



★ Google App Engine

Developer's Guide

Welcome to Google App Engine! This developer's guide contains everything you need to know to build scalable web applications using Google technology.

You can develop your Google App Engine application for either of two application environments:

Java

- a complete Java 6 runtime environment in a secure sandbox environment
- based on common Java web technology standards, including servlets and WARs, JDO and JPA, java.net, JavaMail and JCache
- a [plugin for the Eclipse IDE](#) makes project creation, testing and deployment a snap
- supports other languages that compile to the JVM or use JVM-based interpreters, such as JRuby, JavaScript (Rhino), and Scala

[Get started with Java!](#)

Python

- a fast Python 2.5 interpreter in a secure sandbox environment
- includes the complete Python standard library
- compiled application code is cached for rapid responses to web requests
- supports popular web application frameworks, including [Django](#)
- works with any application that supports CGI or WSGI
- includes a rich data modeling API for the datastore

[Get started with Python!](#)

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



★ Google App Engine

Introduction

Welcome to Google App Engine! Creating an App Engine application is easy, and only takes a few minutes. And it's free to start: upload your app and share it with users right away, at no charge and with no commitment required.

Google App Engine applications can be written in either the Java or Python programming languages. This tutorial covers **Java**. If you're more likely to use Python to build your applications, see [Getting Started: Python](#).

In this tutorial, you will learn how to:

- build an App Engine application using standard Java web technologies, such as servlets and JSPs
- create an App Engine Java project with [Eclipse](#), and without
- use the Google Plugin for Eclipse for App Engine development
- use the App Engine datastore with the [Java Data Objects](#) (JDO) standard interface
- integrate an App Engine application with Google Accounts for user authentication
- upload your app to App Engine

By the end of the tutorial, you will have implemented a working application, a simple guest book that lets users post messages to a public message board.

Next...

To get started developing Google App Engine Java applications, you download and set up the App Engine Java software development kit and related components.

Continue to [Installing the Java SDK](#).

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



★ **Google App Engine**

Installing the Java SDK

You develop and upload Java applications for Google App Engine using the App Engine Java software development kit (SDK).

The SDK includes software for a web server that you can run on your own computer to test your Java applications. The server simulates all of the App Engine services, including a local version of the datastore, Google Accounts, and the ability to fetch URLs and send email from your computer using the App Engine APIs.

Getting Java

Google App Engine supports Java 5 and Java 6. When your Java application is running on App Engine, it runs using the Java 6 virtual machine (JVM) and standard libraries. Ideally, you should use Java 6 for compiling and testing your application to ensure that the local server behaves similarly to App Engine.

For developers that don't have easy access to Java 6 (such as developers using Mac OS X), the App Engine SDK is compatible with Java 5. You can upload compiled classes and JARs made with Java 5 to App Engine.

If necessary, [download and install the Java SE Development Kit \(JDK\) for your platform](#). Mac users, see [Apple's Java developer site](#) to download and install the latest version of the Java Developer Kit available for Mac OS X.

Once the JDK is installed, run the following commands from a command prompt (for Windows, Command Prompt; for Mac OS X, Terminal) to verify that you can run the commands, and to determine which version is installed. If you have Java 6 installed, these commands will report a version number similar to `1.6.0`. If you have Java 5 installed, the version number will be similar to `1.5.0`.

```
java -version  
  
javac -version
```

Using Eclipse and the Google Plugin for Eclipse

If you are using [the Eclipse development environment](#), the easiest way to develop, test and upload App Engine apps is to use the [Google Plugin for Eclipse](#). The plugin includes everything you need to build, test and deploy your app, entirely within Eclipse.

The plugin is available for Eclipse versions 3.3, 3.4, and 3.5. You can install the plugin using the Software Update feature of Eclipse. The installation locations are as follows:

- The Google Plugin for Eclipse, for Eclipse 3.3 (Europa):

```
http://dl.google.com/eclipse/plugin/3.3
```

- The Google Plugin for Eclipse, for Eclipse 3.4 (Ganymede):

```
http://dl.google.com/eclipse/plugin/3.4
```

- The Google Plugin for Eclipse, for Eclipse 3.5 (Galileo):

```
http://dl.google.com/eclipse/plugin/3.5
```

For details on how to use Software Update to install the plugin, and how to create a new project, see [Using the Google Eclipse Plugin](#).

Getting the SDK

If you are using Eclipse and the Google Plugin, you can install the App Engine SDK from Eclipse using Software Update. If you haven't already, install the "Google App Engine Java SDK" component using the locations above.

If you are not using Eclipse or the Google Plugin, you can download the App Engine Java SDK as a Zip archive.

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



★ **Google App Engine**

Creating a Project


App Engine Java applications use [the Java Servlet standard](#) for interacting with the web server environment. An application's files, including compiled classes, JARs, static files and configuration files, are arranged in a directory structure using the [WAR](#) standard layout for Java web applications. You can use any development process you like to develop web servlets and produce a WAR directory. (WAR archive files are not yet supported by the SDK.)

The Project Directory

For this tutorial, we will use a single directory named `Guestbook/` for all project files. A subdirectory named `src/` contains the Java source code, and a subdirectory named `war/` contains the complete application arranged in the WAR format. Our build process compiles the Java source files and puts the compiled classes in the appropriate location in `war/`.

The complete project directory looks like this:

```
Guestbook/
  src/
    ...Java source code...
    META-INF/
    ...other configuration...
  war/
    ...JSPs, images, data files...
    WEB-INF/
    ...app configuration...
    lib/
    ...JARs for libraries...
    classes/
    ...compiled classes...
```

If you are using Eclipse, create a new project by clicking the New Web Application Project button in the toolbar:  Give the project a "Project name" of `Guestbook` and a "Package" of `guestbook`. Uncheck "Use Google Web Toolkit," and ensure "Use Google App Engine" is checked. See [Using the Google Plugin for Eclipse](#) for more information. The wizard creates the directory structure, and the files described below.

If you are not using Eclipse, create the directory structure described above. As you read each of the files described in this section, create the files using the given locations and names.

You can also copy the new project template included with the SDK, in the `appengine-java-sdk/demos/new_project_template/` directory.

The Servlet Class

App Engine Java applications use [the Java Servlet API](#) to interact with the web server. An HTTP servlet is an application class that can process and respond to web requests. This class extends either the [javax.servlet.GenericServlet](#) class or the [javax.servlet.http.HttpServlet](#) class.

