An Oracle White Paper
April 2010

Introduction to Java Platform, Enterprise Edition 6

Executive Overview

Java Platform, Enterprise Edition 6 (Java EE 6) provides new capabilities that make it easier to develop and deploy enterprise and Web applications. It provides a simplified developer experience; embraces innovative open source frameworks; offers a comprehensive Web profile for lightweight, standards-based Web applications; and begins the formal process of pruning outdated and unused technology from the platform.

Introduction

Over the years, Java EE has gracefully evolved as an enterprise application deployment platform focused on robustness, Web services, and ease of deployment. Continually shaped by users through the Java Community Process (JCP), Java EE represents a universal standard in enterprise IT, facilitating the development, deployment, and management of multitier, server-centric applications. Whereas Java EE 5 focused on increasing developer efficiency with the introduction of annotations, the Enterprise JavaBeans (EJB) 3.0 business component development model, new and updated Web services, and improvements to the persistence model, Java EE 6 is further streamlining the development process and making the platform even more flexible so that it can better address lightweight Web applications. Java EE 6 highlights new technologies, embraces open source frameworks, and begins the process of pruning away old technologies.

- In the years since its introduction, Java 2 Platform, Enterprise Edition (J2EE) has grown to address new technologies and methodologies. By increasing flexibility, the full platform continues to address enterprise applications while simplifying the development and deployment of applications that are more lightweight and focused. Recognizing that the platform can be too big for specific tasks, this rightsizing effort creates a more flexible and lightweight development and deployment environment focused on target applications and environments.

- Profiles contribute to the flexibility of Java EE 6. The idea is to decouple specifications to enable combinations suited for different use cases. For example, the Web Profile brings together just the technology needed for Web applications. Additional profiles being considered would be created via the JCP.

- Similar to deprecation, pruning removes outdated or unused technologies such as EJB Entity Beans, Java API for XML Registries (JAXR), and others. Pruning provides an orderly way to alert organizations that a technology will be removed from the specification in future releases.

- Extending into open source libraries and frameworks, Java EE 6 embraces open source innovation by enabling zero-configuration “drag and drop” of frameworks into the container or application.

- To ease application development and deployment, Java EE 6 extends the use of annotations—introduced in Java EE 5—throughout various specifications in the platform. In addition, applications developed and deployed in Java EE 6 typically require less effort to configure than in earlier versions.

Java EE 6 continues to deliver the benefits of standards, transparency, and community participation—while providing flexibility, implementation choice, and investment protection. This white paper provides a technical overview of Java EE 6 and its focus on improving developer productivity, simplifying the platform, and increasing modularity and extensibility.

Introducing Java Platform, Enterprise Edition 6

Today's developers recognize the need for distributed, transactional, and portable applications that leverage the speed, security, and reliability of server-side technology. Enterprise applications must be designed, built, and produced for less money while still providing greater speed and more resources. With Java EE, development of Java enterprise applications has never been easier or faster.

Java EE provides a set of well-integrated technologies that significantly reduce the cost and complexity of developing, deploying, and managing multitier, server-centric applications. Building on Java Platform, Standard Edition (Java SE), Java EE adds capabilities for creating complete, stable, secure, and high-performance Java applications for the enterprise. Figure 1 illustrates the primary focuses of the major Java EE releases, including Java EE 6, with its simpler and more flexible development and deployment and its introduction of profiles.

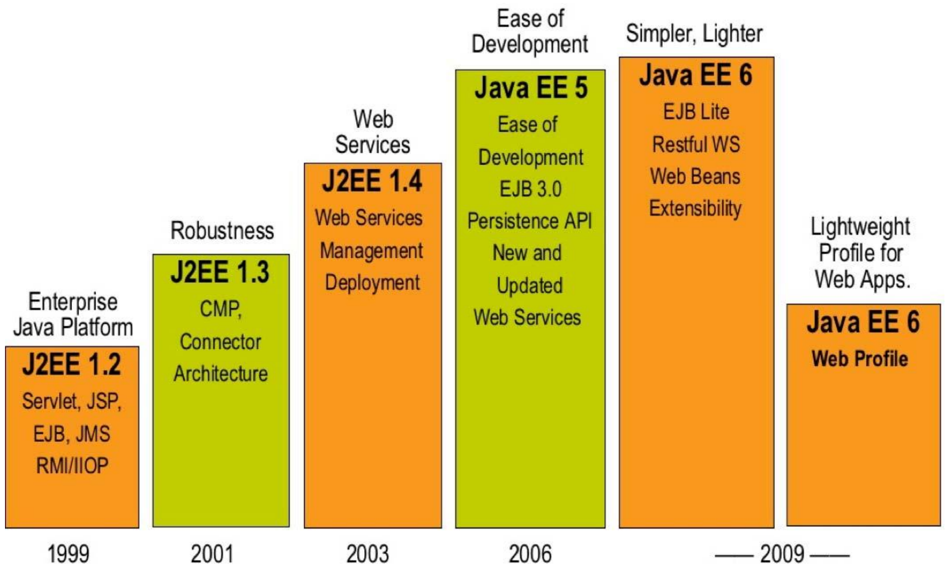


Figure 1. The primary focus of Java EE 6 is to provide a more flexible environment.

