Java EE 7 Development with WildFly

Java Enterprise Edition is a well-known platform for application development. Its seventh release brings many new features and API improvements. WildFly is a successor of the JBoss Application Server family and fully implements the Java EE 7 standards.

Java EE 7 Development with WildFly shows you how to use the latest version of the Java EE 7 platform, including its most advanced features. Every topic is presented using working examples that are prepared to work out of the box with the new WildFly application server.

Who this book is written for
If you are a Java developer who wants to learn about Java EE, this is the book for you. It’s also ideal for developers who already have experience with the Java EE platform but would like to learn more about the new Java EE 7 features by analyzing fully functional sample applications using the new application server WildFly.

What you will learn from this book
- Design reliable and efficient applications based on Enterprise Java Beans and Context and Dependency Injection
- Set up and manage your own WildFly application server for different development scenarios
- Store your data without binding yourself to a specific database provider using the Java Persistence API
- Create message-based applications using Java Messaging System and integrate third-party systems into them using Java Connector Architecture
- Distribute and secure your enterprise applications by using built-in mechanisms from WildFly and Java EE 7
- Decide between using SOAP and JAX-RS web services to create your external API
- Discover the new side of web applications by exploring WebSockets and their usage scenarios
- Test your application using a real environment with Arquillian

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Getting Started with WildFly'
- A synopsis of the book’s content
- More information on Java EE 7 Development with WildFly

About the Author

Michał Ćmil is a Java developer with experience in web and desktop application development. In the past, he has worked for the e-commerce sector. Currently, he is developing manufacturing execution systems for industry clients. He has been a volunteer at GeeCON. Michał works as a software designer/developer for PSI Poland.

Michał Matłoka is a Java developer with experience in both Java EE and Spring technologies. He is a JBoss community contributor, focusing mainly on Arquillian and ShrinkWrap, and is also a winner of the JBoss Community Recognition Award in 2013 for his contributions to ShrinkWrap. He has spoken at Poznan Java User Group and volunteered at GeeCON. Michał is currently one of the 20 CEOs at SoftwareMill, a fully distributed company with no main office and a completely flat organization structure.

Francesco Marchioni is a Sun Certified Enterprise architect employed for an Italian company based in Rome. He started learning Java in 1997, and since then he has followed the path to the newest Application Program Interfaces released by Sun. In 2000, he joined the JBoss community when the application server was running the 2.x release.

He has spent many years as a software consultant, where he has envisioned many successful software migrations from vendor platforms to the open source products such as JBoss AS, fulfilling the tight budget requirements of current times.

For the past 5 years, he has been authoring technical articles for O'Reilly Media and also runs an IT portal focused on JBoss products (http://www.mastertheboss.com).
In December 2009, he authored *JBoss AS 5 Development, Packt Publishing*, which describes how to create and deploy Java Enterprise applications on JBoss AS.

In December 2010, he authored his second title, *JBoss AS 5 Performance Tuning, Packt Publishing*, which describes how to deliver fast and efficient applications on JBoss AS.

In December 2011, he authored yet another title, *JBoss AS 7 Configuration Deployment Administration, Packt Publishing*, which covers all the aspects of the newest application server release.


---

I'd like to thank Packt Publishing for sharing the vision of this new book and for all the effort they put in it. I'd also like to thank my family for being always on my side, particularly, my wife for allowing me to follow my book author ambitions, and my father for buying me a C-64 instead of a motorcycle when I was young.
Java EE 7 Development with WildFly

WildFly, the newest release of the JBoss Application Server, provides developers with a full implementation of the Java EE 7 platform. It is built on solid foundations of the modular architecture introduced with JBoss AS 7, but it has grown in terms of flexibility and performance. The newest version of the Enterprise Java focuses on developers' productivity, and so does WildFly.

This book will introduce Java developers to the world of enterprise applications. We will use modern development techniques and tools that are battle-tested in real-life projects. We'll also utilize the features provided by the WildFly platform, such as security, caching, testing, and clustering. Finally, you will learn how to manage your server using dedicated tools, created specifically for WildFly.

The learning process will be concentrated around a ticket booking application, which is a sample project that will get more features (and sometimes completely different user interfaces) with every chapter.

