# Practical Apache Struts2 Web 2.0 Projects



Ian Roughley

Apress®

**Practical Apache Struts2 Web 2.0 Projects**



ISBN-13 (pbk): 978-1-59059-903-7

ISBN-10 (pbk): 1-59059-903-9

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1



Lead Editor: Steve Anglin
Technical Reviewer: Frank Zammetti
Editorial Board: Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Jason Gilmore, Kevin Goff, Jonathan Hassell, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Candace English
Copy Editor: Julie McNamee
Associate Production Director: Kari Brooks-Copony
Production Editor: Candace English
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: Lisa Hamilton
Indexer: Broccoli Information Management
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.



The source code for this book is available to readers at `http://www.apress.com`.

*For Skooter.*

# Contents at a Glance

# Contents

# Foreword

Apache Struts is one of the most successful open source projects ever created. With the exception of "infrastructure" projects such as Linux, MySQL, and various programming languages, few other open source frameworks have managed to have the success, popularity, market dominance, and ability to change the way developers think as Struts has.

As one of the creators of the original Struts 2.0 codebase, I am overwhelmed with pride and joy to see so many people contribute and use the project. With literally hundreds of thousands of projects developed on top of Struts, and countless more developers experienced with it, the decision to update Struts from version 1.x to 2.x was not a trivial one. And yet through the experience and leadership of the Struts team, the new 2.x version, which this book is about, has been met with wonderful reception among the developer community.

Ian Roughly is a good friend of mine: Over the past 4+ years, he and I both dedicated far too much time on WebWork, the project that merged with Struts and became the foundation for Struts 2.0. Although Ian is not an original Struts developer—in fact, we both got involved with WebWork because, ironically, we didn't feel Struts 1.x was exactly what we needed—he is definitely one of the most qualified people to write a book about Struts.

With a next generation of Struts gaining momentum among classic Struts users as well as new ones, the time is right for a book on this updated, modern technology. Whether you want to learn about AJAX integration, plug-in-oriented development, or just how to build quality web apps, I can think of no one better than Ian to be your guide.

I am certain you will enjoy this book. It's about a great technology, and it's written by an expert who not only created much of this technology but also uses it on a daily basis in his own practice. Ian's words and advice come from real experience—he's not some disconnected architect who doesn't actually write web apps anymore. He's the real deal. He knows what it takes to build quality web applications, all the way from setting up a build system that works well for web development teams, to building complex wizards and workflows, to properly securing your application in a more complicated world dominated by AJAX.

You are in good hands, both in terms of your guide as well as a technology choice. Struts is an evolving framework for building modern web applications, and I encourage you to join the community after you are done with this book so that you may continue to participate in the evolution and be part of one of the most interesting Java web frameworks today.

Enjoy the book!

Patrick Lightbody
*Co-creator, Struts 2.0*

# About the Author

**IAN ROUGHLEY** is a speaker, author, and consultant based in Boston, MA, where he runs From Down & Around, Inc., a consultancy specializing in architecture, development, and process improvement services. For more than 10 years, he has been helping clients ranging in size from Fortune 10 companies to start-ups.

Focused on a pragmatic and results-based approach, he is a proponent for open source, as well as process and quality improvements through agile development techniques. Ian is a committer on the XWork and WebWork projects; member of the Apache Struts PMC; and speaks at conferences in the United States and abroad. He is also a Sun Certified Java Programmer and J2EE Enterprise Architect and an IBM Certified Solutions Architect.

You can reach Ian at `ian@fdar.com`, or via the web at `http://www.fdar.com`.

# About the Technical Reviewer

**FRANK W. ZAMMETTI** is a web architect/developer for a worldwide financial company by day and a jack-of-all-trades by night. Frank has authored a number of books and articles on topics ranging from AJAX to DataVision. Frank is an active participant in a variety of open source projects both small and large; some he leads, and a few he has founded himself. Frank has been involved with computers, in one form another, for more than 25 years, 15 of that being "professional," which just means he was being paid to pretend he knew what he was doing! Frank is an avid indoorsman, shunning the sun like the uncle no one talks about. Frank lives in the United States with his wife, two children who never stop talking, and an assortment of voices in his head that won't stop singing the theme songs from '80s television shows.

# Acknowledgments

It has been a remarkable experience being involved with open source development, what I believe to be the real "beta" of Web 2.0 sharing and collaboration. Where else can you combine talented individuals from around the world, without significant management, to produce a product that hundreds of thousands of companies depend upon every day? I'd like to thank everyone involved in the XWork, WebWork, and Apache Struts projects; without their tireless commitment and contributions, I would have nothing to write about. In particular I'd like to thank Don Brown, Patrick Lightbody, Philip Luppens, Rainer Hermanns, and Rene Gielen; they have always been there when I had a particularly tricky question that needed answering.

