



Capstone Courseware, LLC

33 Boylston Street  
Jamaica Plain, MA 02130

877-227-2477  
capstonecourseware.com

# Developing Web Services with WebLogic

Will Provost

*Instructor's Guide*

Revision 8.1



## Revision Notes

**Revision 8.1.1** updates only the few parts of the course that rely on the J2EE 1.4 SDK from Sun – the course now supports the 1.4 final, not the beta release. Changes are mostly minor and appear in the following places:

- Chapter 6, at the end where we test our completed SAAJ service with an SAAJ/JAXM client
- Chapter 10, in each of the two places where we consider and test interoperability between two different JAX-RPC products
- Chapter 13, in the beginning of the lab instructions

**Revision 8.1** updates the course to work with WebLogic 8.1. Biggest changes:

- Throughout the course, icons in the upper-right corner of pages highlight differences in behavior between WebLogic 7 and 8.1.
- Chapter 2 on WebLogic Workshop and using the Domain Configuration Wizard are substantially the same, but of course both tools have new GUIs, so the look and in some cases the screen flow is different.
- Chapter 5 on SAAJ has been modified in a number of places, to adapt to the evolving API implementation in WebLogic. Major feature changes are summarized in the chapter. Various pieces of code throughout the course (also notably in Chapter 12 on message handlers) have also been affected by this change.
- Chapters 9 and 10 have many changes to reflect enhancements (and in some cases regressions) in the WebLogic JAX-RPC implementation. There is a new example project called **Polymorphic** to illustrate JAX-RPC polymorphism (and a bug in WebLogic 8.1!). There are new tables laying out the entire WebLogic type mapping system, including JAX-RPC-standard and proprietary mappings.
- Chapter 12 includes treatment of the new `<handlerChain>` Ant task.
- Chapter 15 concludes with a new section on WebLogic's implementation of message security as per WS-Security. There is no hands-on exercise, as this would be much too complex within the scope of this course.
- The final appendix includes a quick table comparing features and bugs in WebLogic versions 7 and 8.1.

**Revision 7.0.2** is the initial public release, for WebLogic 7.0.2.





## Course Overview and Philosophy

This course aims to deliver three related things: knowledge of the interoperable Web services architecture (as captured by the WS-I basic profile), knowledge of the standard Java APIs for Web services, and skills in developing services and clients using WebLogic. All three are important, although not all students will instinctively care about the first or (especially) the second. The author strongly believes in the importance of both standards and commercial tools. Accordingly, the course takes a fundamentally tool-specific path, and makes no compromises in delivering details about using WebLogic practically, but tempers this with thorough treatment of standard APIs, and highlights compliance issues with WebLogic's implementation. Instructors are encouraged to carry this philosophy into the classroom.

The course starts with overviews of interoperable and Java architectures for Web services, with an introduction of WebLogic Workshop and Server in-between. The subject is quite large, and so this overview takes nearly a day and can overwhelm students a bit – it's a good idea to take it in small chunks and encourage questions and discussion.

Three more chapters introduce SOAP, SAAJ, and what in class I often call "low-level" Web services – ones whose application code deals directly with SOAP content, element by element. This is of course not where the real heat is in the development market right now, but it's worth covering in depth for two reasons. One is that not all services (or intermediaries, or service-development tools, etc.) can benefit from the "high-level" approach of JAX-RPC, and at some point developers will have to work at the low level with SAAJ. Finally, even JAX-RPC requires the use of SAAJ for some things, notably message headers. So keep students' eyes on the bigger prize of high-level services and rapid development using JAX-RPC, but don't skimp on these chapters!

The center of the course is certainly the next four chapters, on WSDL, JAX-RPC and developing services and clients in each direction – from Java or from WSDL. Students should clearly see this as the big payoff for all the lead-up study of architecture, SOAP, and SAAJ, and accordingly there's a lot of hands-on work. There's also a lot more digging to do, from easy development using JAX-RPC code generation down into how SOAP messages are shaped and how various Java and WSDL names can be controlled in the build. Interoperability becomes an interesting question at this point, and the Java-to-WSDL and WSDL-to-Java services built in labs make for interesting interop studies between WebLogic and the J2EE RI.