What This Book Covers

Chapter 1, Getting Started with WildFly, is an introduction to the Java EE platform and new Java EE 7 version specifications. It also focuses on presenting WildFly's new features, the developer environment setup, and basic server management.

Chapter 2, Your First Java EE Application on WildFly, describes the basics of the WildFly server usage, presenting information required to deploy your first application.

Chapter 3, Introducing Java EE 7 – EJBs, introduces Java EE business objects called Enterprise Java Beans. In this chapter, we create the foundations of the ticket booking application.

Chapter 4, Learning Context and Dependency Injection, covers the CDI technology that connects the building blocks of your applications.

Chapter 5, Combining Persistence with CDI, is a look into the database world and object mapping in Java EE.

Chapter 6, Developing Applications with JBoss JMS Provider, is a dive into HornetQ and enterprise system integration using JCA.

Chapter 7, Adding Web Services to Your Applications, discusses not only the old-style SOAP web services but also the modern and popular approach based on JAX-RS (REST). We'll also take a look at how to integrate a Java EE 7 backend with an AngularJS browser application.
Chapter 8, Adding WebSockets, introduces a completely new asset to the Java EE 7 platform: WebSockets. We will look at them in our sample AngularJS application.

Chapter 9, Managing the Application Server, discusses WildFly management features.

Chapter 10, Securing WildFly Applications, focuses on security-related aspects of the server and your application.

Chapter 11, Clustering WildFly Applications, discusses making Java EE applications highly available and scalable.

Chapter 12, Long-term Tasks' Execution, describes the new area of enterprise Java batch applications and concurrency management on the server.

Chapter 13, Testing Your Applications, demonstrates how we can write integration tests for our applications using Arquillian after covering the most important Java EE technologies.

Appendix, Rapid Development Using JBoss Forge, covers the JBoss Forge tool. It shows how you can use this application to speed up your work with its code generation features on starting Java EE-based projects.
Getting Started with WildFly

Java Enterprise Edition provides a standard to develop enterprise software, but allows the developers to choose its specific implementation. For every technology that is included in the Java EE (Enterprise Edition) specification, there is a reference implementation; an open source library or component that fulfills all of the requirements. Companies and organizations can create their own versions of the components, which means that there is no central Java EE platform that everybody uses. In place of that, we get multiple approaches on the implemented specification, with improvements and optimizations for specific cases. At the time of writing this, there are about 20 certified (full) implementations of Java EE 6 and three implementations of Java EE 7.

An application server is a runtime environment that provides applications with all the Java EE components. Glassfish is the reference implementation sponsored by Oracle, but beginning from Version 4 (created for Java EE 7), there is no longer commercial support for it. In this book, you will learn how to develop applications on the WildFly Application Server, previously known as the JBoss Application Server.

JBoss is a division of Red Hat, which seeks to provide a developer-friendly open source ecosystem for enterprise development. Currently, the company supports multiple projects (around 100), and some of them are implementations of Java EE specifications. The enterprise elements are combined in JBoss's own application server, WildFly.

It is worth noting that the name change from JBoss AS to WildFly was made to separate the application server from the company and other subprojects. The name was chosen in a public vote (more information on this is available at http://jbossas.jboss.org/rename/vote).
The new release features a scalable and high performing web server called Undertow, which supports the HTTP upgrade mechanism and WebSocket protocol. What's more, the new version of the container is even faster than JBoss Application Server 7, and offers a unified configuration mechanism. However, the main essence of the latest release is the Java EE 7 compliance, which allows developers to use technologies from the newest version of the Java EE specification.