The Java EE platform provides developers with a powerful set of APIs that can reduce development time and application complexity while improving application performance. This paper discusses the capabilities that make Java EE 6 a more flexible development and deployment platform and how it can apply a focused set of technologies to an application scenario. This includes profiles, pruning, new ease-of-development capabilities, and other new functionality.

Flexibility

As a robust and mature platform for deploying enterprise applications, Java EE 6 delivers new capabilities that enable it to be a better fit in more situations. Capabilities that contribute to flexibility include profiles and pruning. Profiles enable standards-compliant and more focused solutions to be created via the JCP process. Pruning is an orderly mechanism for reducing outdated and less relevant technologies and thereby simplifying the platform.

Profiles

For many applications, the full set of Java EE APIs may not be appropriate. Profiles enable the deployment of a standardized set of Java EE platform technologies targeting particular classes of applications—for example, aligned with an industry vertical or a frequently occurring use case. The concept of profiles is not new: Java EE leverages experience gained from various sources such as Java Platform, Micro Edition (Java ME), where profiles are applied to specific device runtime environments, and Web service standards such as WS-I Basic Profile.

Java EE 6 introduces the Web Profile, which is designed for modern Web application development. The Web Profile provides transaction processing, security, and persistence management for creating small to medium-size Web applications. As illustrated on the left side of Figure 2, developers often create Web applications by starting with a Web container and then adding technology such as data persistence. As technology is added or updated, it's configured and debugged—a process that's repeated until the required functionality is attained. The result is multiple, nonstandard solution stacks that are difficult to create, integrate, and maintain. In addition, it may become difficult and cost-prohibitive to purchase support on all of the individual technology components.

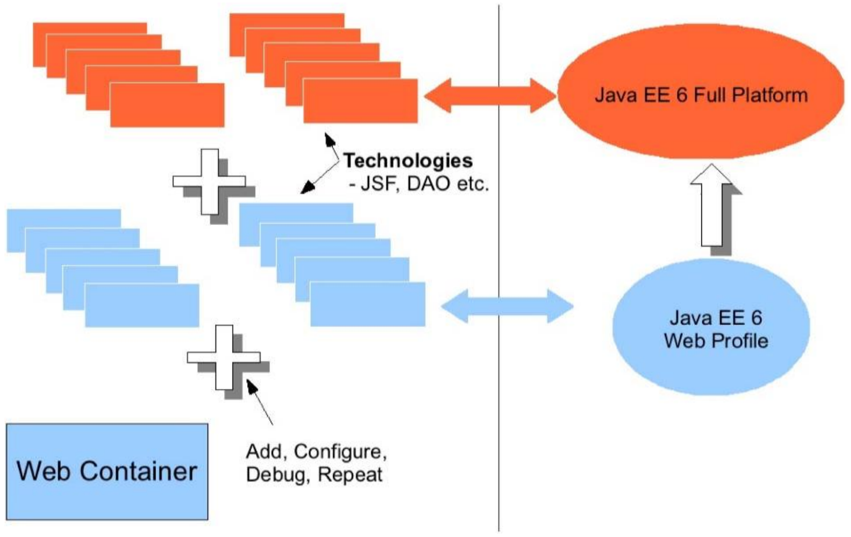


Figure 2: The Web Profile provides everything needed for developing database-driven Web applications.

These concerns are mitigated with Java EE 6. The Web Profile is designed to be a reasonably complete, out-of-the-box platform—composed of standard APIs—that meets the needs of most Web applications. It provides a stable, standard foundation that can be enhanced with innovative open source frameworks and additional technology. As needs change, it's easy to move from the Web Profile to the full platform. Other profiles are expected to be developed and approved, and they can be subsets or supersets of the full Java EE 6 platform. Table 1 identifies the features that make up the Web Profile.