The rest of the course delves into various intermediate and advanced topics surrounding JAX-RPC: EJB endpoints, JAX-RPC handlers for SOAP header content, message attachments (using SAAJ), asynchronous services using JMS, and Web-services security. At this point there is not so much of a linear flow, but rather each chapter is a radial expansion from the core of the course, which is JAX-RPC itself.





## Timeline

### Day 1

Chapter 1	The Web Services Architecture
Chapter 2	Web Services and WebLogic
Chapter 3	The Java Web Services Architecture

### Day 2

Chapter 4	SOAP
Chapter 5	SAAJ
Chapter 6	SAAJ Web Services

### Day 3

Chapter 7	WSDL
Chapter 8	JAX-RPC
Chapter 9	Generating Web Services from Java Code

### Day 4

Chapter 10	Generating Java Web Services from WSDL
Chapter 11	Web Services and EJB
Chapter 12	Message Context and Message Handlers

### Day 5

Chapter 13	SOAP Attachments
Chapter 14	Web Services and JMS
Chapter 15	Security

In fact, you'll usually find that at the end of each day the class is part-way into the first chapter for the next day: into Chapter 4 at the end of Day 1, Chapter 7 at the end of Day 2, etc. This is because the first few chapters have little hands-on work – no labs, just demos – and gives a comfortable pace for the rest of the week. It's not uncommon for Day 5 to start with Chapter 14 and finish early, so if the class needs extra lab time, especially in the middle days on JAX-RPC, don't hesitate to allow this.

Also, note that any of the final three chapters can be skipped or moved into a different order. Really, none of the last five chapters depend on each other, although 11 and 12 are more strongly recommended than the final three.





## Developing Web Services with WebLogic

### Chapter 1

First, the chapter is meant to put the Web-services buzz in broader context. “Why do we need these things, anyway?” should be the question that the instructor is implicitly answering for the first ten pages or so. At the end of the chapter, as well, there is an attempt to step back from the SOAP hype of the moment and note that there are non-SOAP approaches, and that there are just a few fundamental things that one can say about all Web services.

The big middle of the chapter, though, walks through the consensus architecture, which is of course SOAP-based: the interoperability stacks as laid out by Microsoft and IBM are discussed, and generally it's a good idea to note how little of the architecture has really found a consensus technology to support it: SOAP, WSDL, and UDDI account for a fairly small part of the bigger picture of Web-based business software integration.

It's a good idea to instruct students to start the WebLogic Workshop and its example server before breaking between chapters, as server startup takes several minutes.

### Chapter 2

Here we introduce the BEA tools for Web services, with a brief demo of the Workshop and some introduction and setup work for using the Server. After creating a domain just for their in-class work, students build and observe the operation of a Web service and client, and use the SOAP Sniffer distributed with the course software to observe the SOAP traffic between the two. A lot of this chapter is setup for the rest of the course's labs, plus the first opportunity to see Web services in action. No need to go deep on what WLS does and doesn't do at this point; this will be highlighted as the course progresses through various standard APIs.

### Chapter 3

This chapter starts from scratch, in a way, working through the Web-services problem now with the assumption that Java technology will be used for implementation. From this point forward, it will be important to keep the student's eye on the dualities of JAXM+SAAJ/SOAP, JAX-RPC/WSDL, etc., and to highlight the mappings from Java code to SOAP messages and WSDL content. (This chapter mentions JAXR and UDDI, but this is the last significant inclusion of discovery in the course – the rest of the chapters focus on development using SOAP, WSDL, and Java APIs.)





## Chapter 4

This chapter introduces SOAP 1.1 “from scratch.” The protocol is presented in some detail: SOAP namespaces, envelope, header and body, etc., plus a little refresher on using namespaces in XML. To enable students to learn to read and write raw SOAP messages, the SOAPPad application is provided. (This application and several others are good general-purpose tools for studying SOAP-based services: you may want to demonstrate the four listed in the student guide during this lecture, and also suggest at this point that students create a desktop shortcut to SOAPPad, since they’ll use it extensively in testing various labs.) The concept of message validation is introduced here, as it is not Java-specific, and a simple JAXP-based tool is shown to validate various SOAP messages stored in files. A few bits of the SOAP Section 5 encoding are also mentioned here, since these will be encountered in messages generated by JAX-RPC tools later: arrays and multiple-reference values, specifically.