The focus of this book is on application development; therefore, we will first need to gather all resources required to deliver our applications. In this chapter, we will cover the following topics in detail:

- An overview of Java EE and WildFly
- Preparing your environment for the installation of software
- Downloading and installing WildFly
- Verifying the WildFly installation
- Installing other resources needed for development

**An overview of Java EE and WildFly**

Java EE (formerly called J2EE) is an umbrella specification embracing a standard set of technologies for server-side Java development. Java EE technologies include Java Servlet, [JavaServer Pages (JSPs)](https://en.wikipedia.org/wiki/JavaServer_Pages), [JavaServer Faces (JSFs)](https://en.wikipedia.org/wiki/JavaServer_Faces), [Enterprise JavaBeans (EJB)](https://en.wikipedia.org/wiki/Enterprise_JavaBeans), [Contexts and Dependency Injection (CDI)](https://en.wikipedia.org/wiki/Contexts_and_Dependency_Injection), [Java Messaging Service (JMS)](https://en.wikipedia.org/wiki/Java_Messaging_Service), [Java Persistence API (JPA)](https://en.wikipedia.org/wiki/Java_Persistence_API), [Java API for XML Web Services (JAX-WS)](https://en.wikipedia.org/wiki/Java_API_for_XML_Web_Services), and [Java API for RESTful Web Services (JAX-RS)](https://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services), among others. The newest version of Java EE extends the range of available technologies even further by providing support for Batch Applications, Concurrency Utilities, [JSON Processing (JSON-P)](https://en.wikipedia.org/wiki/JSONProcessing) and WebSocket. Several commercial and open source application servers exist, which allow developers to run applications compliant with Java EE; WildFly (formerly known as JBoss AS) is the leading open source solution adopted by developers and, although this is difficult to measure in exact terms, it is likely to be the most widely used application server in the market.

As with all application servers compliant with Java EE, WildFly ships with all the required libraries to allow us to develop and deploy Java applications that are built on the Java EE platform.
WildFly and Enterprise Application Platform

WildFly and previous JBoss Application Servers were freely available for the community in the form of downloadable binary packages (for major releases) or buildable source code (for bug fixing releases). These versions are called community releases and are free to use for development and production.

JBoss also releases more stable and hardened versions of software of the community builds, which are called Enterprise Application Platform (EAP), a commercial product with support service from Red Hat. Red Hat calls this kind of relationship between projects as upstream/downstream. The community builds are the source of changes and innovations for the downstream, the code is downstream. The commercial version numeration differed from the community line, but it was an extended variant of the community releases (for example, EAP 6.1.0 was built on JBoss 7.2.0, which was available only on GitHub in the form of buildable source code; the same goes for EAP 6.2.0 and JBoss 7.3.0). EAP builds have a more complex licensing system; the usage terms depend on the maturity of the build and are as follows:

- **EAP Alpha** is free for the developers and production use, as they are an equivalent of the standard community version with optional fixes included. The corresponding community binaries may not be available for download, as they would be similar to the EAP Alpha version.
- **EAP Beta** is available to developers for free (after registration to a subscription program), but cannot be used in production.
- **EAP Final** is also available to developers for free, but additionally, new security patches are available only in the paid subscription.

The distribution model proposed by JBoss allows the developers to work for free on the same version as the one used in production. This is a huge benefit, especially since the competitive solution from Oracle (Glassfish: the reference implementation of a Java EE compliant server) no longer has a version with commercial support.

Welcome to Java EE 7

Java EE 7 includes several improvements and additions to the existing version. The new version is focused on three themes: developer productivity, HTML5, and providing new features required by enterprise applications. The following sections list the major improvements to the specifications that are of interest to enterprise application developers.

If you are starting your adventure with Java EE, feel free to skip this section. The technologies described in the following sections will be covered in future chapters in more detail.
JavaServer Faces 2.2 – JSR 344
Java EE 7 includes a new version of the JSF specification, which is not so revolutionary as 2.0, but still provides some appealing additions for developers. The key features delivered by JSF 2.2 are as follows:

- The HTML5 markup is now supported by the usage of pass-through elements and attributes. Earlier, custom attributes would have to be supported by an extended renderer for every component. The new constructs allow the developer to pass additional HTML attributes to the markup generated by JSF components.
- The flow scope has been introduced with `@FlowScoped`, which makes the creation of wizards (dialogs with multiple steps) easier.
- The Ajax-based file upload is now supported out of the box.
- Also, stateless views have been presented as a way to improve performance.