Our guest book project begins with one servlet class, a simple servlet that displays a message.

If you are not using the Eclipse plugin, create the directories for the path `src/guestbook/`, then create the servlet class file described below.

In the directory `src/guestbook/`, a file named `GuestbookServlet.java` has the following contents:

```
package guestbook;

import java.io.IOException;
import javax.servlet.http.*;

public class GuestbookServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException {
        resp.setContentType("text/plain");
```

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



★ **Google App Engine**

Using the Users Service

Google App Engine provides several useful services based on Google infrastructure, accessible by applications using libraries included with the SDK. One such service is the Users service, which lets your application integrate with Google user accounts. With the Users service, your users can use the Google accounts they already have to sign in to your application.

Let's use the Users service to personalize this application's greeting.

Using Users

Edit `src/guestbook/GuestbookServlet.java` as indicated to resemble the following:

```
package guestbook;

import java.io.IOException;
import javax.servlet.http.*;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

public class GuestbookServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException {
        UserService userService = UserServiceFactory.getUserService();
        User user = userService.getCurrentUser();

        if (user != null) {
            resp.setContentType("text/plain");
            resp.getWriter().println("Hello, " + user.getNickname());
        } else {
            resp.sendRedirect(userService.createLoginURL(req.getRequestURI()));
        }
    }
}
```

If you are using Eclipse and your development server is running in the debugger, when you save your changes to this file, Eclipse compiles the new code automatically, then attempts to insert the new code into the already-running server. Changes to classes, JSPs, static files and `appengine-web.xml` are reflected immediately in the running server without needing to restart. If you change `web.xml` or other configuration files, you must stop and start the server to see the changes.

If you are using Ant, you must stop the server and rebuild the project to see changes made to source code. Changes to JSPs and static files do not require restarting the server.

Rebuild your project and restart the server, if necessary. Test the application by visiting the servlet URL in your browser:

<http://localhost:8888/guestbook>

Instead of displaying the message, the server prompts you for an email address. Enter any email address (such as `alfred@example.com`, then click "Log In." The app displays a message, this time containing the email address you entered.

The new code for the `GuestbookServlet` class uses the Users API to check if the user is signed in with a Google Account. If not, the user is redirected to the Google Accounts sign-in screen. `userService.createLoginURL(...)` returns the URL of the sign-in screen. The sign-in facility knows to redirect the user back to the app by the URL passed to `createLoginURL(...)`, which in this case is the URL of the current page.

The development server knows how to simulate the Google Accounts sign-in facility. When run on your local machine, the redirect goes to the page where you can enter any email address to simulate an account sign-in. When run on App Engine, the redirect goes to the actual Google Accounts screen.

You are now signed in to your test application. If you reload the page, the message will display again.

To allow the user to sign out, provide a link to the sign-out screen, generated by the method `createLogoutURL()`.

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



★ **Google App Engine**

Using JSPs

While we could output the HTML for our user interface directly from the Java servlet code, this would be difficult to maintain as the HTML gets complicated. It's better to use a template system, with the user interface designed and implemented in separate files with placeholders and logic to insert data provided by the application. There are many template systems available for Java, any of which would work with App Engine.

For this tutorial, we'll use [JavaServer Pages](#) (JSPs) to implement the user interface for the guest book. JSPs are part of the servlet standard. App Engine compiles JSP files in the application's WAR automatically, and maps them to URL paths.

Hello, JSP!

Our guest book app writes strings to an output stream, but this could also be written as a JSP. Let's begin by porting the latest version of the example to a JSP.

In the directory `war/`, create a file named `guestbook.jsp` with the following contents:

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ page import="com.google.appengine.api.users.User" %>
<%@ page import="com.google.appengine.api.users.UserService" %>
<%@ page import="com.google.appengine.api.users.UserServiceFactory" %>

<html>
  <body>

  <%
    UserService userService = UserServiceFactory.getUserService();
    User user = userService.getCurrentUser();
    if (user != null) {
  %>
  <p>Hello, <%= user.getNickname() %>! (You can
  <a href="<%= userService.createLogoutURL(request.getRequestURI()) %>">sign out</a>.)
  </p>
  <%
    } else {
  %>
  <p>Hello!
  <a href="<%= userService.createLoginURL(request.getRequestURI()) %>">Sign in</a>
  to include your name with greetings you post.</p>
  <%
    }
  %>

  </body>
</html>
```

By default, any file in `war/` or a subdirectory (other than `WEB-INF/`) whose name ends in `.jsp` is automatically mapped to a URL path. The URL path is the path to the `.jsp` file, including the filename. This JSP will be mapped automatically to the URL `/guestbook.jsp`.

For the guest book app, we want this to be the application's home page, displayed when someone accesses the URL `/`. An easy way to do this is to declare in `web.xml` that `guestbook.jsp` is the "welcome" servlet for that path.

Edit `war/WEB-INF/web.xml` and replace the current `<welcome-file>` element in the `<welcome-file-list>`. Be sure to remove `index.html` from the list, as static files take precedence over JSP and servlets.

