query strings and plain-text responses between servlets and hand-written JavaScript delivered by the JSPs. This is mostly hidden but students have to confront it in this and a later lab, because certain objects have to be shared between the servlets and JSF classes.





## Chapter 4

This chapter holds a lot of new information, but it's also fairly light work to present it, because at this point we're doing exactly the fun sort of stuff that ICEfaces's authors intend: it's just component to component, feature to feature, and mostly a pretty easy process of adding a little markup and seeing whole new UI capabilities coming online. Towards the end of the chapter we confront some of the necessary server-side programming, which is mostly exposing properties as expected by a component (the chart) or handling events fired by the component (table selection, etc.). Some students will embrace this and others will prefer to stay in page-building mode, and most of the exercises make it easy to opt out of the Java coding.

One fairly glaring bug can be seen in the ICEfaces tabbed panes. Try this: when editing a specific proposal, delete some required value, such as the applicant name; click Submit and see a validation error that keeps you in the detail view; now, tab away to comments or decision data, click Submit ... and go right through to the summary view! There is a failure to apply validators to some components, apparently because they are not currently visible. This would be fine, if validation were being applied when we move from one tab to another, but it isn't, and so it becomes possible and even likely that a user would submit bad data.





## Revision History

**Version 1.8** is the initial release of the course.

## 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.

## Errata

At this time there are no errata for 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.