Enterprise JavaBeans 3.2 – JSR 345
Compared to EJB 3.1, the Version 3.2 is a minor update of the existing version. It concentrates mainly on marking some older features as obsolete (they are now optional, which means that not every Java EE 7-compliant application server will support them). The optional features are connected with persistence to web services based on EJB 2.1 and JAX-RPC. The main enhancements provided by the new specification are as follows:

- Life cycle methods for stateful session beans can now be transactional.
- The Timer Service API, now allows you to access all active timers in the current EJB module.
- A new container provided role (**) has been introduced. It can be used to indicate any authenticated user (without taking his or her actual roles into account).
- Passivation of stateful session beans can now be disabled.

Transactional parts of the EJB specification have been extracted and reused in other parts of the Java EE platform (the transaction support has been placed in the JTA 1.2 specification). For instance, the transactional behavior can now be used in CDI beans, thanks to the introduction of the `@Transactional` annotation.
Java Persistence API 2.1 – JSR 338

JPA was introduced as a standard part of Java EE in Version 5 of the specification. JPA was intended to replace entity beans as the default object-relational mapping framework for Java EE. JPA adopted ideas from third-party object-relational frameworks, such as Hibernate and JDO, and made them a part of the standard version.

JPA 2.1 is an improvement over JPA 2.0 as it provides several facilities for developers, which are as follows:

• It provides a standardized schema generation mechanism, thanks to an extended set of annotations and persistence.xml properties
• It adds support for type conversion, by the introduction of the @Converter annotation
• Stored procedures are now supported by the Entity Manager API, so that the use of the SQL query mechanism for them is no longer required
• Criteria API has been extended by bulk updates and deletes
• Injection is possible into entity listener classes along with the usage of life cycle callback methods
• Named queries can now be created during runtime
• The JPA Query Language (JPQL) has been extended with new database functions

Additionally, Java EE 7-compliant containers must now support preconfigured data sources (along with other resources), which can be instantly used by JPA entities.

WildFly uses Hibernate as its JPA provider and is shipped with a ready-to-use H2 in-memory database. The default data source points to the H2 instance hosted inside of the application server.
Contexts and Dependency Injection for
Java EE 1.1 – JSR 346

Version 1.1 of Contexts and Dependency Injection (CDI) provides improvements for the issues identified in CDI after its introduction in Java EE 6. The process of simplifying the programming model started in Version 1.0 and is now being continued. The areas covered by the update are as follows:

• CDI is now enabled by default (without the need to add the bean.xml file to the deployment), with the possibility to specify the desired component scanning mode.

• More fine-grained control over the bean discovery mechanism is now available for the developer, thanks to the use of the @Vetoed annotation and class or packages filters in beans.xml. Interceptors, decorators, and alternatives can now be globally enabled for the whole application using the @Priority annotation, instead of enabling every module.

• Event metadata can now be examined when a CDI event is handled.

• Interceptors have been enhanced with the possibility to be executed around constructor invocation.

• Finally, the new version contains a significant number of enhancements for the development of portable extensions.

Weld is the CDI implementation internally used in WildFly.

Java Servlet API 3.1 – JSR 340

The new version of the Java Servlet API has a clear focus on new features. The most important of them is the HTTP upgrade mechanism, which allows the client and server to start a conversation in HTTP 1.1, and negotiate another protocol for subsequent requests. This feature was used to implement the WebSockets mechanism in Java EE 7. Other features of the new version of specification are as follows:

• Non-blocking I/O API for Servlets has been provided to improve scalability of web applications

• Multiple security improvements have been introduced; the most notable of them is the possibility to set the default security semantics for all HTTP methods
JAX-RS, the Java API for RESTful Web Services 2.0 – JSR 339

In Java EE 7, the JAX-RS specification has been enriched with some long-awaited features. The version has changed from 1.1 to 2.0 because of the major impact of the improvements that came with the new specification. The most important features are listed as follows:

- The client API is now part of the specification, so the usage of third-party libraries is no longer needed. The implementation of the specification is required to provide a REST client that conforms to the common API.
- Asynchronous requests are now supported so that the client does not have to passively wait for the completion of the task.
- Filters and handlers have been introduced as a common mechanism to provide extension points for the developer. They can be used for cross-cutting concerns, such as auditing and security.
- Bean Validation has been integrated into JAX-RS, making constraint annotations usable for request parameters.