```
<welcome-file-list>
  <welcome-file>guestbook.jsp</welcome-file>
</welcome-file-list>
```

Tip: If you are using Eclipse, the editor may open this file in "Design" mode. To edit this file as XML, select the "Source" tab at the bottom of the frame.

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



★ **Google App Engine**

Using the Datastore with JDO

Storing data in a scalable web application can be tricky. A user could be interacting with any of dozens of web servers at a given time, and the user's next request could go to a different web server than the one that handled the previous request. All web servers need to be interacting with data that is also spread out across dozens of machines, possibly in different locations around the world.

With Google App Engine, you don't have to worry about any of that. App Engine's infrastructure takes care of all of the distribution, replication and load balancing of data behind a simple API—and you get a powerful query engine and transactions as well.

The App Engine datastore is one of several services provided by App Engine with two APIs: a standard API, and a low-level API. By using the standard APIs, you make it easier to port your application to other hosting environments and other database technologies, if you ever need to. Standard APIs "decouple" your application from the App Engine services. App Engine services also provide low-level APIs that exposes the service capabilities directly. You can use the low-level APIs to implement new adapter interfaces, or just use the APIs directly in your app.

App Engine includes support for two different API standards for the datastore: [Java Data Objects](#) (JDO) and [Java Persistence API](#) (JPA). These interfaces are provided by [DataNucleus Access Platform](#), an open source implementation of several Java persistence standards, with an adapter for the App Engine datastore.

For the guest book, we'll use the JDO interface to retrieve and post messages left by users.

Setting Up DataNucleus Access Platform

Access Platform needs a configuration file that tells it to use the App Engine datastore as the backend for the JDO implementation. In the final WAR, this file is named `jdoconfig.xml` and resides in the directory `war/WEB-INF/classes/META-INF/`.

If you are using Eclipse, this file has been created for you as `src/META-INF/jdoconfig.xml`. This file is automatically copied into `war/WEB-INF/classes/META-INF/` when you build your project.

If you are not using Eclipse, you can create the directory `war/WEB-INF/classes/META-INF/` directly, or have your build process create it and copy the configuration file from another location. The Ant build script described in [Using Apache Ant](#) copies this file from `src/META-INF/`.

The `jdoconfig.xml` file should have the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<jdoconfig xmlns="http://java.sun.com/xml/ns/jdo/jdoconfig"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://java.sun.com/xml/ns/jdo/jdoconfig">

  <persistence-manager-factory name="transactions-optional">
    <property name="javax.jdo.PersistenceManagerFactoryClass"

value="org.datanucleus.store.appengine.jdo.DatastoreJDOPersistenceManagerFactory"/>
    <property name="javax.jdo.option.ConnectionURL" value="appengine"/>
    <property name="javax.jdo.option.NontransactionalRead" value="true"/>
    <property name="javax.jdo.option.NontransactionalWrite" value="true"/>
    <property name="javax.jdo.option.RetainValues" value="true"/>
    <property name="datanucleus.appengine.autoCreateDatastoreTxns"

value="true"/>
  </persistence-manager-factory>
</jdoconfig>
```

JDO Class Enhancement

When you create your JDO classes, you use Java annotations to describe how instances should be stored in the datastore, and how they should be recreated when retrieved from the datastore. Access Platform connects your data classes to the implementation using a post-compilation processing step, which DataNucleus calls "enhancing" the classes.

If you are using Eclipse and the Google Plugin, the plugin performs the JDO class enhancement step automatically as part of the build process.

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



☆ Google App Engine

Using Static Files

There are many cases where you want to serve static files directly to the web browser. Images, CSS stylesheets, JavaScript code, movies and Flash animations are all typically served directly to the browser. For efficiency, App Engine serves static files from separate servers than those that invoke servlets.

By default, App Engine makes all files in the WAR available as static files except JSPs and files in `WEB-INF/`. Any request for a URL whose path matches a static file serves the file directly to the browser—even if the path also matches a servlet or filter mapping. You can configure which files App Engine treats as static files using the `appengine-web.xml` file.

Let's spruce up our guest book's appearance with a CSS stylesheet. For this example, we will not change the configuration for static files. See [App Configuration](#) for more information on configuring static files and resource files.

A Simple Stylesheet

In the directory `war/`, create a directory named `stylesheets/`. In this directory, create a file named `main.css` with the following contents:

```
body {
    font-family: Verdana, Helvetica, sans-serif;
    background-color: #FFFFCC;
}
```

Edit `war/guestbook.jsp` and insert the following lines just after the `<html>` line at the top:

```
<html>
  <head>
    <link type="text/css" rel="stylesheet" href="/stylesheets/main.css" />
  </head>

  <body>
    ...
  </body>
</html>
```

Visit <http://localhost:8888/>. The new version uses the stylesheet.

Next...

The time has come to reveal your finished application to the world.

Continue to [Uploading Your Application](#).

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).
Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



★ **Google App Engine**

Uploading Your Application

You create and manage applications in App Engine using the Administration Console. Once you have registered an application ID for your application, you upload it to App Engine using either the Eclipse plugin, or a command-line tool in the SDK.

Note: Once you register an application ID, you can delete it, but you can't re-register that same application ID after it has been deleted. You can skip these next steps if you don't want to register an ID at this time.

Registering the Application

You create and manage App Engine web applications from the App Engine Administration Console, at the following URL:

<https://appengine.google.com/>

Sign in to App Engine using your Google account. If you do not have a Google account, you can [create a Google account](#) with an email address and password.

To create a new application, click the "Create an Application" button. Follow the instructions to register an application ID, a name unique to this application. If you elect to use the free appspot.com domain name, the full URL for the application will be `http://application-id.appspot.com/`. You can also purchase a top-level domain name for your app, or use one that you have already registered.


Edit the `appengine-web.xml` file, then change the value of the `<application>` element to be your registered application ID.

Uploading the Application

You can upload your application using Eclipse, or using a command at the command prompt.

Uploading From Eclipse

You can upload your application code and files from within Eclipse using the Google Plugin.

To upload your application from Eclipse, click the App Engine deploy button on the toolbar: 

Enter your Google account username (your email address) and password when prompted, then click the **Upload** button. Eclipse gets the application ID and version information from the `appengine-web.xml` file, and uploads the contents of the `war/` directory.

Uploading Using the Command Prompt

You can upload your application code and files using a command included in the SDK named `appcfg.cmd` (Windows) or `appcfg.sh` (Mac OS X, Linux).

AppCfg is a multi-purpose tool for interacting with your app on App Engine. The command takes the name of an action, the path to your app's `war/` directory, and other options. To upload the app code and files to App Engine, you use the `update` action.

To upload the app, using Windows:

```
..\appengine-java-sdk\bin\appcfg.cmd update war
```

To upload the app, using Mac OS X or Linux:

```
../appengine-java-sdk/bin/appcfg.sh update war
```

Enter your Google username and password at the prompts.

Accessing Your Application

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



★ **Google App Engine**

Using the Google Plugin for Eclipse

It's easy to use [the Eclipse development environment](#) to develop your Java App Engine application, just as you can to develop any other servlet-based web application. With the Google Plugin for Eclipse, it's even easier. The plugin lets you create, test and upload App Engine applications from within Eclipse.

The Google Plugin for Eclipse also makes it easy to develop applications using [Google Web Toolkit](#) (GWT), to run on App Engine or in any other environment.

This article describes how to install the Google Plugin for Eclipse, create a new App Engine project, and debug it using the development server running within Eclipse. The article also describes how to use the plugin to upload your project to App Engine.

For more information about the plugin, including how to use it for Google Web Toolkit projects, see [the Google Plugin for Eclipse documentation](#).