## Chapter 5

Now the course moves to the question of how to read and write SOAP messages using Java. There is a certain amount of rather dry coverage of the many SAAJ interfaces; the flow through these is directed first by a UML diagram that provides an overview of the compositional model (message to part to envelope to header, etc.), and then another UML diagram showing inheritance relationships between the various Node types. Labs are simple and don’t involve any actual Web-service implementation – instead message content is read from and written to files – but the code built in this chapter will be fed into service code in the next.

## Chapter 6

Since WebLogic is weak on support for SAAJ in general, and low-level services (a la JAXM) in particular, this chapter straddles WebLogic and the J2EE RI to show the service side and client side, respectively. The tension between standard APIs and WebLogic implementation will be an important issue throughout the course, and it is most obvious here. It will be important to direct attention both to the standard, JAXM, and to WebLogic’s alternative approaches to low-level services and, later, asynchronous messaging using JMS instead of JAXM.





## Chapter 7

We jump back out of the Java realm now to study WSDL. The beginning of the chapter develops some motivation for use and study of WSDL, especially by comparison to major DOC infrastructures, each of which features an IDL of some sort. Then the language is developed component-by-component, focusing first on the abstract model and then on the concrete model of services, ports and bindings, with a lab for each. The labs are the only hands-on work in writing WSDL descriptors, and should be completed by all students if at all possible. (By contrast, learning to write SOAP by hand is not so critical – there it's just a matter of getting familiar with syntax, writing as reinforcement to reading, really. Skill in writing WSDL descriptors is of real, practical value.)

## Chapter 8

This chapter brings Java back into the picture in the form of the JAX-RPC specification. We take a “how does it work?” approach to most of this chapter, seeing how a minimal body of WSDL and/or Java code can be parleyed into a fully-functional Web service or client. Where the previous chapter explicitly compares Web services to COM, CORBA, and EJB, in this chapter we're a little more detail-oriented. Still, it may be interesting and help tie the lecture together to compare JAX-RPC to RMI and EJB, especially if students have any exposure to the latter technology.

The discussion near the end of the chapter can be as limited or as free-ranging as the instructor and students prefer. This is included here rather than later, as the structure of the course positions the latter two chapters as the more nuts-and-bolts treatments of each of the two main development paths. Thus this is the last “central” JAX-RPC chapter, and this discussion seems to flow well from the high-level conceptual tour of the JAX-RPC architecture and process, even though students don't have hands-on experience with JAX-RPC yet. The instructor is encouraged to explore other best-practice issues with the class (and to share his or her thoughts with us here at Capstone Courseware, as well – we'd love to see this section evolve with new ideas and opinions). It's also possible simply to defer coverage of these sorts of issues for an impromptu discussion after Chapter 10.





## Chapter 9

This and the following chapter are lighter on lecture and heavier on lab time. Here we start to look at the details of Java-to-XML/WSDL mapping, and also address the common development problem of converting or enhancing an existing Web application to provide SOAP-based Web service. (A working title for the chapter was “Adding a SOAP Interface to Java Code.”) This is of practical interest to many developers and also makes it easy to concentrate on what’s distinct to Web-service development by assuming that domain and persistence code exist (and are well-organized and correctly decoupled).

Lab 9 is meant to be a fairly in-depth exercise not only in mapping/generating from Java, but in the sorts of domain changes or adaptations one often must perform in adding SOAP service to a codebase. This is hopefully of some interest to the students, but it does involve a good bit of low-level and error-prone coding. We’ve tried to minimize the risk of going off the tracks by providing complete new source files in some cases. The lab also illustrates exactly the issue of declared vs. runtime type (polymorphism) raised in the few pages prior to the lab.

During this chapter, you’ll likely see answers to the question of “Which Way To Go?” introduced in the previous chapter start to gain adherents. Especially, students may wonder why there’s a WSDL-to-Java path at all. Encourage these questions and this discussion, but urge everyone to keep an open mind, as the next chapter does implicitly make some of the case for the alternative approach.

## Chapter 10

In this chapter students will look at the reverse direction from Chapter 9. Again, we walk through the mappings in detail, this time considering issues specific to generation from WSDL. It will be good to emphasize that WSDL-to-Java can be used to create services and not just clients. Although this approach will be of less practical use in the short term (compared, say, to converting existing Web applications by generating from Java code), in the abstract it is preferable and in the long term it will be a more common scenario. The arguments to this effect have already been made in Chapter 8, but some of these are emphasized in this chapter, now that more details about the mappings and process have been explained: client-side validation, strong typing in XML Schema, etc.