TABLE 1. JAVA EE 6 WEB PROFILE AND JAVA EE 6 FULL PLATFORM

JAVA EE 6 WEB PROFILE	JAVA EE 6 FULL PLATFORM
Servlet 3.0	Servlet 3.0
JavaServer Pages (JSP) 2.2	JavaServer Pages (JSP) 2.2
Expression Language (EL) 2.2	Expression Language (EL) 2.2
Debugging Support for Other Languages (JSR 045)	Debugging Support for Other Languages (JSR 045)
Standard Tag Library for JavaServer Pages (JSTL) 1.2	Standard Tag Library for JavaServer Pages (JSTL) 1.2
JavaServer Faces (JSF) 2.0	JavaServer Faces (JSF) 2.0
Common Annotations for Java Platform 1.1 (JSR 250)	Common Annotations for Java Platform 1.1 (JSR 250)
EJB Lite	• EJB 3.1 • EJB 3.1 Lite
Java Transaction API (JTA) 1.1	Java Transaction API (JTA) 1.1
Java Persistence API (JPA) 2.0	Java Persistence API (JPA) 2.0
Bean Validation 1.0	Bean Validation 1.0
Managed Beans 1.0	Managed Beans 1.0
Interceptors 1.1	Interceptors 1.1
Contexts and Dependency Injection for the Java EE Platform 1.0 (JSR 299)	Contexts and Dependency Injection for the Java EE Platform 1.0 (JSR 299)
Dependency Injection for Java 1.0 (JSR 330)	• Dependency Injection for Java 1.0 (JSR 330)JMS1.1 • JavaMail 1.4 • Connector 1.6 (JAX-WS, JAX-RS 1.1, JAX-RPC 1.1, JAXB 2.2, and JAXR 1.0) • Java EE Management 1.1 • Java EE Deployment 1.2 • JACC 1.3 • JASPIC 1.0 • JSP Debugging 1.0 • Web Services Metadata 2.0 • SAAJ 1.3 • Web Services 1.3

Pruning

Pruning a specification means that it can become an optional (rather than a required) component in the next release of the platform. By carefully pruning APIs that are outdated, not well supported, or not widely deployed, Java EE can achieve a more lightweight platform that's easier to use and provides room for enhancements and refinements. Pruning is marked in the JavaDoc, indicating that a specification is a candidate for becoming optional. More specifically, this means that the identified technologies may be optional in the next release and possibly deleted in the following release, although vendors may continue to offer and support pruned specifications.

Candidates for pruning are recommended by the Java EE Expert Group. Current specifications in the process of being pruned include the following:

- **Java APIs for XML-based RPC (JAX-RPC, JSR 101).** This API was effectively superseded by JAX-WS with the release of Java EE 5.
- **EJB Entity Beans (defined as part of JSR 153: Enterprise JavaBeans 2.0, and earlier)**. Replaced by Java Persistence API (JPA).
- **Java API for XML Registries (JAXR, JSR 93).** JAXR has not been superseded but has seen very limited adoption.
- **Java EE Application Deployment (JSR 88)**. This has not been widely adopted by vendors.

Development Productivity and Extensibility

Java EE 6 advances the ease-of-development features introduced in Java EE 5—improving developer productivity by simplifying application development. The heavy use of annotations in Java EE 6 results in more "plain old Java objects" (POJOs) and far less configuration, resulting in fewer developer artifacts to create, manage, and maintain. Ease-of-development features in Java EE 6 include Web fragments in servlets, EJB Lite, annotations in Connectors, and simplifies application packaging with the ability to directly embed EJB components in Web modules. These and other enhanced features provide an easier development environment that significantly reduces development time and increases developer productivity.

Several new capabilities in Java EE 6 enable developers to leverage the innovation coming out of the open source community. One example of extensibility and ease of use is the zero-configuration, drag-and-drop design methodology for Web frameworks. Within the Java EE 6 release, servlets, servlet filters, and context listeners are now discovered and registered automatically. By providing features that are easier to use, such as plug-in library Java Archive (JAR) files that contain Web fragments, the user community will be even more productive by simplifying application configuration and reducing configuration errors.

The following subsections present a high-level description of some of the major Java EE 6 technologies and innovations that contribute to developer productivity and platform extensibility.

Servlet 3.0

The Servlet 3.0 specification is a major focus of Java EE 6. In Java EE 5, servlet classes and the accompanying web.xml file (which includes configuration metadata) were needed. In Servlet 3.0, annotations are used to declare servlets—which means that web.xml files are no longer required. Servlet 3.0 makes it easier to add third-party libraries without any web.xml changes. (Note that web.xml is optional and can still be used where desirable.) For example, JavaServer Faces (JSF) 2.0 is integrated into Oracle GlassFish Server 3 through the use of this capability. Developers can continue to use other annotations related to security and dependency injection from the servlet or the filter.

Web Fragments

Servlet 3.0 offers a Web fragments descriptor for specifying the details of each library used by the container. Web fragments provide modularity, enabling logical partitioning of the web.xml deployment descriptor so that frameworks such as JSF, IceFaces 2.0, and Atmosphere can have their information added to the JAR file. Framework authors do not have to edit web.xml; instead, they can predefine configuration information with the framework itself.

There can be multiple Web fragments, and the set can be logically viewed as an entire web.xml file. This logical partitioning of the web.xml file enables containers to autoregister Web frameworks. Each Web framework used in a Web application can define in a Web fragment all the artifacts—such as servlets and listeners—it needs without requiring additional edits to the web.xml file.