- [Getting Eclipse](#)
- [Installing the Google Plugin for Eclipse](#)
- [Creating a Project](#)
- [Running the Project](#)
- [Uploading to Google App Engine](#)
- [Running the Command Line Tools](#)

Getting Eclipse

The Google Plugin for Eclipse is available for Eclipse 3.3 (Europa), Eclipse 3.4 (Ganymede) and Eclipse 3.5 (Galileo). The "Eclipse IDE for Java EE Developers" includes all of the components you will need for web application development.

In addition to the Google Plugin for Eclipse, we recommend the Web Tools Platform (WTP) plugins for web development. See [the Web Tools Platform website](#). Among other things, WTP provides editing modes for JSP and HTML files.

Installing the Google Plugin for Eclipse

You can install the Google Plugin for Eclipse using the Software Update feature of Eclipse.

To install the plugin, using **Eclipse 3.6 (Helios)**:

1. Select the **Help** menu > **Install New Software....**
2. In the **Work with** text box, enter:

```
http://dl.google.com/eclipse/plugin/3.6
```

Click the **Add...** button. In the dialog that shows, click **OK** (keep the name blank, it will be retrieved from the update site.)

3. Click the triangle next to "Plugin" and "SDKs". Check the boxes next to "Google Plugin for Eclipse 3.6" and "Google App Engine Java SDK". You can also select the "Google Web Toolkit SDK" if you'd like to use [Google Web Toolkit](#) with your apps. Click the **Next** button. Follow the prompts to accept the terms of service and install the plugin.
4. When the installation is complete, Eclipse prompts you to restart. Click **Yes**. Eclipse restarts. The plugin is installed.

The process for installing the plugin for **Eclipse 3.5 (Galileo)** is almost exactly the same, with only a different install location. The location for the Google Plugin for Eclipse 3.5 is as follows:

- ```
http://dl.google.com/eclipse/plugin/3.5
```

To install the plugin, using **Eclipse 3.4 (Ganymede)**:

1. Select the **Help** menu > **Software Updates...** The "Software Updates and Add-ons" window opens.
2. Select the **Available Software** tab. Click the **Add Site...** button. The "Add Site" window opens. For "Location," enter the install location for the Eclipse 3.4 version of the plugin:

```
http://dl.google.com/eclipse/plugin/3.4
```

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)





---

## Google App Engine


# Uploading and Managing a Java App

The App Engine Java SDK includes a command for interacting with App Engine. You can use this command to upload new versions of the code, configuration and static files for your app to App Engine. You can also use the command to manage datastore indexes and download log data.

- [Uploading the App](#)
- [Updating Indexes](#)
- [Managing Task Queues](#)
- [Managing DoS Protection](#)
- [Managing Scheduled Tasks](#)
- [Downloading Logs](#)
- [Using an HTTP Proxy](#)
- [Command-Line Arguments](#)

---

## Uploading the App

If you are using Eclipse and the Google Plugin, you can upload your application directly from within Eclipse. To upload the app, click the App Engine deploy button in the toolbar:  For more information, see [Google Eclipse Plugin](#).

You can also upload your application from a command prompt. To use other features of the command, such as downloading logs, you must run the command from the command prompt. The command to run is in the SDK's `appengine-java-sdk/bin/` directory.

If you are using Windows, the command is as follows:

```
appengine-java-sdk\bin\appcfg.cmd [options] <action> <war-location>
```

If you are using Mac OS X or Linux, the command is as follows:

```
./appengine-java-sdk/bin/appcfg.sh [options] <action> <war-location>
```

The command takes the name of an action to perform and the location of your application's WAR directory as arguments.

To upload an application, use the `update` action, as follows:

```
./appengine-java-sdk/bin/appcfg.sh update myapp/war
```

These commands are OS-specific wrapper scripts that run the Java class `com.google.appengine.tools.admin.AppCfg` in `appengine-java-sdk/lib/appengine-tools-api.jar`.

---

## Updating Indexes

When you upload an application using the `update` action, the update includes the app's index configuration (the `datastore-indexes.xml` and `generated/datastore-indexes-auto.xml` files). If the index configuration defines an index that doesn't exist yet on App Engine, App Engine creates the new index. Depending on how much data is already in the datastore that needs to be mentioned in the new index, the process of creating the index may take a while. If the app performs a query that requires an index that hasn't finished building yet, the query will raise an exception.

To prevent this, you must ensure that the new version of the app that requires a new index is not the live version of the application until the indexes finish building. One way to do this is to give the app a new version number in `appengine-web.xml` whenever you add or change an index in the configuration. The app is uploaded as a new version, and does not become the default version automatically. When your indexes have finished building, you change the default version to the new one using the "Versions" section of the [Admin Console](#).

Another way to ensure that new indexes are built before the new app goes live is to upload the index configuration separately before uploading the app. To upload only the index configuration for an app, use the `update_indexes` action:

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



---

## Google App Engine

## Queries and Indexes

- [Overview](#)
- [Restrictions on Queries](#)
- [Fetching Results](#)
- [Introduction to Indexes](#)
- [Big Entities and Exploding Indexes](#)
- [Query Cursors](#)
- [Setting the Read Policy and Datastore Call Deadline](#)

---

### Overview

A query retrieves datastore entities that meet a specified set of conditions. The query specifies an entity kind, zero or more conditions based on entity property values (sometimes called "filters"), and zero or more sort order descriptions. When the query is executed, it fetches all entities of the given kind that meet all of the given conditions, sorted in the order described.

The [Master/Slave Datastore and the High Replication Datastore](#) have different guarantees when it comes to query consistency. By default:

- The Master/Slave datastore is strongly consistent for all queries.
- The High Replication datastore is strongly consistent by default for queries within an [entity group](#). With the High Replication Datastore, non-ancestor queries are always eventually consistent.

For more information, see [Setting the Read Policy and Datastore Call Deadline](#).

The low-level Java API provides a [Query](#) class for constructing queries and a [PreparedQuery](#) class for fetching and returning entities from the datastore.

```
import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.PreparedQuery;
import com.google.appengine.api.datastore.Query;

// ...
// Get the Datastore Service
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();

// The Query interface assembles a query
Query q = new Query("Person");
q.addFilter("lastName", Query.FilterOperator.EQUAL, lastNameParam);
q.addFilter("height", Query.FilterOperator.LESS_THAN, maxHeightParam);

// PreparedQuery contains the methods for fetching query results
// from the datastore
PreparedQuery pq = datastore.prepare(q);

for (Entity result : pq.asIterable()) {
 String firstName = (String) result.getProperty("firstName");
 String lastName = (String) result.getProperty("lastName");
 Long height = (Long) result.getProperty("height");
 System.out.println(lastName + " " + firstName + ", " + height.toString() + "
inches tall");
}
```

A filter includes a property name, a comparison operator, and a value. An entity passes the filter if it has a property of the given name and its value compares to the given filter value as described by the operator. The entity is a result for the query if it passes all of the query's filters.

