



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

XML Programming Using Java

Version 1.4

Instructor's Guide



Overview

This course is designed to deliver training in topics that will be most valuable to server-side Java developers who want to incorporate XML into enterprise applications. The course assumes Java programming experience, and some basic XML knowledge, and focuses on combining the two via the JAXP and JAXB. By the end of the course, the student will have occasion to work with APIs from all of the JAXP packages, plus done a bit of work with JAXB, although this latter API is not treated in depth.

The first four chapters of module one should hold few surprises: JAXP parsers, SAX, and DOM readin' and 'ritin'. Chapters 5 and 6 are a little less conventional, but especially 6 on JAXB should be increasingly interesting as binding strategies are more common and reliable now.

The second module ostensibly covers JAXP and how it makes XSLT possible for Java applications, but without a prereq on XPath and XSLT we have to spend most of the module teaching those languages. This does match the technology fairly well, though: JAXP is almost entirely an enabler of XPath and XSLT, and not really a new API for transformations.





Timeline

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

Day 1

XML Parsing, Chapter 1	The Java API for XML Processing (JAXP)
XML Parsing, Chapter 2	The Simple API for XML (SAX)
XML Parsing, Chapter 3	The Document Object Model (DOM) (runs to Day 2)

Day 2

XML Parsing, Chapter 4	Manipulating XML Information with the DOM
XML Parsing, Chapter 5	DOM Level 2 Modules

Day 3

XML Parsing, Chapter 6	XML Serialization and the JAXB
Transformations, Chapter 1	JAXP for Transformations
Transformations, Chapter 2	XPath (runs to Day 4)

Day 4

Transformations, Chapter 3	Templates and Production
Transformations, Chapter 4	Dynamic Content and Flow Control

It is not impossible that a well-prepared group of students could complete this material in three days, but every student would need to have strong Java and XML skills coming in, and everything would need to run just about perfectly. If either of Chapters 5 or 6 in the first module is dropped or shortened (and these chapters won't be of great interest to everyone), a three-day timeline is more realistic, but still aggressive.



The Transformations Console

A few tips on using the primary tool for the second module – the transformations console:

- Convenient keyboard accelerators built into JFC: Ctrl-Page-Up and Ctrl-Page-Down to move between tabs (this is mentioned in the student guide); Tab and Shift-Tab to navigate between controls on a single card (especially useful in the Transform tab); Space to check a radio button, toggle a checkbox or trigger a command button. Note also that all these keystrokes will be consumed by the multi-line text areas when they have the input focus.
- Editing source or transform documents right in the console is meant to be a convenient means to try out simple changes. The JFC text area is not a full-fledged code editor, though, and so for longer instruction sets in labs, use of TextPad or another smart editor is recommended.
- When using an external text editor, it can be very convenient to check the auto-update box in the relevant console tab. This saves the trouble of manually loading changes from the disk file before checking results. Otherwise, the way to load updates to a source or transform document is to set input focus to the filename text field and to hit Enter, as if you were loading the document for the first time.
- It is quite workable to mix and match the two approaches: one can make small changes in the console, click the Save button, open the document elsewhere, save changes, and get the new changes automatically when checking results in the console. TextPad has a nice feature by which it detects that the underlying file has changed and prompts you to load the changes, so the two tools should work well in tandem.





Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse 3.1 for this course. (See the course Setup Guide for download URLs.) Instructors, use this package on your own initiative and at your own risk. You should have experience yourself with Eclipse 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 file `c:\Capstone\XMLParseJava\Eclipse\ReadMe.html` for general notes on how to use the Eclipse overlays for Capstone courses. 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.

For this course, some specific notes:

- Projects in the JAXB chapter will be a bit too much for this basic Eclipse environment, in that they rely on XML-to-Java code generation before the application source files can compile correctly. Ant can coordinate this, but without a great deal of extra configuration for each project, Eclipse cannot. So these projects will show many errors, and will not build successfully. See the **EclipseErrata.html** file for more on this, but the basic idea is that these will need to be built and deployed from the command line.

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

Module 1

This module covers the parsing capabilities of the JAXP and compliant parsers in depth, along with a final chapter that discusses serialization strategies leading up to the JAXB.

Chapter 1

The relevant XML parsing standards are identified, and JAXP is then introduced as (a) a Java implementation of the standards, and (b) a standard of its own, for Java applications and components, that fills in some of the gaps in current SAX and DOM specs and allows completely portable parsing code to be developed. This chapter is mostly overview and architecture. At the end, a simple demonstration provides an opportunity to prove out the environment on student machines, and illustrates the basic use of the JAXP to instantiate and use a SAX parser.