I would like to thank Steve Anglin, Candace English, and Julie McNamee from Apress, as well as all of the people behind the scenes that I haven't had the opportunity to meet personally. Without your ongoing support and assistance, this book would not have been possible.

I'd also like to thank Frank Zammetti, my technical reviewer and Struts2 community member, for keeping me on my toes, always questioning, and always making sure that the information presented was at its very best.

Finally, I would like to thank my remarkable wife LeAnn. Her continuing support and ongoing review and nongeek analysis of the manuscript has been invaluable.

# Introduction

Web application development has been around for a long time. In fact, it has been around long enough that a new term, Web 2.0, is being used to describe the next generation of web applications. Web 2.0 is an intersection of new business models, new ideas, and multifaceted sharing and collaboration—with iterative development techniques getting new features to users at a much faster pace. Along with Web 2.0 came a revival of scripting languages (and even a few new ones), all dynamic and supporting fast-paced and highly productive development environments.

Around the same time, Struts (the first, and most popular Java web application framework ever) was reaching an important milestone—its second major release. This was not only an important milestone for the framework in terms of functionality but also for the improvements made to increase developer productivity. By decreasing coupling within the framework, reducing configuration, providing default and different configuration options (via annotations), and providing a plug-in mechanism to easily extend the base features, Struts2 is providing a platform that can be built upon for the next generation of web applications. With these new enhancements, Struts2 is poised to compete as the development framework of choice for Web 2.0 applications.

To use a new framework, you first have to know the features that are available, and learning a new technology from scratch using reference manuals and disconnected examples can be difficult. In writing this book, my goal was to provide the information to you, on how to develop a Web 2.0 application using Apache Struts2 in a practical and hands-on manner. You will achieve this goal by understanding the architecture of Struts2, by knowing the features that Struts2 provides, by seeing how these features are used, and by using and further exploring each of the features through the code provided. Each chapter builds on the last, providing more and more information until a complete web application emerges.

Time to get started!

CHAPTER 1

# Web 2.0 and Struts2

Before charging forward with developing a Web 2.0 application, you need to understand what a Web 2.0 application really is. In this chapter, you will learn what Web 2.0 means from a development as well as end user perspective.

With Struts2 being the technology of choice, you will also learn how Struts2 provides the features to make developing a Web 2.0 application easy.

## What Is Web 2.0?

One of the questions that needs to be answered before embarking on developing a Web 2.0 application is "What is Web 2.0?" As it turns out, this is a particularly difficult question to answer.

From a programming perspective, Web 2.0 is synonymous with AJAX (Asynchronous JavaScript and XML). The term AJAX was coined in February 2005 by Jesse James Garrett and is used to describe the interaction between many technologies. At the core is the `XMLHttpRequest` object, which is supplied by the web browser. This object was first present in Microsoft Internet Explorer 5 (released in March 1999), although similar techniques using `IFRAMES` and `LAYER` elements have been available since 1996.

Along with the `XMLHttpRequest` object, the technologies that make up an AJAX interaction are the following:

- *HTML/XHTML (Hypertext Markup Language)*: Used to present information to the user from within the web browser.
- *DOM (Document Object Model)*: The object structure of the HTML document in the web browser. By manipulating the DOM with JavaScript, the page rendered to the user can be modified dynamically without reloading the current page.
- *CSS (Cascading Style Sheets)*: Used to format and style the HTML presented. By separating formatting from structure, the code can be modified consistently and maintained more easily. Similarly to the DOM, CSS for the current page can be modified via JavaScript to dynamically change the formatting without reloading the current page.

- *JavaScript*: A programming language that can be embedded within HTML documents. JavaScript code or functions can be executed inline (as the page is processed), in response to HTML events (by providing JavaScript in the value of HTML attributes), or triggered by browser events (for example, timers or user events).
- *XML (eXtensible Markup Language)*: The format of the data returned by the server in response to the asynchronous call from the web browser. The XML response returned is processed by JavaScript in the web browser, causing changes in the HTML (by manipulating the DOM or CSS).

Recently, another data format has been gaining popularity: JSON (JavaScript Object Notation). Similar to XML, JSON returns data that can be processed within the web browser using JavaScript. The advantage of JSON is that it can be very easily parsed by JavaScript; in fact, to convert a response of any size from the JSON transport format to JavaScript objects involves a single call of `eval('('+responseJSON+')')` (where `responseJSON` is the JSON data represented as text or a string). Using JavaScript to process XML is much more involved and requires at least one line of code to assign a value from the XML document to a JavaScript object.