WildFly comes bundled with RESTEasy, an implementation of JAX-RS 2.0.

Java Message Service 2.0 – JSR 343

The JSR 343 is the first update for the JMS specification in over a decade. Once more, the main theme of the update is the simplification of the API. The new API dramatically decreases the amount of boilerplate code that has to be written by the programmer while still maintaining backwards compatibility. Other new features are listed as follows:

- Asynchronous message sending is now supported, so the application does not have to be blocked until an acknowledgment from the server is received
- Messages can now be sent with a scheduled delay for the delivery

HornetQ is the JMS provider used and developed by JBoss. It is possible to use it outside of WildFly as a standalone message broker.
Bean Validation 1.1 – JSR 349
The process of updating the Bean Validation in Java EE 7 concentrates on two main features:

- Methods validation, which allows the developer to validate parameters and return values
- Tighter CDI integration, which changes the life cycle of the elements of the validation framework, allowing the developer to use dependency injection in his or her own ConstraintValidator implementations

Concurrency utilities for Java EE 1.0 – JSR 236
Concurrency utilities is a new feature pack to use multithreading in Java EE application components. The new specification provides ManagedExecutorService (a container-aware version of ExecutorService known from Java SE), which can be used to delegate the execution of tasks to a separate thread. These managed tasks could use most of the features that are available for application components (such as EJBs or Servlets). It is also possible to schedule cyclic or delayed tasks using new ManagedScheduledExecutorService. These new additions to the platform are filling a functional gap for Java EE, which was very hard to overcome within its architecture earlier on.

Batch applications for the Java Platform 1.0 – JSR 352
Batch jobs were another area of enterprise application development, which was not covered by earlier versions of Java EE. The new batch processing framework is used to provide a common solution to run tasks that are executed without user interaction. Java EE provides the developer with the following options:

- Batch runtime for the execution of jobs
- A job description language (based on XML)
- The Java API for the implementation of the business logic for the batch tasks
- jBeret, which is the batching framework used in WildFly

Java API for JSON Processing 1.0 – JSR 353
Java EE 7 now comes with out-of-the-box JSON processing, so the developer is no longer forced to use external libraries for this task. The new API allows JSON processing to use two approaches: object model (DOM based) and streaming (event-based).
Java API for WebSocket 1.0 – JSR 356

To fully support the development of applications based on HTML5, Java EE 7 requires a standardized technology for two-way communication between the server and the user’s browser. The WebSocket API allows the developer to define server-side endpoints, which will maintain a TCP connection for every client that will connect to them (using, for instance, a JavaScript API). Before the new specification, developers had to use vendor-specific libraries and not portable solutions to achieve the same goal.

New features in WildFly

The eighth release of WildFly is based on modular architecture introduced in the previous release named JBoss AS 7. It has improved on several key points, including areas of performance and management. The most important change for the developers is that this release implements the Java EE 7 standard completely. Some of the most notable improvements include the following:

- WildFly 8 implements all standards presented by Java EE 7 specifications, which are also described in this chapter.
- The web server module was completely rewritten under the name Undertow. It supports both blocking and non-blocking operations. Early performance tests (for example, http://www.techempower.com/benchmarks/#section=data-r6&hw=i7&test=plaintext) show major performance improvements in HTTP request handling. Undertow is also available as a separate project and is possible to be used without WildFly.
- The final WildFly release has reduced the number of used ports. Now, it uses only two of them, one (9990) for management, JMX, and web administration, and the second one (8080) for standard services, including HTTP, WebSockets, remote JNDI, and EJB invocations.
- Now, it is possible to limit a user's management permissions using the Management role based access control (RBAC). All configuration changes can be tracked using the audit log.
- For previous releases, any upgrade operation requires a completely new server installation. WildFly brings the patching feature, allowing to install and rollback modules using management protocols.

In the next section, we will describe all the required steps to install and start a new application server.
Installing the server and client components

The first step in learning about the application server will be to install all the necessary components on your machine in order to run it. The application server itself requires just a JDK environment to be installed.

As far as hardware requirements are concerned, you should be aware that the server distribution, at the time of writing this, requires about 130 MB of hard disk space, and allocates a minimum of 64 MB and a maximum of 512 MB for a standalone server.