Chapter 2

This one chapter covers SAX in great depth. It is therefore quite long, and will probably occupy a half a day of class time at least. Students now get their hands dirty with a number of parsing demonstrations and lab exercises, and learn the actual parsing API in detail. The event-based nature of SAX may take some describing, but if students have any experience with the Java event model, this should be a simple structure by comparison.

Some of the subtleties of handling qualified names are investigated. Students may or may not have had much exposure to namespaces at this point; Capstone's "Introduction to XML" module, for instance, treats this very lightly. It is not until XML Schema are understood that namespaces can be fully treated, and in this course it is left to student imagination how a document with qualified names can be validated. The instructor may wish to insert a brief discussion of the role of XML Schema in populating namespaces with type information before or during this demo. After the namespace demonstration, the remainder of the module works without much reliance on namespaces, so as to focus on parsing issues.





Chapter 3

The next three chapters cover the DOM: this one focuses on using the DOM to read XML information – that is, parsing proper, largely parallel to the previous chapter on SAX. Students learn to do a lot of the things they knew how to do with SAX, naturally taking a different approach more suited to the DOM tree model as opposed to the event-driven SAX. They also learn DOM-specific features such as traversing an element association using IDs and IDREFs.

It is in this chapter that combinations of multiple SAX and DOM parsers are first treated. For the moment, students will simply run SAX and DOM parsers on the same document, one after the other. This helps to illustrate the different strengths of the two parsing APIs. In the next chapter, passing information from one parser to another using the JAXP Transformer class is covered.

Chapter 4

This chapter reconsiders the DOM as a means of writing XML information: either modifying existing documents and trees, or creating new documents from scratch. This also brings us to the question of how to write an XML document from a DOM tree, and now the JAXP Transformer is brought into play. This also allows study of piping information from a DOM tree to a SAX parsing pass, etc.





Chapter 5

This final chapter on the DOM introduces the modular structure of the recommendation and then focuses on two modules that are implemented by Xerces 2 and are most likely of interest to application developers: Traversal and Mutation Events.

The Traversal module should be most interesting as a combination of the convenience of SAX document-order parsing and DOM read/write capability, as illustrated in the updated Signature application. Although it is not featured as prominently in the exercises, the ability to filter content per iterator is a very powerful feature, and if there is extra time it would be great to explore this further with students.

The Mutation Events module will likely be intuitive for experienced Java programmers, as it mimics the Java event model fairly closely. The basic usefulness of this feature should be evident to students, and the simple Journal exercise illustrates most of the key techniques here.

Chapter 6

Of the many applications for XML in Java programming, use as an object persistence format (or, to turn it around, mapping of XML content to Java objects) is perhaps the one most in need of a generic solution. As such, we've gathered several strategies and techniques for XML serialization and persistence into this intermediate chapter. The chapter is roughly organized to treat SAX and DOM as straw men before introducing the JAXB as the best solution for the general case. This is a significant departure from the rest of the module – JAXB not JAXP, and the code generation and development process that come with JAXB – but with some discussion students should see the two APIs as neatly complementary, and that they solve problems on different levels. The analogy JAXP:JAXB::SAAJ:JAX-RPC might be useful if there is interest in Web services in the group.





Module 2

This module sets up JAXP for managing XSLT-based transformations, and then focuses on knowledge of XPath and XSLT. Only the first chapter is really Java-specific, and it guides students through the basics of using JAXP Transformers.

Chapter 1

This chapter introduces the basic use of the JAXP Transformer class to effect XSLT-based transformations. So far, students have seen this class only as a pipe between SAX and DOM parsers and streams. This makes a good conceptual starting point, and the first exercise implements this same behavior, but reconsiders it not as piping so much as an identity transformation. Then the transformer in use is populated with templates from an XSLT stylesheet, and the full transformation capabilities of the class are suggested. The tool that is built in this chapter is used in the rest of the module as a graphical transformation console.

Chapter 2

This chapter focuses solely on XPath. This is essential for XSLT, of course, and fluency in XPath is helpful in several other key XML technologies. We've tried to develop the expression grammar incrementally, although this is not entirely possible – even the XPath recommendation contains a great deal of forward-reference, and by one author's estimation the 1.0 grammar is fundamentally broken.