Asynchronous Processing

Servlet 3.0 introduces support for asynchronous processing, avoiding a thread blocking request while waiting for a response from a database or message connection. With this support, a servlet no longer has to wait for a response from a resource, such as a database, before its thread can continue processing. Developers can use annotations to mark servlets as asynchronous. Support for asynchronous processing makes using servlets with Ajax more efficient and improves the ability to scale application load with minimal server resources.

JAX-RS 1.1 (JSR 311)

The Java API for RESTful Web Services (JAX-RS) 1.1 specification provides a means by which URIs can be used to access and manipulate Web resources (including data and functionality) from anywhere on the internet. Using familiar commands—`GET`, `PUT`, `POST`, and `DELETE`—this technology enables clients and servers to communicate and exchange representations of these resources.

RESTful Web services are built in the REST architectural style. Building Web services with the RESTful approach has emerged as a popular alternative to using SOAP-based technologies, due to its lightweight nature and much simpler interface. For example, Oracle GlassFish Server 3 exposes its administration and monitoring capabilities as a RESTful service, enabling developers to create innovative services.

EJB 3.1 (JSR 318)

The EJB 3.0 specification (part of Java EE 5) simplified the use of EJB technology. The latest release of the technology—JSR 318: Enterprise JavaBeans 3.1 (available in Java EE 6)—takes that simplification even further, affording many improvements that reflect common usage patterns. These include

- **Singletons.** A singleton is a session bean that's instantiated once per application and can be shared among multiple components within that application. Singleton beans provide the ability to easily share state between multiple instances of an enterprise bean component or between multiple enterprise bean components in an application. Singleton beans enable concurrent access, which can be managed with synchronization or by the container. Singleton beans are like any other EJB components—they are POJOs that developers can mark as singleton beans, using annotations.
- **No-interface view.** This enables the specification of an enterprise bean with only a bean class, without the need to write a local business interface. A local interface defines the business methods that are exposed to the client and implemented on the bean class. The no-interface view has the same behavior as the EJB 3.0 local view (for example, it supports other semantics such as pass-by reference and transaction and security propagation). This interface is now optional, which means that the enterprise bean can provide the same functionality without the need to write a separate business interface.
- **Java Naming and Directory Interface (JNDI).** This interface provides a means of globally locating EJB components. Java EE 6 specifies portable global JNDI names, improving EJB portability across multiple implementations.
- **Asynchronous session bean.** A session bean can be annotated to support asynchronous method invocations. Prior to EJB 3.1, method invocations on session beans were always synchronous.
- **Embeddable API.** EJB 3.1 provides an embeddable API and container for use in the Java SE environment (the same Java Virtual Machine [JVM]). This enables easier testing of EJB components outside of a Java EE container.
- **Timer service.** The EJB 3.1 architecture enhances the EJB timer service, which enables cron-like scheduling (both declarative and programmatic) for all types of enterprise beans (except for stateful session beans). The server is implemented with annotations.

EJB Lite

EJB component technology provides robust functionality—so robust, in fact, that it's more than some applications need. EJB Lite is designed to meet the needs of applications that require only a subset of the features provided by EJB technology. As a standardized subset of the EJB 3.1 API, EJB Lite is neither a one-off product nor an implementation. Applications developed with EJB Lite can run in application servers that implement either the Web Profile or the full Java EE 6 platform.

Developers can use EJB Lite to simplify the development of secure, transactional business components without having to master the full power of EJB up front. Vendor implementations of EJB Lite are expected to be simpler and more focused than full EJB component implementations. EJB Lite is also part of the Web Profile. See Table 2 for a comparison.

TABLE 2. EJB LITE AND EJB 3.1 FEATURES COMPARISON

FEATURE	EJB LITE	EJB 3.1
Local session beans	X	X
Declarative security	X	X
Transactions (CMT/BMT)	X	X
Interceptors	X	X
Security	X	X
Message-driven beans		X
RMI/IIOP interoperability and remote view		X
EJB 2.x backward compatibility		X
Time service		X
Asynchronous method invocations		X

Java Persistence API (JPA) 2.0 (JSR 317)

Persistence is the technique through which object models broker the access and manipulation of information from a relational database. JPA handles the details of how relational data is mapped to Java objects, and it standardizes object-relational mapping. Introduced in Java EE 5, JPA provided a POJO-based persistence model for Java EE and Java SE applications. JPA has been enhanced in Java EE 6, facilitating methodology that is more effective and reliable (that is, more strongly typed) for building object-centric, criteria-based dynamic database queries.

- **Object-relational mapping enhancements.** For example, JPA 2.0 adds the ability to map collections of basic datatypes, such as strings or integers, as well as collections of embeddable objects. New annotations support collection mappings.
- **Java Persistence Query Language (JPQL) enhancements.** These include new operators such as NULLIF, VALUE, and others, and case expressions can be used in queries.