### EVALUATING VS. PARSING

There is a security concern when calling `eval()` on a JSON string, especially when the JSON is obtained from a source external to the code currently being executed. The problem lies in the fact that the `eval()` function compiles and executes any JavaScript code in the text string being parsed to create the object representation. For this reason, you need to be sure that you trust the source of the JSON text. Even better still, you can use a JSON parser, which avoids the problems associated with the `eval()` function.

One such parser can be found at `http://www.json.org/json.js` (the web site `http://www.json.org` is the gateway to all things JSON). When using this JavaScript script, additional methods are added to the basic JavaScript objects to both generate JSON and parse JSON. When provided with a JSON string to be parsed (say `jsonText`), the following code is used:

```
jsonText.parseJSON(filter);
```

The parameter `filter` is an optional JavaScript function, which can be used to further filter and transform the result. To generate JSON, use the `toJSONString()` method. For example, to convert a boolean `myBoolean`, use the following:

```
myBoolean.toJSONString();
```

By using a JavaScript JSON parser, the JSON text can be converted just as simply but without security concerns.

By using AJAX interactions, developers can make the user experience less awkward. Rather than requiring the entire HTML page to be reloaded from the server (along with processing the request on the server) and rerendered to update values in a drop-down selection box, now a smaller request to the server can be made. More importantly, the page is not rerendered; instead, the only change to the HTML is that the values for the drop-down selection box have now been changed.

Smaller and more targeted information requests to the server means that the time spent waiting for the network and server processing will be less. Not having to rerender the entire browser page on each server request will also be perceived as the web application performing faster. With these pieces working together in an AJAX interaction, the web browser will become more responsive and act more like a traditional desktop application—increasing the usability and overall user experience.

It also means that developers need to think differently. In fact, developers need to reexamine the fundamental way that a web application is constructed; rather than thinking of a page as a unit of work, they need to think of functionality from a page as being the unit of work, with many functions being combined to create the final page. Furthermore, the same functionality can now be easily shared among pages.

### THE PAVLOV EFFECT

Changing the user interaction (even for the better) has its own problems. Users have been trained to understand that nothing on HTML pages changes until you click a link or a form submit button. All of a sudden, things are different. Now, at any time, any part of the HTML page has the potential of being updated or removed, and new information can be added.

To help transition users to the new browser interaction model, as well as to provide developers with guidelines of when and how to use AJAX in web applications, a series of patterns has emerged. AJAX patterns cover a wide range of topics, including how to signal the user that a UI element has changed; when to make server calls to obtain data; options for introducing AJAX into non-AJAX web applications; how to manage technical aspects such as server timeouts; and ways to provide a non-AJAX fall-back when JavaScript is not available on the user's browser.

From a marketing or end-user perspective, things are a little different. There is no doubt that more interactive user interfaces can make the overall web application's usability better, however, the shift from Web 1.0 to Web 2.0 is more than user interfaces.

In September 2005, Tim O'Reilly published an article titled "What Is Web 2.0" (`http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`). This article explored what Web 2.0 was by comparing a new breed of web sites that were available to those that had been around for some time. The result was that no hard boundaries of principles or technologies signified an application as a Web 2.0 application. Instead, there were guiding principles that, when adopted, resulted in a web application that is more Web 2.0 than when the principles were not used. Following is the list of proposed principles:

- *Web as a platform*: Applications should take advantage of the Web as a platform rather than simply providing a presence on the Web. By working symbiotically with the openness and connectedness of the Web, services can reach out to all users. And in doing so, will get better as more people use the service.
- *Harness collective intelligence*: Hyperlinking has been the foundation of the Web, allowing users to explore related content, but it has always been provided by the web site being visited. The new breed of applications takes this a step further, allowing users to provide information to the web application in the form of content and metadata (information about the content, such as ranking or popularity). Individual publishing, via blogging, has also become popular, allowing anyone to become a publisher of information and opinions.
- *Data is the next "Intel inside"*: Originally, companies owned and provided data to users of their application. Although an initial set of data is still this way, a new and much more valuable set of data is being provided by users of the application. Now the race is on for companies to own a particular category of user-provided data, leading to the question, "who owns the data?"
- *End of the software release cycle*: With software being delivered as a service rather than a product, all users of a web application can take advantage of new features being provided immediately. In a sense, users then become codevelopers and can be monitored to determine which features are used, and how often—shaping the features of the final product. Releasing often also requires operations to become a core competency.
- *Lightweight programming models*: There is a preference to use simple protocols, such as REST (Representation State Transfer) rather than SOAP (Simple Object Access Protocol), and RSS (Really Simple Syndication) to provide syndication and remixability. The innovation is that combining many loosely coupled services together in a unique or novel manner provides value in the assembly.
- *Software above the level of a single device*: In a connected world, the browser is no longer the single device of choice. Web applications will need to interact with devices accessing them from web browsers and cell phones, as well as more specialized devices such as iPods, PDAs, and cable boxes.
- *Rich user experiences*: Services are becoming RIAs (Rich Internet Applications), providing dynamic and responsive interactions with users. AJAX (which was explained earlier) as well as Java applets and proprietary solutions such as Macromedia Flash, are all enabling technologies.