It may be best to introduce the UNIX-like syntax of location steps (with abbreviations already in place) as a way of clarifying the purpose of XPath expressions, before moving ahead with the more rigorous `axis::node-test[predicate]` grammar. Then come back to the UNIX syntax when abbreviations are introduced. The lab for this chapter is meant to provide the most immediate feedback possible to students as they experiment with the XPath syntax, letting them see what works and what doesn't. This frankly will be a challenging lab for the instructor, as much invention is left to the student, and some may need a bit more direction and hand-holding. Bringing students along gradually by letting them test out the example expressions during the main lecture should help develop some confidence prior to the lab.





Chapter 3

The first of two chapters on XSLT proper. We've decided to start by drilling down on the production side of the language: built-in template rules, template matching, whitespace control, etc., all of which could be ignored in simple examples, but without which the transform author cannot keep control of the output, leading to a sense that XSLT processing, or the language itself, are somehow imprecise. Because of this focus, the exercises are a little bland; try to clarify for students that once they gain control of production, the next chapter will introduce plenty of more interesting source data and transformation problems.

Chapter 4

Now we look at source-data processing: how to get data for production, and how to control the order of output, which for programmers boils down to flow control. First, though, students get a closer look at how XSLT can be used in real applications. Specifically, they build a servlet that implements the common strategy of mapping URIs to different transformations of XML sources.

Now the exercises get a good deal more interesting, because we're dealing more naturally with persistent data. Once the ability to find and reproduce values and element trees is understood, the chapter turns to "flow control" constructs, including conditionals, loops, and callable templates. The more experienced programmers in a classroom may be tempted to see this as the real language, so to speak, and to assume that general-purpose algorithmic programming can be effected in XSLT. This is of course not the case, and a discussion to this effect might be helpful.





Revision History

Revision 1.4 implements the following changes:

- Supports JAXP 1.3 and JAXB 1.0. No JAXP-1.3 features are introduced to the courseware; this is just re-testing existing code on newer tools.
- We've removed the full JWSDP as a required tool. Instead, we get the four things we needed in smaller bundles: JAXP 1.3 and JAXB 1.0 are delivered with the lab image; Ant and Tomcat are set up separately from their own download sites within apache.org. This results in a simpler and more reliable setup – a couple more steps on paper, but the JWSDP setup was buggy and introduced limitations that could make life hard in the classroom. (For instance once installed it couldn't be relocated without a lot of grief – not true for any of the four pieces mentioned above.)
- This entails the use of different environment variables. No JWSDP_HOME anymore, but rather JAXP_HOME, JAXB_HOME, CATALINA_HOME, and putting Ant in the executable path. (Each of these is documented either in the setup guide or the coursebook itself.)
- The XPath lab (module 1, chapter 2) has been simplified and improved with a list of questions to answer via XPath and a file of correct expressions in an answer directory.
- Various typos and lab glitches were fixed up.
- Code was tested on both J2SE 1.4.2 SDK and JDK 5.0.





Revision 1.2 implements the following changes:

- Supports JAXP 1.2. This is really just a matter of shifting from the Xerces-2 properties for XML Schema validation to the standard JAXP ones.
- There is a new chapter on JAXB. This is an excellent companion to JAXP for general purposes, but also naturally absorbs the discussions of XML serialization and persistence that were previously living at the ends of SAX and DOM chapters.
- Java, XPath and XSLT have been more closely integrated. There is a new Java application, whose completion is the basis of the first chapter's labs, that acts as a testing console for XPath and XSLT exercises in later chapters. In the final chapter, students build a simple Web application that uses JAXP to pipe transformation output to the HTTP response stream.

The course also lengthens to four days, as it was pretty well crammed into three in the previous version, and we've added a chapter or so of content.

Revision 1.0 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.

Some specific issues that often arise:

- Be sure that `JAVA_HOME` and `JWSDP_HOME` have been set correctly.
- A favorite mistake in XPath labs will be to incorrectly set or reset the node-set checkbox in the console application. This will produce two different errors in parsing the resulting XSLT transformation. After a while, students will learn to recognize these responses.
- Beware of setting the auto-update checkbox and then making changes in the console – these changes will always be lost! unless the user clicks Save before checking results.





Errata

Since the latest release of the course, the following issues have been discovered. These will be corrected in the next release.

- Chapter 17, page 505: the correct working directory for this demo is **Demos\JUnit**.





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.