See also the article, “WSDL First!” also by the author of this course, at:

<http://webservices.xml.com/pub/a/ws/2003/07/22/wsdfirst.html>

You may want to print and distribute this during class, especially if students seem engaged by the deeper questions of “Which Way to Go?”





## Chapter 11

This chapter assumes a lot, really: knowledge of JAX-RPC and servlet-endpoint development as well as EJB concepts. It is fine if students are weak on EJB, as the chapter begins with a little explainer (and the instructor may want to go into more depth if time permits). From this basis, the chapter explains the merging of JAX-RPC and EJB practices, and shows the EJB Web-service development process as similar to that for servlet endpoints. This is another area where there is a disconnect between standards (EJB 2.1, not out yet as of this writing) and WebLogic, and so a few points in the chapter have to be made twice, essentially. The basic idea is simple enough, though, and consistent between EJB 2.1 and WebLogic 7.

Beyond this, the chapter focuses on important limitations on Web-service endpoint interfaces under EJB. What students are really doing is learning JAX-RPC as RMI/SOAP, where they may be familiar with RMI/IIOP, and it's good to highlight the major impacts on interface design and component interactions that result from JAX-RPC's simpler type model.

See the Troubleshooting and Tips section for a number of pitfalls in this chapter's exercises. These bring in a few new and demanding tools, including EJB itself and also relational databases via PointBase. It's a very good idea to go through both the Shareware and Love Is Blind exercises for this chapter before leading students through them, to be certain you have the steps down, and know how to clean up each project for a clean WLS startup later, as well.

## Chapter 12

This chapter gives students a look at JAX-RPC's means of managing SOAP headers in requests and responses. The UML diagrams are meant to establish a theme, which is that from three disparate starting points – servlet endpoint class, EJB endpoint class, message handler – the developer can navigate to the same SOAP message context object, and from there can manage and share information, about the message and about how it is to be processed. If time permits, the instructor may want to expand on the fairly simple exercise in Lab 12 by introducing the possibility of faults – this complicates the possible message paths between handlers and endpoint considerably, and is of course a practical consideration.





## Chapter 13

This chapter is perhaps the most cookbook-style of the module. The idea of attachments is fundamentally simple, and adds an important dimension to SOAP messaging for Java applications. However, there is a lot of supporting technology, notably the JAF, into which there isn't nearly enough time to delve deeply. So, without a lot of embroidery, we plough through the relevant SAAJ classes, with a quick intro of the JAF.

It's a good idea to mention the (relatively incomplete and poorly implemented) support for attachments via JAX-RPC, even though it isn't a practical alternative at this point.

## Chapter 14

This chapter completes the tour of possible forms for Web service endpoint implementations in WebLogic. That is, we've seen plain Java classes, and we've seen EJBs; now we see JMS queues as endpoints for a specific style of messaging which is not the synchronous request/response we've been using all along. The chapter is written to emphasize the complete decoupling of application code and SOAP URI effected by a JMS message destination, and the main demo proves this point by dropping the application code entirely; the JMS queue is the Web service. This is an elegant combination of technologies.

You'll probably notice (and so will students) a warning in the WLS console when deploying SOAPQueue. I have yet to track down the exact reason for this warning, but I expect that it has to do with the absence of any proper Web-service or even J2EE components in the application – hence WLS can't attach monitors. The warning is harmless in the scope of the demo.

One little trick that the instructor can play in this chapter is enabled by a pre-built Web service deployed in **Examples\Auction\Step3**. Nothing in the student guide points students to this application, but as the instructor you might want to drop this into the Capstone domain's application directory at the beginning of Lab 14. A common mistake for students is to leave their SOAPPad host field set to the name of the instructor machine at the end of the SOAPQueue demo. Then, they'll get strange results while testing their Lab 14 solutions – either no response or (worse) successful messaging but the wrong results based on shared use of the instructor machine, if the instructor has built the Auction answer project. Instead, deploy **RefuseBids.ear** from the above directory. This application is a single servlet that will respond to any requests under the /Auction/Bid or /Auction/Results URIs with an HTML error page saying, "Are you sure you set SOAPPad back to LOCALHOST?" This has the advantage of saving students some time during their lab work, and will also be a nice little parlor trick, as they will at first wonder how this message appears.