Almost one year later in August 2006, Tim O'Reilly gathered a group of people together to build on his initial paper. Gregor Hohpe was one of those people invited, and he blogged (`http://www.eaipatterns.com/ramblings/45_web20.html`) about the values, principles, and patterns that were discussed.

As an agile developer, the style the values were presented in hit an accord. Using the same format as the Agile Manifesto, it presented the differences between a Web 1.0 and Web 2.0 application as a range. The closer the application is represented by the descriptions on the left, the more Web 2.0 the web application is. In the end, whether an application is Web 1.0 or

Web 2.0 is still subjective, but grading the level of Web 2.0-ness is easier. The values, with my interpretation, are provided here:

- *Simplicity over Completeness*: Application features do not need to be absolutely complete, having every variation and every option possible. Instead, the most used options are all that is required, making the application much simpler to use and quicker to market.
- *Long Tail over Mass Audience*: Business models are focusing on selling smaller volumes of a large variety of hard-to-find or unique items rather than selling large volumes of a small number of popular items. The same can be said about knowledge (see the Wikipedia entry for more information on the Long Tail `http://en.wikipedia.org/wiki/The_Long_Tail`).
- *Share over Protect*: Web sites are no longer gated enclosures; instead, information and services are shared using techniques such as web services and feeds.
- *Advertise over Subscribe*: The preferred revenue model for Web 2.0 sites is advertisement rather than subscription (although of all the values, this is the one that is most controversial because as applications move from products to Web 2.0 services, a subscription model will be required).
- *Syndication over Stickiness*: An early goal of web applications was to keep users on the site for as long as possible. By providing services, the information that could only reach users on the site can now have a much farther reach by syndication (with links leading them back to the application).
- *Early Availability over Correctness*: Rather than working behind closed doors to perfect a web application feature, it's more important to get the features out to users so they can assist as codevelopers in the perfecting the features.
- *Select by Crowd over Editor*: The opinions and aggregated wisdom of many people is far more valuable than the opinion of a single person.
- *Honest Voice over Corporate Speak*: The opinions of experts participating in or using a service or product are more valuable than marketing information that has no personal insight.
- *Participation over Publishing*: Whenever possible, it's better to allow the users to participate and share their experience, rather than publishing edited information.
- *Community over Product*: Creating a community and then taking advantage of the collective knowledge of the community is more important than providing a product with individual user access.

The interesting thing is that in this second phase of the Web, the focus is once again on collaboration and sharing information and opinions. This was an original goal of the Internet (`http://en.wikipedia.org/wiki/History_of_the_Internet`) when universities were exploring ways to collaborate.

# Web Application Development 2.0

After reviewing the values and principles that make up a Web 2.0 application, you might be asking yourself "how is this different from what I am doing now?" We have reviewed AJAX interactions in the previous section, and this is by far the most significant change from a development perspective. Other changes are at a far more fundamental software development level and less visible to the end user:

- *Development process agility*: As a service, software features can be changed at lightning speed. It could be at a client's request or as new business requirements are introduced, but either way, a process must be in place to efficiently introduce new features and validate that the new code has not broken existing features. More than ever, unit testing, continuous integration, and automated deployment processes are required to support the development efforts.
- *Syndication and integration*: Two sides of the same coin, syndication and integration allow your application to share data with other external applications as well as use services from external sources. When architecting your web application, thought needs to be put into determining how the application will technically achieve these objectives, as well as what format the data and services being provided will take.
- *Web framework agility*: Having a web development environment that works with the developer to provide an environment that is flexible, productive, and encompasses the values of Web 2.0 is of utmost importance. With Web 2.0, there has been a resurgence of development in existing dynamic languages, such as PHP, as well as newer languages and frameworks, such as Ruby and Ruby on Rails. Struts2 is one of many Java frameworks that provide the maturity, experience, and features to compete with dynamic language frameworks.

The features listed previously are not technical features of web development frameworks, and this is important. As web development matures into a second phase of growth, the focus is on business models and features provided to the users. Technically, the difference is on how the applications are developed—by integrating services (that may be provided by other applications, known as mashups) and data together to provide value.