The filter operator can be any of the following:

- `Query.FilterOperator.LESS_THAN`
- `Query.FilterOperator.LESS_THAN_OR_EQUAL`
- `Query.FilterOperator.EQUAL`
- `Query.FilterOperator.GREATER_THAN`
- `Query.FilterOperator.GREATER_THAN_OR_EQUAL`

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



---

## Google App Engine

## Defining Data Classes with JDO

You can use JDO to store plain Java data objects (sometimes referred to as "Plain Old Java Objects" or "POJOs") in the datastore. Each object that is made persistent with the `PersistenceManager` becomes an entity in the datastore. You use annotations to tell JDO how to store and recreate instances of your data classes.

---

**Note:** Earlier versions of JDO use `.jdo` XML files instead of Java annotations. These still work with JDO 2.3. This documentation only covers using Java annotations with data classes.

---

- [Class and Field Annotations](#)
- [Core Value Types](#)
- [Serializable Objects](#)
- [Child Objects and Relationships](#)
- [Embedded Classes](#)
- [Collections](#)
- [Object Fields and Entity Properties](#)
- [Inheritance](#)

---

### Class and Field Annotations

Each object saved by JDO becomes an entity in the App Engine datastore. The entity's kind is derived from the simple name of the class (inner classes use the `$` path without the package name). Each persistent field of the class represents a property of the entity, with the name of the property equal to the name of the field (with case preserved).

To declare a Java class as capable of being stored and retrieved from the datastore with JDO, give the class a `@PersistenceCapable` annotation. For example:

```
import javax.jdo.annotations.PersistenceCapable;

@PersistenceCapable
public class Employee {
 // ...
}
```

Fields of the data class that are to be stored in the datastore must be declared as persistent fields. To declare a field as persistent, give it a `@Persistent` annotation:

```
import java.util.Date;
import javax.jdo.annotations.Persistent;

// ...
 @Persistent
 private Date hireDate;
```

To declare a field as not persistent (it does not get stored in the datastore, and is not restored when the object is retrieved), give it a `@NotPersistent` annotation.

---

**Tip:** JDO specifies that fields of certain types are persistent by default if neither the `@Persistent` nor `@NotPersistent` annotations are specified, and fields of all other types are not persistent by default. See [the DataNucleus documentation](#) for a complete description of this behavior. Because not all of the App Engine datastore core [value types](#) are persistent by default according to the JDO specification, we recommend explicitly annotating fields as `@Persistent` or `@NotPersistent` to make it clear.

---

The type of a field can be any of the following. These are described in detail below.

- one of the core types supported by the datastore
- a Collection (such as a `java.util.List<...>`) or an array of values of a core datastore type
- an instance or Collection of instances of a `@PersistenceCapable` class
- an instance or Collection of instances of a Serializable class
- an embedded class, stored as properties on the entity

A data class must have one field dedicated to storing the primary key of the corresponding datastore entity. You can



Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



---

## Google App Engine

## Using Apache Ant

If you are not using [Eclipse](#) and [the Google Plugin for Eclipse](#), you'll probably want some other way to manage the process of building and testing your App Engine application. [Apache Ant](#) makes it easy to manage your project from the command line, or from other IDEs that work with Ant. The Java SDK includes a set of Ant macros to perform common App Engine development tasks, including starting the development server and uploading the app to App Engine.

This article describes an Ant build file useful for most projects. To skip the description and go directly to the complete file for copying and pasting, see [The Complete Build File](#).

- [Installing Ant](#)
- [The Project Directory](#)
- [Creating the Build File](#)
- [Defining the Class Path](#)
- [Enhancing JDO Classes](#)
- [Running the Development Server](#)
- [Uploading and Other AppCfg Tasks](#)
- [The Complete Build File](#)

---

### Installing Ant

If you do not already have Ant installed on your system, visit [the Apache Ant web site](#) to download and install it.

When Ant is installed and the `ant` command is on your command path, you can run the following command to verify that it works, and see which version is installed:

```
ant -version
```

---

### The Project Directory

As described in the [Getting Started Guide](#), your App Engine project must produce a directory structure using the [WAR](#) standard layout for Java web applications. (WAR archive files are not yet supported by the SDK.)

For these instructions, we will put all project-related files in a single directory named `Guestbook/` (for the guest book project described in [the Getting Started Guide](#)), with a subdirectory named `src/` for Java source code and a subdirectory named `war/` for the finished application files. The build process will compile the source code, and put the compiled Java bytecode in the appropriate location under `war/`. You will create other files for the WAR directly in the `war/` directory.

The complete project directory should look like this:

```
Guestbook/
 src/
 ...Java source code...
 META-INF/
 ...other configuration...
 war/
 ...JSPs, images, data files...
 WEB-INF/
 ...app configuration...
 classes/
 ...compiled classes...
 lib/
 ...JARs for libraries...
```

To create this directory structure from the command prompt, use the following commands. If you are using Windows, change the forward slashes `/` in these paths to backslashes `\`. (Within the Ant file itself, paths using forward slashes `/` work for all operating systems.)

```
mkdir Guestbook
cd Guestbook
...
...
...
...
```

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



---

## Google App Engine

## Using JDO with App Engine

Java Data Objects (JDO) is a standard interface for storing objects containing data into a database. The standard defines interfaces for annotating Java objects, retrieving objects with queries, and interacting with a database using transactions. An application that uses the JDO interface can work with different kinds of databases without using any database-specific code, including relational databases, hierarchical databases, and object databases. As with other interface standards, JDO simplifies porting your application between different storage solutions.

The App Engine Java SDK includes an implementation of JDO 2.3 for the App Engine datastore. The implementation is based on DataNucleus Access Platform, the open source reference implementation for JDO 2.3.

See [the Access Platform 1.1 documentation](#) for more information about JDO. In particular, see "[JDO Mapping](#)" and "[JDO API](#)".