In order to get started, we will need to go over this checklist:

- Install JDK where WildFly will run
- Install WildFly
- Install the Eclipse development environment
- Install the Maven build management tool

At the end of this chapter, you will have all the instruments to get started with the application server.

Installing Java SE

The first mandatory requirement is to install a JDK 8 environment. The Java SE download site can be found at http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Choose the latest version of Java SE 8 and install it. If you don't know how to install it, please take a look at http://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html.
Testing the installation

Once you have completed your installation, run the `java -version` command from a command prompt to verify that it is correctly installed. Here is the expected output from a Windows machine:

```
C:\>java -version
java version "1.8.0_11"
Java(TM) SE Runtime Environment (build 1.8.0_11-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.11-b03, mixed mode)
```

Installing WildFly

The JBoss WildFly application server can be downloaded for free from http://wildfly.org/downloads/.

As you can see in the following screenshot, at the moment of writing this book, the latest stable release of WildFly is 8.1.0.Final, which features a Certified Java EE 7 full profile:
Once you have chosen the appropriate server distribution, you will then be warned that this download is part of a community release and, as such, is not supported.

If you need Enterprise support for your applications, you can opt for the Red Hat Enterprise Application Platform.

Compared to the community version, the EAP has gone through different quality tests and might be different in terms of features/packaging. More information about the differences between EAP and the community version can be found at the beginning of this chapter. However, at the time of writing this book, EAP does not yet support Java EE 7, and no road map is publicly available.

Installing WildFly is a piece of cake; it does not require anything else besides unpacking the wildfly-8.1.0.Final.zip archive.

Windows users can simply use any uncompressing utility, such as built-in compressed folders (in newer Windows releases), WinZip, WinRAR, or 7-Zip taking care to choose a folder name that does not contain empty spaces and white spaces. Unix/Linux should use the $ unzip wildfly-8.1.0.Final.zip unzip shell command to explode the archive.

Security warning

Unix/Linux users should be aware that WildFly does not require root privileges, as none of the default ports used by WildFly are below the privileged port range of 1024. To reduce the risk of users gaining root privileges through WildFly, install and run WildFly as a non-root user.

Starting WildFly

After you have installed WildFly, it is wise to perform a simple start up test to validate that there are no major problems with your Java VM / operating system combination. To test your installation, move to the bin directory of your JBOSS_HOME (the path to which you have unzipped your application server) directory and issue the following command:

standalone.bat    # Windows users
$ ./standalone.sh  # Linux/Unix users
The following is a screenshot of a sample WildFly start-up console:

![WildFly Start-up Console Screenshot]

The preceding command starts up a WildFly standalone instance, which is equivalent to starting the application server with the `run.bat/run.sh` script used by earlier JBoss AS releases. You will notice how amazingly fast the new release of the application server is; this is due to the modular architecture introduced in Version 7 of JBoss AS, which only starts up necessary parts of the application server container needed by the loaded applications.

If you need to customize the start-up properties of your application server, open the `standalone.conf` file (or `standalone.conf.bat` for Windows users), where the memory requirements of Wildfly have been declared. Here is the Linux core section of this file:

```
if [ "x$JAVA_OPTS" = "x" ]; then
    JAVA_OPTS="-Xms64m -Xmx512m -XX:MaxPermSize=256m
    -Djava.net.preferIPv4Stack=true"
    JAVA_OPTS="$JAVA_OPTS
    -Djboss.modules.system.pkgs=$JBOSS_MODULES_SYSTEM_PKGS
    -Djava.awt.headless=true"
else
    echo "JAVA_OPTS already set in environment; overriding default settings with values: $JAVA_OPTS"
```

[19]
So, by default, the application server starts with a minimum memory requirement of 64 MB of heap space and a maximum of 512 MB. This will be just enough to get started; however, if you need to run more robust Java EE applications on it, you will likely require at least 1 GB of heap space or 2 GB or more, depending on your application type. Generally speaking, 32-bit machines cannot execute a process whose space exceeds 2 GB; however, on 64-bit machines, there is essentially no limit to the process size.