## Web Framework Agility with Struts2

Because the focus of this book is on web development, we will explore how Struts2 provides agility as a web application framework. However, before getting to Struts2, we need to talk briefly about a new web framework that made its debut around the same time that web applications were releasing Web 2.0 features. This framework is Ruby on Rails.

When Ruby on Rails was released in August 2004, many (if not all) existing web application frameworks went through a period of self-examination; new frameworks were also created (Grails, for example). Several driving factors made Rails so compelling to use as a developer:

- *Full web application stack*: All the basic elements necessary to build a web application were provided in the base distribution.
- *Convention over configuration*: Rather than configuring every element of the application, conventions were used (that could be overridden). A standard directory structure (where each development artifact had a known location) and standard naming conventions are a large part of the conventions.
- *Scaffolding*: The framework can provide fully functional basic user interfaces (with controller logic) for model objects, allowing the application to be used while the real production code is being developed.
- *Interpreted development language*: The underlying development language is Ruby, which is dynamic, object-oriented, and interpreted.

All these features allow developers to be more productive from the initial download and setup of the framework, to the day-to-day development of new web application features.

Struts was first released in July 2001 and was an overwhelming success. It provided an MVC (Model View Controller) pattern implementation for Java web development, allowing web applications written in Java to segment code (rather than writing HTML code in servlets or Java code in JSPs) and to manage reusability and maintenance of existing code.

Over time, developing in Struts required more developer-provided code to implement the necessary web application features, and a proposal for the next generation of Struts was suggested. Architecturally, the changes necessary to implement the proposed features were significant and, rather than starting from scratch, Struts developers approached other open source Java frameworks with a proposal for a merger. Without covering all the details, the result was that WebWork (an OpenSymphony project that itself was an early fork of the Struts code base) merged with Apache to become the basis of Struts2.

---

■ **Note** The history of Struts Ti and the WebWork/Struts merger is documented by Don Brown at `http://www.oreillynet.com/onjava/blog/2006/10/my_history_of_struts_2.html`.

---

Interestingly, one of the WebWork lead developers, Pat Lightbody, had been reviewing features of Ruby on Rails with the goal of making WebWork more productive and easier for developers to use. Some of these features are now part of the Struts2 feature set, and some (because of the Java language constraints as well as maturity reasons) did not make the transition.

Following is a list of Struts2 features that drive the framework forward to be more developer friendly and productive:

- *Java*: The Java language has been around for 10 years now and has matured to a point where most of the features (nonbusiness-related) already exist as libraries. Java is typed (a plus for some developers) and can access an organizational infrastructure that has already been developed (although JRuby and Groovy have options for calling existing Java classes via dynamic languages).
- *Plug-ins*: Functionality provided as core or third-party plug-ins can be added as needed, rather than requiring the core framework to include everything. As well, the plug-in development life cycle (and hence the introduction of new features) is no longer tied to the core framework, allowing more frequent upgrades.
- *Convention over configuration*: Wherever possible, configuration has been eliminated. By using zero-configuration, class names can provide action mappings, and returned result values can provide names for JSPs to be rendered.
- *Annotations rather than XML configuration*: There are two benefits to using annotations: first is a reduction in XML configuration, and second is the configuration is closer to the action class (reducing the effort necessary to determine the action configuration).
- *Data conversion*: The conversion of string-based form field values to objects or primitive types is handled by the framework (and vice-versa), removing the necessity of providing this infrastructure code in the action class.
- *Dependency injection*: All the objects that the action needs to interact with to perform the logic of the action are provided via setters. This reduces the coupling between application layers, and makes the application simpler and easier to test.
- *Testability*: Testing Struts2 actions, interceptor, and other classes is very easy.
- *MVC pattern*: Struts2 continues to use the MVC pattern, providing a layer of abstraction between the view (usually rendered as HTML) and the framework by using a URL. This is important because it doesn't tie the framework to a particular device or rendering style (such as, always refresh an entire page; partial HTML updates; and processing events supplied as AJAX requests).

A lot of work is still needed to get to the productivity level that Ruby on Rails is today, and some features will just never make it (due to the restrictions of the Java language). However, in choosing a web application framework, many factors are involved, and the selection of a programming language that can take advantage of the organization infrastructure that is already in place is one of the most important. With numerous options available to choose from for Java web application frameworks, Struts2 is just as strong of a contender today as it was when Struts was first released—providing the developer productivity features and Web 2.0 functionality that is needed to develop today's web applications.