- [Setting Up JDO](#)
- [Enhancing Data Classes](#)
- [Getting a PersistenceManager Instance](#)
- [Unsupported Features of JDO](#)
- [Disabling Transactions and Porting Existing JDO Apps](#)

---

### Setting Up JDO

To use JDO to access the datastore, an App Engine app needs the following:

- The JDO and DataNucleus App Engine plugin JARs must be in the app's `war/WEB-INF/lib/` directory.
- A configuration file named `jdoconfig.xml` must be in the app's `war/WEB-INF/classes/META-INF/` directory, with configuration that tells JDO to use the App Engine datastore.
- The project's build process must perform a post-compilation "enhancement" step on the compiled data classes to associate them with the JDO implementation.

If you are using [the Google Plugin for Eclipse](#), these three things are taken care of for you. The new project wizard puts the JDO and DataNucleus App Engine plugin JARs in the correct location, and creates the `jdoconfig.xml` file. The build process performs the "enhancement" step automatically.

If you are using Apache Ant to build your project, you can use an Ant task included with the SDK to perform the enhancement step. You must copy the JARs and create the configuration file when you set up your project. See [Using Apache Ant](#) for more information about the Ant task.

### Copying the JARs

The JDO and datastore JARs are included with the App Engine Java SDK. You can find them in the `appengine-java-sdk/lib/user/orm/` directory.

Copy the JARs to your application's `war/WEB-INF/lib/` directory.

Make sure the `appengine-api.jar` is also in the `war/WEB-INF/lib/` directory. (You may have already copied this when creating your project.) The App Engine DataNucleus plugin uses this JAR to access the datastore.

### Creating the jdoconfig.xml File

The JDO interface needs a configuration file named `jdoconfig.xml` in the application's `war/WEB-INF/classes/META-INF/` directory. You can create this file in this location directly, or have your build process copy this file from a source directory.

Create the file with the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<jdoconfig xmlns="http://java.sun.com/xml/ns/jdo/jdoconfig"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://java.sun.com/xml/ns/jdo/jdoconfig">

 <persistence-manager-factory name="transactions-optional">
 <property name="javax.jdo.PersistenceManagerFactoryClass"
 value="org.datanucleus.store.appengine.jdo.DatastoreJDOPersistenceManagerFactory"/>
 </persistence-manager-factory>
</jdoconfig>
```

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



---

## Google App Engine



# Java Application Configuration

In addition to the [web.xml deployment descriptor](#), an App Engine Java application uses a configuration file, named `appengine-web.xml`, to specify the app's registered application ID and the version identifier of the latest code, and to identify which files in the app's WAR are static files (like images) and which are resource files used by the application. The [AppCfg](#) command uses this information when you upload the app.

- [About appengine-web.xml](#)
- [Static Files and Resource Files](#)
  - [Including and Excluding Files](#)
  - [Setting the Browser Cache Expiration for Static Files](#)
  - [Changing the MIME Type for Static Files](#)
  - [Changing the Root Directory](#)
- [System Properties and Environment Variables](#)
- [Configuring Secure URLs \(SSL\)](#)
- [Enabling Sessions](#)
- [Inbound Services](#)
- [Disabling Precompilation](#)
- [Administration Console Custom Pages](#)
- [Custom Error Responses](#)
- [Warming Requests](#)

---

## About appengine-web.xml

An App Engine Java app must have a file named `appengine-web.xml` in its WAR, in the directory `WEB-INF/`. This is an XML file whose root element is `<appengine-web-app>`. A minimal file that specifies the application ID, a version identifier, and no static files or resource files looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
 <application>application-id</application>
 <version>1</version>
</appengine-web-app>
```

The `<application>` element contains the application's ID. This is the application ID you register when you create your application in [the Admin Console](#). When you upload your app, AppCfɡ gets the application ID from this file.

The `<version>` element contains the version identifier for the latest version of the app's code. The version identifier can contain letters, digits, and hyphens. AppCfɡ uses this version identifier when it uploads the application, telling App Engine to either create a new version of the app with the given identifier, or replace the version of the app with the given identifier if one already exists. You can test new versions of your app using a URL such as `http://version-id.latest.application-id.appspot.com`. You can select which version of the app your users see, the "default" version, using [the Admin Console](#).

The `<static-files>` and `<resource-files>` elements are described in the next section.

You can find the DTD and schema specifications for this file in the SDK's `docs/` directory.

---

## Static Files and Resource Files

Many web applications have files that are served directly to the user's browser, such as images, CSS style sheets, or browser JavaScript code. These are known as *static files* because they do not change, and can benefit from web servers dedicated just to static content. App Engine serves static files from dedicated servers and caches that are separate from the application servers.

Files that are accessible by the application code using the filesystem are called *resource files*. These files are stored on the application servers with the app.

By default, all files in the WAR are treated as both static files and resource files, except for JSP files, which are compiled into servlet classes and mapped to URL paths, and files in the `WEB-INF/` directory, which are never served as static files and always available to the app as resource files.

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)

[\( Sign In/Register for Account | Help \)](#)

United States ▼

Communities ▼

I am a... ▼

I want to... ▼

Secure Search

[Sun Quick Links ▼](#)[Products and Services](#)[Downloads](#)[Store](#)[Support](#)[Education](#)[Partners](#)[About](#)[Oracle Technology Network](#)[Oracle Technology Network](#) > [Java](#)[Java SE](#)[Java for Business](#)[Java Embedded](#)[Java EE](#)[Java ME](#)[JavaFX](#)[Java DB](#)[Web Tier](#)[Java Card](#)[Java TV](#)

## Java Data Objects (JDO)

### Overview

The Java Data Objects (JDO) API is a standard interface-based Java model abstraction of persistence, developed under the auspices of the [Java Community Process](#). The original JDO 1.0 is Java Specification Request 12 ( [JSR 12](#) ), and the current JDO 2.0 is Java Specification Request 243 ( [JSR 243](#) ). Beginning with JDO 2.0, the development of the API and the Technology Compatibility Kit (TCK) takes place within the [Apache JDO](#) open-source project.

If you are an application programmer, you can use JDO technology to directly store your Java domain model instances into the persistent store (database). Alternatives to JDO include direct file I/O, serialization, [JDBC](#), [Enterprise JavaBeans](#) (EJB), Bean-Managed Persistence (BMP) or Container-Managed Persistence (CMP) entity beans, and the [Java Persistence API](#).

The Apache JDO project is focused on building the JDO API and the TCK for compatibility testing of JDO implementations. Commercial and open-source [implementations](#) of JDO, providing the APIs used by application developers and their customers, are available for relational databases, object databases, and file systems.

### Benefits of Using JDO for Application Programming

**Ease of use:** Application programmers can focus on their domain object model and leave the details of persistence (field-by-field storage of objects) to the JDO implementation.