- **Criteria API.** Based on the metamodel concept, this type-safe query mechanism can verify persistent Java objects. It can be statically or dynamically accessed. A string-based model is also available, but it's not as type-safe.

Contexts and Dependency Injection (CDI) for Java EE (JSR 299)

Dependency injection can be applied to all the resources a component needs, effectively hiding the creation and lookup of resources from application code. It enables the Java EE container to automatically insert references to other required components or resources, using annotations. Dependency injection can be used in EJB containers, Web containers, and application clients.

Previously known as Web Beans, Context and Dependency Injection (CDI) adds dependency injection and context management facilities to the existing Java EE component models. This functionality enables developers to use plain JavaBeans, session beans, and JSF managed beans interchangeably to hold state in their applications. Moreover, each bean can be attached to a particular scope, such as a single request, an HTTP session, or a conversation. The CDI runtime will then ensure that beans are created, injected, and destroyed at the appropriate time within the lifecycle of the application, in accordance with the scope they are attached to, resulting in simpler, error-free state management. Beans also have the ability to fire and observe events, enabling components within a single application to interact in a more loosely coupled way than was previously possible.

CDI brings transactional support to the Web tier, making it easier to access transactional resources in Web applications. For example, CDI services facilitate building a Java EE Web application that accesses a database with persistence provided by the JPA.

Although dependency injection has become an increasingly popular technique for developing enterprise Java applications, there was no standardized approach until recently for applying the technique. Dependency Injection for Java (JSR 330) changes that, introducing a standard set of annotations that can be used for dependency injection.

JavaServer Faces (JSF) 2.0 (JSR 314)

JSF technology, the server-side component framework designed to simplify the development of user interfaces for Java EE applications, has been simplified and improved, especially in the area of page authoring. Facelets is a powerful but lightweight page declaration language for designing JSF views, using HTML-style templates and building component trees. Facelets also enables code reuse through templating, significantly reducing the time required to develop and deploy user interfaces.

Templating is used to create a page that acts as a template for other pages in an application. This saves on labor while maintaining a standard look and feel in an application with many pages. Facelets views are usually written with the XHTML markup language, which makes them more portable across development platforms.

An example of annotation support in JSF 2.0 is the simplified approach to configuring managed beans. Instead of registering a managed bean by configuring it in the JSF configuration file (faces-config.xml), all that is necessary is to mark the bean and set the scope set with the appropriate annotations.

Bean Validation 1.0 (JSR 303)

Data validation is a common task that occurs throughout an application—from the presentation layer to the persistence layer. For example, validation logic can be applied to an entry field, ensuring that a number is within a certain range. Often the same validation logic is implemented within each layer, in a time-consuming and error-prone process. To avoid duplicating these validations in each layer, developers often bundle validation logic directly into the domain model, cluttering domain classes with validation code that is, in effect, metadata about the class itself.

JSR 303 defines a metadata model and an API to provide JavaBeans validation. Bean validation is integrated across the Java EE 6 platform. For example, presentation layer technologies such as JSF and enterprise layer technologies such as JPA have access to the constraint definitions and validators provided through the Bean Validation framework.

The default metadata syntax uses annotations, with the ability to override and extend the metadata through the use of XML validation descriptors. The validation API described within JSR 303 is specifically not intended to be limited to any particular tier of the multitiered programming model. As such, the functionality offered by the validation API is available for both server-side application programming and rich-client Swing application developers. Seen as a general extension to the JavaBeans object model, this API is expected to be used as a core component within other specifications. Ease of use and flexibility were the guiding philosophies for the development of this specification.

Connector Architecture 1.6 (JSR 322)

For enterprise application integration, bidirectional connectivity between enterprise applications and an enterprise information system (EIS) is essential. The Java EE Connector Architecture defines a standard set of contracts that allow bidirectional connectivity between enterprise applications and EIS providers. It also formalizes the relationships, interactions, and packaging of the integration layer, thus enabling unfettered enterprise application integration. If an application server vendor extends its system once to support the connector architecture, the application server vendor is assured of being able to seamlessly connect to multiple off-the-shelf EIS offerings. Likewise, if an EIS vendor provides an industry-standard resource adapter (such as is described within JSR 322), the EIS vendor will be assured of being able to plug in to any application server that supports the Connector Architecture.

The Connector 1.6 specification (JSR 322) enhances the Connector Architecture through each of the following means:

- **Ease of development.** The specification dramatically simplifies the development of resource adapters through extensive use of Java language annotations, reducing the need to write redundant code as well as the need for a deployment descriptor (ra.xml). It also provides better configuration defaults.