# Using this Book

Throughout the course of this book, the Struts2 framework will be used to develop a Web 2.0 application. As we have already discussed, Web 2.0 characteristics mostly focus around business features and the underlying business model of the organization. However, to develop a

fully featured application, you need to understand the framework, concepts, configuration, and the (non-Web 2.0 specific) features. With this in mind, this book is divided into four sections:

- Chapter 2 and Chapter 3 provide the fundamentals on Struts2 with information on how to get up and running, how a request is processed, background information on the framework, and configuration information and extension points.
- Chapter 4 provides the background information on the application that is to be developed throughout the course of the book, including the development process to be used, an overview of the application, the use cases that will be developed, and supporting technologies (that are used in combination with Struts2).
- Chapters 5 through 8 describe the core features of any web application: data manipulation, wizards and workflows, security, and rendering information.
- Chapter 9 and Chapter 10 focus on the Web 2.0 features of the application, including syndication, integration, and ways that AJAX can be integrated into the application.

---

■ **Note** The code was developed using Struts version 2.0.9. In most cases, the provided code should be compatible with any 2.0.x release; however, if you do choose to use a more recent version, there may be some incompatibilities.

---

Because the concepts and features being introduced are built upon the knowledge from previous chapters, reading the book forward from start to finish is recommended. If you are familiar with Java web application frameworks and don't want to read the entire book, start with Chapter 2 and Chapter 3. These provide the necessary Struts2-specific information so that you can pick and choose the other chapters to read. If you are familiar with Struts2 and are looking for specific implementation information, Chapters 1 through 4 can be safely skipped.

Finally, the application developed in this book illustrates the most common technologies used when developing web applications today: JSPs, the Spring Framework, and JPA. By using Struts2, you have many other options for the view, business tier, and data tier. Plug-ins are the most common mechanism for integrating new technologies, and the list of all the current plug-ins can be found at `http://cwiki.apache.org/S2PLUGINS/home.html`. When considering alternatives, this is the first place you should look.

The other reference you should keep handy is the Struts2 official document site: `http://struts.apache.org/2.x/docs/guides.html`. Here you will find the most up-to-date information on Struts2, as well as reference documentation, guides, tutorials, and release notes for released versions.

CHAPTER 2

# Getting Up and Running

Starting to work with a new technology or framework can be intimidating. Where do you start learning? How do you know that you are implementing classes correctly? How do you know that the configuration is correct? The easiest way to start out is to follow an example, and Struts2 provides just that, but not in the traditional sense. By using a build tool called *Maven2*, Struts2 is able to generate an example project's files and configuration.

In this chapter, you will learn everything you need to get started with Struts2. Starting with information on the build process, you will continue on to generate an example application. You will run the example project on an application server, and with a running example, learn how the different parts of a basic Struts2 application interact.

## The Build Process

The build process represents an independent, consistent, and repeatable method to package an application in a state that can be deployed or distributed. When presented like this, it is incomprehensible to think that any organization would not be employing such a process. It's easy, right? However, widespread use is limited. Either organizations have no common process, or there is a process, but it is specific to the developers' environment—clicking the Build Project button, using a script that was developed locally, using a common build script that contains hard-coded environmental information, and so on. Each of these scenarios will lead to equally disastrous outcomes when used on a system other than the one where the process was created.

To facilitate good development processes, we will start out using a build process that can be utilized in any environment: on a developer's workstation, on the integration server, on the test server, or on the build server that creates the final distribution packages. The tool we will be using is *Maven2* from the *Apache Foundation*.

---

**Note** There is no requirement to use Maven2 to create a Struts2 application; you could use ANT scripts or your IDE to create the WAR file. The important thing is to have the process independent, consistent, and repeatable.

---

## Maven2

Maven2 is a command-line tool that is used to build, test, report on, and package projects. It provides many features that will make developing your project easier. Here are a few of the features that you will be taking advantage of:

- *Standard directory structure*: Each project that uses Maven2 will have the same directory structure; this makes it easier when developers are working across multiple projects.
- *Plug-in architecture*: Each function of Maven2 is performed by a plug-in, whether the function is compiling classes or deploying the site. If a feature is being used for the first time, the plug-in will be downloaded from a common repository; you no longer need to manually obtain all the parts before starting work.
- *Dependency management*: When dependencies are described in the Maven2 configuration file, they will be accessed from a local repository or downloaded to the local repository during the build process (just as the core Maven2 functionality is). As well as the explicitly configured dependency, the transitive dependencies are managed and downloaded as necessary.
- *Scope management*: The final distribution package contains only the elements required. Test code and dependencies that are not needed (or provided by application servers) in the final package are left out.
- *Archetypes*: The archetype plug-in allows developers to create a default implementation template for a project category. This is then used to quickly create a new project without the need for creating the common directory structure, creating the configuration files, and coding default classes and tests from scratch.

---

■ **Note** More information on the many features of Maven2 is provided in the official document at `http://maven.apache.org`.