**Portability:** Applications written with the JDO API can be run on multiple implementations without recompiling or changing source code. Metadata, which describes persistence behavior external to the Java source code including most commonly used features of O/R mapping, is highly portable.

**Database independence:** Applications written with the JDO API are independent of the underlying database. JDO implementations support many different kinds of transactional data stores, including relational and object databases, XML, flat files, and others.

**High performance:** Application programmers delegate the details of persistence to the JDO implementation, which can optimize data access patterns for optimal performance.

**Integration with EJB:** Applications can take advantage of EJB features such as remote message processing, automatic distributed transaction coordination, and security, using the same domain object models throughout the enterprise.

### What's New

#### In Progress

**[JDO 2.1 Maintenance Release](#)** A list of changes proposed and approved for the forthcoming JDO 2.1 Maintenance Release.

#### Downloads

##### **[JDO 2.0 Specification \(JSR 243\)](#)**

Download the final release of the JDO 2.0 spec from the Java Community Process (JCP) site.

##### **[API and TCK](#)**

Download the JDO 2.0 API source and binaries along with the Technology Compatibility Kit source.

#### Resources

##### **[Javadoc](#)**

Browse the Javadoc for the JDO 2.0 API online or download it as a zip file.

##### **[JDO TCK](#)**

Get information on how to obtain, use, and see the results from the JDO Technology Compatibility Kit.

#### Community

##### **[Apache JDO](#)**

Apache JDO is a project of the Apache Software Foundation. Find out more about JDO and the Apache JDO developer community.

##### **[Bug Database](#)**

The JIRA issue repository for Apache JDO.

### Java SDKs and Tools

[Java SE](#)[Java EE and Glassfish](#)[Java ME](#)[JavaFX](#)[Java Card](#)[NetBeans IDE](#)

### Java Resources

[New to Java?](#)[APIs](#)[Code Samples & Apps](#)[Developer Training](#)[Documentation](#)[Java BluePrints](#)[Java.com](#)[Java.net](#)[Student Developers](#)[Tutorials](#)

**Hardware and Software**  
**Engineered to Work Together**

[About Oracle](#) | [Oracle and Sun](#) |  [Subscribe](#) | [Careers](#) | [Contact Us](#) | [Site Maps](#) | [Legal Notices](#) | [Terms of Use](#) | [Your Privacy Rights](#)



---

## Google App Engine

# The Java Development Server

The App Engine Java SDK includes a development web server for testing your application on your computer. The development web server simulates the App Engine Java runtime environment and all of its services, including the datastore. The [Google Plugin for Eclipse](#) can run the server in the Eclipse debugger. You can also run the development server from the command line.

- [Running the Development Web Server](#)
- [Using the Datastore](#)
- [Using Users](#)
- [Using URL Fetch](#)
- [The Development Console](#)
- [Command-Line Arguments](#)

---

## Running the Development Web Server

If you are using Eclipse and the Google Plugin, you can run the development web server in the Eclipse debugger. To run the server with the default configuration, select the **Run** menu, **Debug As > Web Application**. For more control over how the server is started, such as which port the server uses, create a new debug configuration using the configuration type "Web Application". For more information, see [Google Eclipse Plugin](#).

---

**Note:** When you start the development server from within Eclipse using the Google Plugin for Eclipse (discussed later), the server uses the port 8888 by default. When you start the server using the `dev_appserver` command described below, it uses the port 8080 by default. You can change the port number used by the `dev_appserver` command by providing the `--port=...` argument.

---

You can also run the development web server from a command prompt. The command to run is in the SDK's `appengine-java-sdk/bin/` directory.

If you are using Windows, the command is as follows:

```
appengine-java-sdk\bin\dev_appserver.cmd [options] war-location
```

If you are using Mac OS X or Linux, the command is as follows:

```
appengine-java-sdk/bin/dev_appserver.sh [options] war-location
```

The command takes the location of your application's WAR directory as an argument.

To stop the web server, press Control-C (on Windows, Mac or Linux).

These commands are OS-specific wrapper scripts that run the Java class `com.google.appengine.tools.KickStart` in `appengine-java-sdk/lib/appengine-tools-api.jar`.

---

## Using the Datastore

The development web server simulates the App Engine datastore using a file on your computer. The file is named `local_db.bin`, and it is created in your application's WAR directory, in the `WEB-INF/appengine-generated/` directory. (It is not uploaded with your application.)

This file persists between invocations of the web server, so data you store will still be available the next time you run the web server. To clear the contents of the datastore, shut down the server, then delete this file.

As described in [Datastore Index Configuration](#), the development server can generate configuration for datastore indexes needed by your application, determined from the queries it performs while you are testing it. This generates a file named `datastore-indexes-auto.xml` in the directory `WEB-INF/appengine-generated/` in the WAR. To disable automatic index configuration, create or edit the `datastore-indexes.xml` file in the `WEB-INF/` directory, using the attribute `autoGenerate="false"` for the `<datastore-indexes>` element. See [Datastore Index Configuration](#) for more information.

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)

[\( Sign In/Register for Account | Help \)](#)

United States ▼

Communities ▼

I am a... ▼

I want to... ▼

Secure Search

[Sun Quick Links ▼](#)[Products and Services](#)[Downloads](#)[Store](#)[Support](#)[Education](#)[Partners](#)[About](#)[Oracle Technology Network](#)[Oracle Technology Network](#) > [Java](#)[Java SE](#)[Java for Business](#)[Java Embedded](#)[Java EE](#)[Java ME](#)[JavaFX](#)[Java DB](#)[Web Tier](#)[Java Card](#)[Java TV](#)

## Java Servlet Technology

Java Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems. A servlet can almost be thought of as an applet that runs on the server side—without a face. Java servlets make many Web applications possible. »  
[Read More](#)

The reference implementation is included in the Java EE 5 SDK and also in the open-source Java Platform, Enterprise Edition (Java EE) application server, available through the [GlassFish project](#), on [java.net](#). The reference implementation source code for Servlet technology is available from the [svn repository](#) on [java.net](#). Additional information on all webtier technologies in GlassFish can be found at the [GlassFish Webtier page](#).

### What's New

**[Java Servlet 3.0 Specification now available for Public Review](#)** The Public Review Draft Specification for the Java Servlet 3.0 (JSR 315) Specification is now available for Public Review from <http://jcp.org/en/jsr/stage?listBy=public>

**[Learning GlassFish for Tomcat Users](#)** This paper demonstrates that the Web container capabilities provided by GlassFish offer superior extensibility, scalability, and usability compared to those of Tomcat—without sacrificing performance.