- **Generic Work Context mechanism.** The specification defines a generic mechanism for propagating contextual information during Work execution. It also standardizes passing in security, transactional, and other quality-of-service parameters from an EIS to a Java EE component during Work execution.
- **Security Work Context.** The specification defines a standard contract that a resource adapter can employ to establish security information (established identity) while submitting a Work instance for execution to a WorkManager and while delivering messages to message endpoints (or message-driven beans) residing in the application server.

Packaging

Packaging and deployment are much easier in Java EE 6 than in previous versions. In the example in Figure 3, the left side represents what's needed to develop and deploy under Java EE 5: the web.xml (for configuration metadata and presentation code) is packaged in BuyBooks.war, the business logic is packaged in ShoppingCart.jar, and everything is then packaged up in the BuyBooks.ear file.

This arrangement is greatly simplified under Java EE 6, as shown in the shopping cart example on the right side of Figure 3. As you can see from the illustration, EJB components can be packaged directly in a .war file; however, the traditional method of supporting EJB files is also supported.

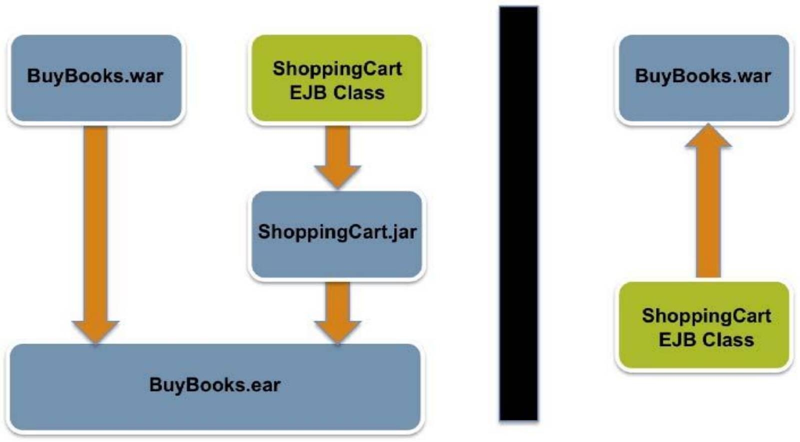


Figure 3. Packaging and deploying applications is easier under Java EE 6 (on the right). EJB class files can be packaged as part of the .war file.

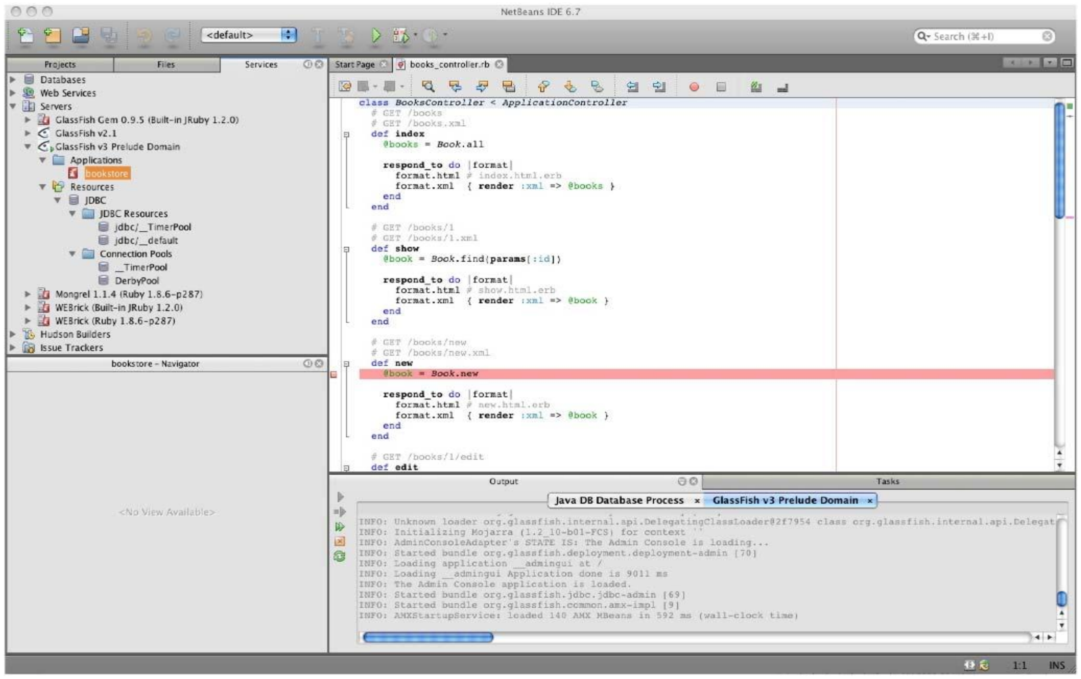
Oracle GlassFish Server 3

Oracle GlassFish Server 3 is the first compatible, production implementation of the Java EE 6 platform specification. Many of the Java EE 6 themes—including flexibility, extensibility, and developer ease of use—carry over to Oracle GlassFish Server 3. This lightweight, flexible, and open source application server enables organizations to not only leverage the new capabilities introduced in the Java EE 6 specification but also to add to their existing capabilities through a faster and more streamlined development and deployment cycle. Notable features of the Oracle GlassFish Server 3 release include a modular architecture based on OSGi for improved startup time, reduced resource utilization, rapid iterative development, and fine-grained monitoring. Oracle GlassFish Server 3 also delivers support for third-party dynamic languages such as JRuby, Jython, and Groovy.