## Chapter 15

This final chapter on Web-service security only scratches the surface of what will become a massive topic. It provides an overview of the sorts of issues Web services will face, and of the areas of technology from which solutions will emerge: mostly this means J2EE (via the JCP) and XML/SOAP (via the W3C, OASIS, the WS-I, etc.). Most of the real practice is in securing URLs – a traditional Web-application technique – but the lab gives students a peek at the sorts of message security that they might need to implement in the future – albeit with tools a bit stronger than the **SecretDecoderRing** class.

Either before or after the final lab (the book places it before), you can dive into the new section on WS-Security for secure messaging as implemented by WebLogic 8.1. This topic is moderately complex in theory, but extremely complex and error-prone in practice, which is why there are no hands-on exercises in this new feature. The overview given in this part of the chapter should be enough to explain clearly what WS-Security is doing, why, and how, and that's about all you're likely to have time to do. (Full treatment of secure messaging, with coding exercises, will likely be part of a later master class in Web service design and development.)





## Troubleshooting and Tool Tips

First, be sure to be familiar with the lab and environment setup instructions in the Table of Contents. Assure that the instructions in the module Setup Guide have been followed, and that you know the locations of the installed tools. If, having followed those steps, a classroom machine is failing to run demos or lab answers correctly, here is a list of items to consider and to doublecheck:

- Make sure the `JAVA_HOME`, `J2EE_HOME`, `JAXM_HOME`, and `WL_HOME` variables are correctly set.
- Make sure that the **bin** directories under all three variables are in the executable path.
- Go over the instructions in Chapter 2 very carefully with each student, and confirm that they have the right settings for their WLS domain (“mydomain”, “myserver”, etc.); that they have added **webservices.jar** to the **setWLSEnv** script; etc.
- Probably the most common mistake students will make is to forget to run **setWLSEnv** in a newly-opened console. This will usually manifest itself by the failure of the compiler to find certain types, such as those from SAAJ or JAX-RPC.
- Conversely, be sure everyone runs the handful of J2EE-RI exercises in a new console in which **setWLSEnv** hasn't been run.
- When starting the Sun server – which you won't need to do until Chapter 10 – be sure to give it a couple minutes to “warm up.” The complete startup sequence can take several minutes, and until this is complete the server can give some strange behavior. In particular, on deployment, you may see exception stacks in Ant's output that reference “invocation target” exceptions. Sadly, once you've gotten one of these, you will usually keep getting them until you restart J2EE (and wait a little longer).
- The second most common error will be mistakenly selecting text in a DOS console with the mouse, when the student only meant to bring the console window to the front for viewing. DOS boxes under Windows 2000 will freeze when text is selected. If the WLS or Pointbase console is frozen in this way, processing of remote messages will be suspended as well, resulting in multi-process lockup. Get in the habit of hitting the ESC key before leaving a highlighted server console.





- A few troubleshooting items for Chapter 11 exercises in particular:
  - If the server can't find "Targets" equal to "myserver", the student very likely failed to change the default server name to "CapstoneServer" when creating the Capstone domain way back in Chapter 2. This error won't cause any trouble until now. Workaround is to change the XML pasted into **config.xml** to use "myserver" instead of the prepared "CapstoneServer".
  - Multiple instances of WLS or PointBase will cause problems. Be sure students aren't starting up more than one.
  - It is critical that students disconnect from the database before closing the PointBase console. Otherwise the connection will be retained for a while in PointBase, and this in turn will lock out WLS when it tries to create its connection pool, because by default only a single connection is allowed in PointBase DBs. If one forgets to do this, one can always restart the console, log in, and then disconnect.
  - Another common mistake will be to fail to set the password for the PBPUBLIC user. This will cause a failure to connect from WLS. Confirm by trying to log in using the PB console; if the password is still "" then perform the steps in the demo to alter the PBPUBLIC user; disconnect; close the console; and restart WLS.





## Feedback

We truly do welcome feedback, both of a specific nature (pointing out mistakes) and general suggestions. For the former sending email with a numbered list of corrections would be most helpful.

Please send feedback to:

Will Provost  
Capstone Courseware  
<mailto:provost@capstonecourseware.com>  
[www.capstonecourseware.com](http://www.capstonecourseware.com)