You can verify that the server is reachable from the network by simply pointing your browser to the application server's welcome page, which can be accessed from the well-known address, http://localhost:8080. The welcome page of WildFly is shown in the following screenshot:
Connecting to the server with the command-line interface

If you have been using previous releases of the application server, you might have heard about the twiddle command-line utility that queried MBeans installed on the application server. This utility has been replaced by a more sophisticated interface named the command-line interface (CLI); it can be found in JBOSS_HOME/bin.

Just launch the jboss-cli.bat script (or jboss-cli.sh for Linux users), and you will be able to manage the application server via a shell interface. This is shown in the following screenshot:

We started an interactive shell session that can also use the command-line completion (by pressing the Tab key) to match partly typed command names. No more searches are needed to find the exact syntax of commands!

In the previous screenshot, we connected to the server using the connect command; it uses the loopback server address and plugs into port 9990 by default.

The command-line interface is discussed in depth in Chapter 9, Managing the Application Server, which is all about server management interfaces; we will, however, get an initial taste of its basic functionalities in the next sections to get you accustomed to this powerful tool.
Stopping WildFly

The easiest way to stop WildFly is by sending an interrupt signal with Ctrl + C.

However, if your WildFly process was launched in the background or, rather, is running on another machine, you can use the CLI interface to issue an immediate shutdown command:

```
[disconnected ]/ connect
Connected to localhost:9990
[standalone@localhost:9990 ]/:shutdown
```

Locating the shutdown script

There is actually one more option to shut down the application server that is pretty useful, if you need to shut down the server from within a script. This option consists of passing the --connect option to the admin shell, thereby switching off the interactive mode as shown in the following command line:

```
jboss-cli.bat --connect command=:shutdown      # Windows
./jboss-cli.sh --connect command=:shutdown       # Unix / Linux
```

Stopping WildFly on a remote machine

Shutting down the application server that is running on a remote machine is just a matter of providing the server’s remote address to the CLI, and for security reasons, a username and password, as shown in the following code snippet (see the next chapter to learn more about user creation):

```
[disconnected ]/ connect 192.168.1.10
Authenticating against security realm: ManagementRealm
Username: admin1234
Password:
Connected to 192.168.1.10:9990
[standalone@192.168.1.10:9990 ]/:shutdown
```

However, you have to remember that you need access to a given port because often, it may be blocked by a firewall.
Restarting WildFly

The command-line interface contains many useful commands. One of the most interesting options is the ability to reload the AS configuration or parts of it using the `reload` command.

When issued on the root node path of the AS server, the `reload` command can reload the services' configuration:

```
[disconnected /] connect
Connected to localhost:9990
[standalone@localhost:9990 /] :reload
```

Installing the Eclipse environment

The development environment used in this book is Eclipse, which is known by Java developers worldwide, and it contains a huge set of plugins to expand its functionalities. Besides this, Eclipse is the first IDE that is compatible with the new application server.

So, let's move to the download page of Eclipse, which is located at http://www.eclipse.org.

From here, download the latest Enterprise Edition (at the time of writing this book, it is Version 4.4 and is also known as Luna). The compressed package contains all the Java EE plugins already installed. This is shown in the following screenshot:

Once you have unzipped the previously downloaded file, you will see a folder named `eclipse`. In this folder, you will find the Eclipse application (a big blue dot). It is recommended that you create a shortcut on the desktop to simplify the launching of Eclipse. Note that, just as with WildFly, Eclipse does not have an installation process. Once you have unzipped the file, you are done!
Installing JBoss Tools

The next step will be installing the JBoss AS plugin that is a part of the suite of plugins named JBoss Tools. Installing new plugins in Eclipse is pretty simple; just follow these steps:

1. From the menu, navigate to Help | Install New Software.
2. Then, click on the Add button, where you will enter JBoss Tools' download URL (along with a description), http://download.jboss.org/jbosstools/updates/development/luna/. This is shown in the following screenshot:

3. As you can see in the preceding screenshot, you need to check the JBossAS Tools plugin and move forward to the next option to complete the installation process.
Enter JBossAS into the filter field to quickly find the JBoss AS Tools plugin among the large set of JBoss Tools.