Oracle GlassFish Server 3 has significantly improved application productivity beyond the advances in Java EE itself. Now, because of support for application redeployment without loss of application state, instead of six time-consuming steps (edit, save code, compile, package, deploy, and repopulate session data), the same process on GlassFish requires just three (edit, save, and refresh the browser) in integrated development environments (IDEs) such as Eclipse and NetBeans. Strong support and tight integration with Oracle GlassFish Server 3 improve ease of use and productivity.

Integrated Development Environments

Leading IDEs such as NetBeans and Eclipse can be used to develop applications and other components for Java EE 6. These leading IDEs support virtually all the Java EE 6 capabilities described in this paper, such as singletons, Web fragments, and Facelets. NetBeans provides comprehensive support of the Java EE 6 platform and its reference implementation—Oracle GlassFish Server 3—as shown in Figure 4.



```
class BooksController < ApplicationController
  # GET /books
  # GET /books.xml
  def index
    @books = Book.all

    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @books }
    end
  end

  # GET /books/1
  # GET /books/1.xml
  def show
    @book = Book.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.xml  { render :xml => @book }
    end
  end

  # GET /books/new
  # GET /books/new.xml
  def new
    @book = Book.new

    respond_to do |format|
      format.html # new.html.erb
      format.xml  { render :xml => @book }
    end
  end

  # GET /books/1/edit
  def edit
```

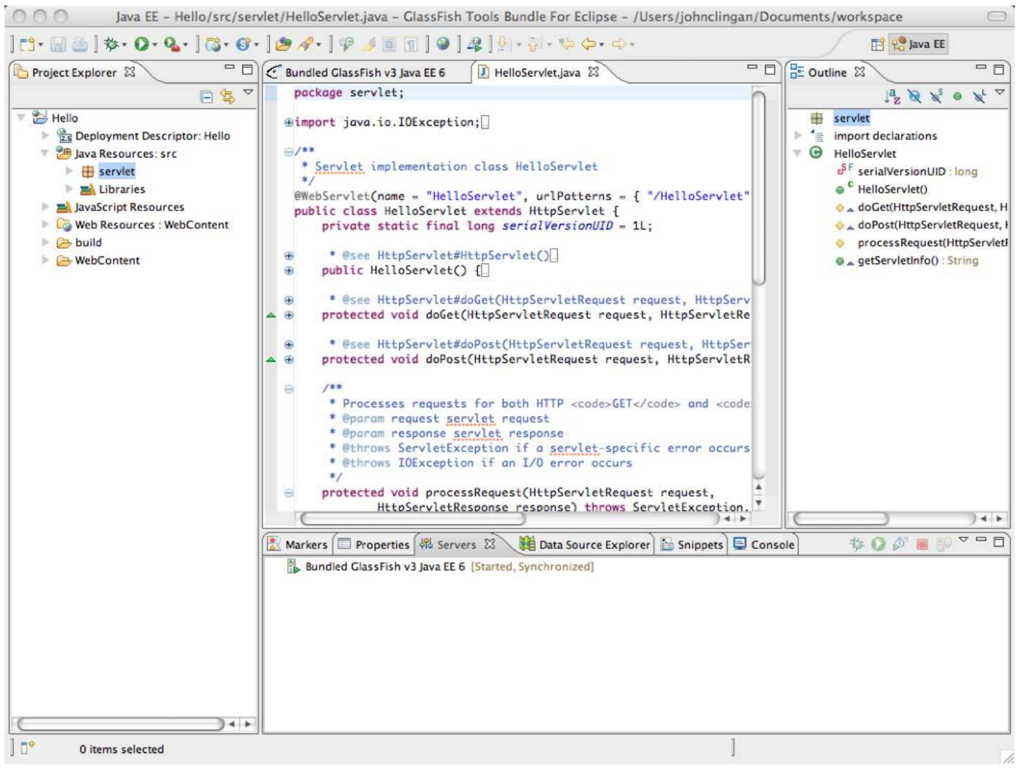
```
INFO: Unknown loader org.glassfish.internal.api.DelegatingClassLoader@2f7954 class org.glassfish.internal.api.Delegat
INFO: Initializing Mojarra (1.2_10-b01-FCS) for context ''
INFO: AdminConsoleAdapter's STATE IS: The Admin Console is loading...
INFO: Started bundle org.glassfish.deployment.deployment-admin [70]
INFO: Loading application __admingui at /
INFO: Loading __admingui Application done is 9011 ms
INFO: The Admin Console application is loaded.
INFO: Started bundle org.glassfish.jdbc.jdbc-admin [69]
INFO: Started bundle org.glassfish.common.amx-impl [9]
INFO: AMXStartupService: loaded 140 AMX MBeans in 592 ms (wall-clock time)
```