---

To build the project locally, you need to install Maven2 and learn about build life cycles.

### COMPARING ANT AND MAVEN2

If you are familiar with Apache ANT build scripts, by now Maven2 may look a little overwhelming. The best example of the differences is comparing ANT vs. Maven2 to Struts2 vs. Ruby on Rails.

In Struts2, developers need to do a lot of work (although this is changing for the better), such as creating actions, mapping results, creating interceptors, and so on. Ruby on Rails abstracts away from the developer all possible configuration and common developer tasks and instead provides intelligent defaults (that can be modified if needed). Maven2 is the same. Instead of creating the same "clean," "compile," "test," and so on ANT tasks, these are handled by Maven2 by relying on a common directory structure. The order (Maven2 life cycle) that the tasks (Maven2 phases) are executed in is also handled automatically, as the common tasks are usually called in the same order for every ANT build file.

One of the most compelling reasons to use Maven2 is that project dependencies can be handled declaratively via the `pom.xml` configuration file. And, if the dependent libraries also use Maven2, transitive dependencies are resolved automatically (because they are defined in the dependent libraries configuration file, they can be automatically downloaded as well).

More information can be found at the Maven2 web site (`http://maven.apache.org`), including a complete list of features (`http://maven.apache.org/maven-features.html`).

## Installing and Using Maven2

Installing Maven2 is easy; the project can be downloaded from the Apache project web site at `http://maven.apache.org/download.html`. Once downloaded, your development environment path needs to be modified to include the Maven2 `bin` directory. On a Linux or UNIX system, this is achieved by `export PATH=/usr/local/maven-2.0.6/bin:$PATH` (for Maven2 having been installed in the `/usr/localmaven-2.0.6` directory), and on a Windows system, with `set PATH=%PATH%; C:\maven-2.0.6\bin` (for Maven2 being installed in the `maven-2.0.6` directory).

---

■ **Note** At the time of writing this book, the most recent version of Maven2 is 2.0.6. However, using any 2.0.x release should work without any significant issues. Version 1 of Maven uses a completely different configuration file and structure, and should be avoided.

---

Once installed, you can check whether the installation is correct by issuing the command `mvn -v`. If the installation is correct, you will get a response of `Maven version: 2.0.6`.

Using Maven2 to build a project is just as simple. First you need to create a Maven2 project configuration file. By convention, this file is called `pom.xml` and is located in the root directory of the project. To make it even easier, Maven2 provides an archetype feature that will create empty directory structures, the project's `pom.xml` file, and even configuration files and sample project files—all for a specific type of project.

After you have a `pom.xml` configuration file, you issue the `mvn` command in the directory that it is located, followed by one or many life cycle phases, for example, `mvn clean package`. Another option is to use a plug-in goal rather than a life cycle phase, for example, `mvn archetype:create`.

### The Maven2 Life Cycle Phases

Unlike other build tools, Maven2 uses common life cycles for building a project. Each life cycle provides multiple phases, which are executed in a specific order to consistently generate the outcome expected for your project. The phases and the order cannot be modified; however, each plug-in (and remember everything in Maven2 is a plug-in) can bind a goal (which can be thought of as a target in ANT) to each and any phase. Because order is important, the following default life cycle phases for building a project are listed in the order that they are called:

- *validate*: Verifies that all needed resources are available.
- *compile*: Compiles the source code for the project.
- *test-compile*: Compiles the source code for any tests within the project.
- *test*: Runs unit tests from the project using an applicable testing framework. These tests should not require the code to be packaged or deployed.
- *package*: Packages the compiled code and resources into a distributable format.
- *integration-test*: Deploys the packaged project into an environment where any integration tests can be run and executes any integration tests.
- *install*: Installs the packaged project into a local repository so that other projects may use it.
- *deploy*: Deploys the package into a remote repository to share with other developers and projects.

---

■ **Note** This is not a complete listing of all the life cycle phases that are available. If you are interested in learning more, the full life cycle phase list can be found at `http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html`.

---

Two additional life cycles are available: *clean* and *site*.

The *clean* lifecyle removes the build directory, along with any other configured directories to restore the state of the project to a baseline.

The *site* life cycle has the following goals:

- *site*: Runs the reports configured for a project, rendering HTML documents.
- *site-deploy*: Deploys the HTML reports to a configured web server.

Similar to the *default* life cycle, both contain additional phases that are not listed.

# The Struts2 Starter Application

The Struts2 project includes several Maven2 archetypes that can be used to kickstart project development. The full list of different archetypes can be found at `https://svn.apache.org/repos/asf/struts/maven/trunk`. Included are archetypes for portlet development, plug-in development, and Struts2 projects.