4. Once done, restart the process when prompted.

You can also download JBoss Tools as individual zip files for an offline installation. See JBoss Tools Downloads at http://tools.jboss.org/downloads/.

5. Now, you should be able to see WildFly listed as a server by navigating to New | Server from the upper menu and expanding the JBoss Community option, as shown in the following screenshot:
Completing the server installation in Eclipse is quite straightforward as it just requires pointing to the folder where your server distribution is; we will therefore leave this to you to implement as a practical exercise.

**Alternative development environments**

Since this book is all about development, we should also account for some other alternatives that might suit your programming style or your company standards better. So, another valid alternative is IntelliJ IDEA, which is available at [http://www.jetbrains.com/idea/index.html](http://www.jetbrains.com/idea/index.html).

IntelliJ IDEA is a code-centric IDE focused on developer productivity. The editor exhibits a nice understanding of your code and makes great suggestions right when you need them and is always ready to help you shape your code.

Two versions of this product exist—Community edition and Ultimate edition—that require a license. In order to use Java EE and the WildFly plugin, you need to download the ultimate edition from [http://www.jetbrains.com/idea/download/index.html](http://www.jetbrains.com/idea/download/index.html) and then simply install it using the installation wizard.

Once you have installed the Ultimate edition, you will be able to get started with developing applications with WildFly by going to **File | Settings** and choosing the **IDE Settings** option. Here, you can choose to add new application server environments. This is shown in the following screenshot:
Another development option that is quite popular among developers is NetBeans (http://netbeans.org), which has support for WildFly in its releases 7.4 and 8.0, but requires installation of additional plugins available in the NetBeans plugins registry.

**Installing Maven**

Besides graphical tools, you are strongly encouraged to learn about Maven, the popular build and release management tool. By using Maven, you will enjoy the following things:

- A standard structure for all your projects
- A centralized and automatic management of dependencies

Maven is distributed in several formats, for your convenience, and can be downloaded from http://maven.apache.org/download.html.

Once the download is complete, unzip the distribution archive (for example, apache-maven-3.1.1-bin.zip for Windows) to the directory in which you wish to install Maven 3.1.0 (or the latest available version), for example, C:\\Programs\apache-maven-3.1.1. Some operating systems such as Linux or OS X offer Maven packages in their application repositories.

Once done, add the M2_HOME environment variable to your system so that it will point to the folder where Maven has been unpacked.

Next, update the PATH environment variable by adding the Maven binaries to your system path. For example, on the Windows platform, you should include %M2_HOME%/bin in order to make Maven available on the command line.

Some additional Maven learning materials are available on the Sonatype website in the form of free books; refer to http://www.sonatype.com/resources/books.

**Testing the installation**

Once you have completed your installation, run the mvn version to verify that Maven has been correctly installed. Refer to the following code snippet to verify the correct installation:

```
> mvn -version
Apache Maven 3.1.1 (0728685237757ffbf44136acec0402957f723d9a; 2013-09-17 17:22:22+0200)
Maven home: C:\Programs\Dev\apache-maven-3.1.1
```
Java version: 1.8.0_11, vendor: Oracle Corporation
Java home: C:\Programs\Java\jdk1.8.0\jre
Default locale: en_US, platform encoding: Cp1250
OS name: "windows 8.1", version: "6.3", arch: "amd64", family: "dos"

---

### Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at [http://www.packtpub.com](http://www.packtpub.com). If you purchased this book elsewhere, you can visit [http://www.packtpub.com/support](http://www.packtpub.com/support) and register to have the files e-mailed directly to you.

---

### Summary

In this chapter, we ran our first mile on the track to application server development. We introduced the new features of the application server and had an overview of the Java Platform Enterprise Edition in Version 7, also known as Java EE 7.

Next, we discussed the installation of the WildFly AS and all the core components that include JDK and a set of development tools, such as Eclipse and Maven, which will be your companions on this journey.

In the next chapter, we will summarize all the application server features with a special focus on the components and commands needed to deliver an application, which is the main aim of this book.
Where to buy this book

You can buy Java EE 7 Development with WildFly from the Packt Publishing website. Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers. Click here for ordering and shipping details.