Figure 4. The NetBeans IDE provides comprehensive support for Java EE 6 and Oracle GlassFish Server 3 and later

For example, NetBeans automatically determines which versions of the subspecifications to use, based on what the target environment (including Java EE and application server release levels) supports.

Some Java EE 6 technologies can be used in Java EE 5 application servers—for example, JSF 2.0 can be used with Java EE 5 projects and JAX-RS 1.0 and NetBeans allows this option as appropriate.

The community-supported GlassFish Tools Bundle for Eclipse shown in Figure 5, enables developers to build applications with the Eclipse IDE and deploy them to Oracle GlassFish Server. This distribution contains a comprehensive environment for using Eclipse to develop Java EE applications for deployment in the Oracle GlassFish Server environment. Oracle GlassFish Server is bundled and preconfigured with the distribution.


Java EE – Hello/src/servlet/HelloServlet.java – GlassFish Tools Bundle For Eclipse – /Users/johnclingan/Documents/workspace
Java EE
Project Explorer
Hello
Deployment Descriptor: Hello
Java Resources: src
servlet
Libraries
JavaScript Resources
Web Resources : WebContent
build
WebContent
Bundled GlassFish v3 Java EE 6
HelloServlet.java
Outline
servlet
import declarations
HelloServlet
serialVersionUID : long
HelloServlet()
doGet(HttpServletRequest, H
doPost(HttpServletRequest, H
processRequest(HttpServletR
getServletInfo() : String
Markers
Properties
Servers
Data Source Explorer
Snippets
Console
0 items selected

```
package servlet;

import java.io.IOException;

/**
 * Servlet implementation class HelloServlet
 */
@WebServlet(name = "HelloServlet", urlPatterns = { "/HelloServlet"
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

     * @see HttpServlet#HttpServlet()
    public HelloServlet() {

     * @see HttpServlet#doGet(HttpServletRequest request, HttpServ
    protected void doGet(HttpServletRequest request, HttpServletRe

     * @see HttpServlet#doPost(HttpServletRequest request, HttpSer
    protected void doPost(HttpServletRequest request, HttpServletR

    /**
     * Processes requests for both HTTP <code>GET</code> and <code
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,
```

```
Bundled GlassFish v3 Java EE 6 [Started, Synchronized]
```

Figure 5. With GlassFish Tools Bundle for Eclipse, developers can build applications with the Eclipse IDE and deploy them to Oracle GlassFish Server.

Conclusion

After 10 years of running business-critical applications for thousands of organizations, Java EE continues to make tremendous progress as an enterprise application development and deployment platform. It delivers a comprehensive set of APIs that can simplify the developer experience while improving productivity. Annotation-based programming and zero-configuration Web frameworks that leverage open source innovation further simplify the developer environment while extending the platform. Profiles create focused architectures of standards-based EJB components, and pruning clears out unused technologies. Technical enhancements include Servlet 3.0, Facelets, RESTful Web services, Connectors, an enhanced persistence architecture, dependency injection, and much more. Developing enterprise applications with Java has never been easier.

Appendix 1: References

- JSR 045: Debugging Support for Other Languages
- JSR 052: Standard Tag Library for JavaServer Pages 1.2
- JSR 109: Implementing Enterprise Web Services 1.1
- JSR 196: Authentication Service Provider Interface 1.1
- JSR 222: Java Architecture for XML Binding (JAXB) 2.1
- JSR 224: Java API for XML Web-Based Services (JAX-WS) 2.2
- JSR 245: JavaServer Pages (JSP) 2.1
- JSR 250: Common Annotations for the Java Platform
- JSR 252: JavaServer Faces (JSF) 1.2
- JSR 299: Contexts and Dependency Injection for the Java EE Platform
- JSR 303: Bean Validation 1.0
- JSR 311: The Java API for RESTful Web Services (JAX-RS)
- JSR 314: JavaServer Faces 2.0
- JSR 315: Java Servlet 3.0
- JSR 317: Java Persistence API (JPA) 2.0
- JSR 318: Enterprise JavaBeans (EJB) 3.1
- JSR 322: Java EE Connector Architecture 1.6
- JSR 907: Java Transaction API (JTA) 1.1

Oracle is committed to developing practices and products that help protect the environment

Introduction to Java Platform,
Enterprise Edition 6
April 2010

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com