You will be using the *starter* project archetype, called `struts2-archetype-starter`. To generate the starter project, select the working directory, and issue the following command:

```
mvn archetype:create
  -DgroupId=com.fdar.apress.s2
  -DartifactId=app
  -DarchetypeGroupId=org.apache.struts
  -DarchetypeArtifactId=struts2-archetype-starter
  -DarchetypeVersion=2.0.9-SNAPSHOT
  -DremoteRepositories=http://people.apache.org/maven-snapshot-repository
```

This command has two parameters that can be varied. The `artifactId` property specifies the directory name to use as the base directory for the project (created in the working directory that you selected and in which you issued the Maven2 command to create the archetype) and is also used as the project's name. Into this directory, the project will be created with the common Maven2 directory structure. The `groupId` is the package name to use as the base class directory and the directory in which the starter classes will be located.

Now that the starter project has been created, you can see it working in a browser. The Maven2 command `mvn jetty:run` will start a servlet container with the application deployed, but remember to issue the command from the directory containing the `pom.xml` configuration file (the `app` directory). When this command is run for the first time, there will be many plug-ins and dependency artifacts to download, so it may take some time. After the artifacts are cached in your local repository, the startup time will improve. We will discuss how the servlet container is configured in the following chapters, but for now, you have a working application in only two steps.

## The Maven2-Generated Directory and File Structure

After the Maven2 `struts2-archetype-starter` archetype has been run, many directories (in accordance with the Maven2 standard directory structure) and files are created. The complete directory structure follows.

```
app
  +- pom.xml
  +- src
       +- main
       |    +- java
       |    |    +- com
       |    |         +- fdar
       |    |              + apress
       |    |                  +- s2
```

```
|    +- resources
|    +- webapp
|         +- jsp
|         +- styles
|         +- WEB-INF
|              +- decorators
+- test
     +- java
     |    +- com
     |         +- fdar
     |              + apress
     |                   +- s2
     +- resources
```

The standard directory structure goes like this: the `src` directory is the root for all code in the project, and within this directory, there is a `main` directory (for production code) and a `test` directory. Only the `src` directory's contents (after any processing/compiling is performed) go into the packaged artifact.

---

■ **Note** For more detail on the standard directory structure, see the Maven2 documentation at `http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html`.

---

These two directories can contain many other directories, which are aligned with a plug-in or technology. The most common are the `java` and `resource` (containing property, XML, and other configuration files) directories—because Maven2 is a build tool for Java—but there are many more. When the AspectJ (AspectJ provides a way to weave additional code for cross-cutting concerns, with logging and transaction management being the commonly used examples, into existing compiled code) plug-in is included, an `aspects` directory contains the `.aj` files. The same goes for scripting, where a `groovy` directory contains the Groovy scripts that are to be executed. (Groovy is a dynamic scripting language with syntax similar to Java that can be executed on the fly or compiled down to byte code.) For a `war` packaged artifact, the `main` directory includes a `webapp` directory that contains the additional information for a WAR file that an EAR or JAR does not need.

## The Maven2 Configuration File

As well as the starter program elements, the archetype will create a Maven2 `pom.xml` configuration file that is used to build the project. This file defines the project's dependencies, the packaging details, and the testing and reporting requirements.

The configuration file plays a central role, so let's take some time to understand its parts. The first part is the header information:

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.fdar.apress.s2</groupId>
    <artifactId>app</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>Struts 2 Starter</name>
    <url>http://www.myComp.com</url>
    <description>Struts 2 Starter</description>

    …

</project>
```

The header of the configuration file contains information for the final packaging, and you should see several values that are familiar: the `groupId` and `artifactId` have the same values that were provided in the command to create the starter package.

Some information will remain constant; the `modelVersion` will always have a value of `4.0.0` (until a significant Maven2 configuration format revision occurs), which refers to the Maven2 model version. The `packaging` value for web applications is usually `war`. If this were a component of a larger application, the value would be `jar`; and if it were a J2EE application containing web components, EJB components, and other resources, it would be `ear`. The `version` value remains constant for the time being but will change over time. As this component becomes stable or goes into preview or testing phases, it may change. When the component is released for production use, it should be changed to `1.0`. As further development starts, it may be changed to `1.1-SNAPSHOT` (for enhancements) or `2.0-SNAPSHOT` (for new major features).

Three elements—the `name`, `url`, and `description` tags—have a default value that you should change soon after the starter code has been generated. Each provides descriptive information to developers and consumers of the packaged artifact but is not utilized during building or packaging.

The next interesting part of the `pom.xml` file is the dependency section:

```
<project>

    …

    <dependencies>
        <!-- Junit -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
```