**[Maintenance Release of the Java Servlet 2.5 Specification](#)** Download the maintenance release of the Java Servlet Specification, version 2.5. This version of Java Servlet technology is included in the Java EE 5 Platform.

**[Web Tier to Go With Java EE 5: A Look at Resource Injection](#)** Read about the support for annotations by Java web tier technologies and how they can simplify access to resources, environment data, and life-cycle control.

**[Check Out JSPWiki!](#)** Learn about and download JSPWiki, the wiki engine based on JSP technology.

### Community

To learn about webtier technologies in GlassFish please visit the [GlassFish Webtier page](#) and post questions on [GlassFish Webtier forum](#) or send e-mail to [webtier@glassfish.dev.java.net](mailto:webtier@glassfish.dev.java.net).

Java Servlet Developers! [The Java Servlet Technology Forum](#) is a great place to learn more about developing with servlet technology.

### Events

**[Sun Tech Days - Advance your development skills with in-depth technical training](#)**. Topics include Java EE, Java SE, Java ME, Tools, Solaris, Cool Stuff, plus bonus NetBeans Day and OpenSolaris Day. Attend in a city near you. PH\*PH\*PHu00BBPH\*PH\*PH [Read More](#)

### Java SDKs and Tools

[Java SE](#)[Java EE and Glassfish](#)[Java ME](#)[JavaFX](#)[Java Card](#)[NetBeans IDE](#)

### Java Resources

[New to Java?](#)[APIs](#)[Code Samples & Apps](#)[Developer Training](#)[Documentation](#)[Java BluePrints](#)[Java.com](#)[Java.net](#)[Student Developers](#)[Tutorials](#)

**Hardware and Software**  
**Engineered to Work Together**

[About Oracle](#) | [Oracle and Sun](#) | [Subscribe](#) | [Careers](#) | [Contact Us](#) | [Site Maps](#) | [Legal Notices](#) | [Terms of Use](#) | [Your Privacy Rights](#)



[\( Sign In/Register for Account | Help \)](#)

United States ▼

Communities ▼

I am a... ▼

I want to... ▼

Secure Search

[Sun Quick Links ▼](#)[Products and Services](#)[Downloads](#)[Store](#)[Support](#)[Education](#)[Partners](#)[About](#)[Oracle Technology Network](#)[Oracle Technology Network](#) > [Java](#)[Java SE](#)[Java for Business](#)[Java Embedded](#)[Java EE](#)[Java ME](#)[JavaFX](#)[Java DB](#)[Web Tier](#)[Java Card](#)[Java TV](#)

## Java Servlet Technology

Java Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems. A servlet can almost be thought of as an applet that runs on the server side—without a face. Java servlets make many Web applications possible. »

[Read More](#)

The reference implementation is included in the Java EE 5 SDK and also in the open-source Java Platform, Enterprise Edition (Java EE) application server, available through the [GlassFish project](#), on [java.net](#). The reference implementation source code for Servlet technology is available from the [svn repository](#) on [java.net](#). Additional information on all webtier technologies in GlassFish can be found at the [GlassFish Webtier page](#).

### What's New

**[Java Servlet 3.0 Specification now available for Public Review](#)** The Public Review Draft Specification for the Java Servlet 3.0 (JSR 315) Specification is now available for Public Review from <http://jcp.org/en/jsr/stage?listBy=public>

**[Learning GlassFish for Tomcat Users](#)** This paper demonstrates that the Web container capabilities provided by GlassFish offer superior extensibility, scalability, and usability compared to those of Tomcat—without sacrificing performance.

**[Maintenance Release of the Java Servlet 2.5 Specification](#)** Download the maintenance release of the Java Servlet Specification, version 2.5. This version of Java Servlet technology is included in the Java EE 5 Platform.

**[Web Tier to Go With Java EE 5: A Look at Resource Injection](#)** Read about the support for annotations by Java web tier technologies and how they can simplify access to resources, environment data, and life-cycle control.

**[Check Out JSPWiki!](#)** Learn about and download JSPWiki, the wiki engine based on JSP technology.

### Community

To learn about webtier technologies in GlassFish please visit the [GlassFish Webtier page](#) and post questions on [GlassFish Webtier forum](#) or send e-mail to [webtier@glassfish.dev.java.net](mailto:webtier@glassfish.dev.java.net).

Java Servlet Developers! [The Java Servlet Technology Forum](#) is a great place to learn more about developing with servlet technology.

### Events

**[Sun Tech Days - Advance your development skills with in-depth technical training.](#)** Topics include Java EE, Java SE, Java ME, Tools, Solaris, Cool Stuff, plus bonus NetBeans Day and OpenSolaris Day. Attend in a city near you. PH\*PH\*PHu00BBPH\*PH\*PH [Read More](#)

### Java SDKs and Tools

[Java SE](#)[Java EE and Glassfish](#)[Java ME](#)[JavaFX](#)[Java Card](#)[NetBeans IDE](#)

### Java Resources

[New to Java?](#)[APIs](#)[Code Samples & Apps](#)[Developer Training](#)[Documentation](#)[Java BluePrints](#)[Java.com](#)[Java.net](#)[Student Developers](#)[Tutorials](#)

**Hardware and Software**  
Engineered to Work Together

[About Oracle](#) | [Oracle and Sun](#) | [Subscribe](#) | [Careers](#) | [Contact Us](#) | [Site Maps](#) | [Legal Notices](#) | [Terms of Use](#) | [Your Privacy Rights](#)

SDN Home > Java Technology > Java Platform, Enterprise Edition (Java EE) >

## J2EE

# JavaServer Pages Technology

### Downloads

#### Reference

- [API Specifications](#)
- [BluePrints](#)
- [Documentation](#)
- [FAQs](#)
- [Technical Articles & Tips](#)
- [White Papers](#)
- [Third-Party](#)

#### Community

- [Books & Authors](#)
- [Forums](#)
- [Bug Database](#)

#### Learning

- [Course Certification](#)
- [Instructor-Led Courses](#)
- [Online Sessions & Courses](#)
- [Quizzes](#)

JavaServer Pages (JSP) technology provides a simplified, fast way to create dynamic web content. JSP technology enables rapid development of web-based applications that are server- and platform-independent. » [Read More](#)

The JSP Standard Tag Library (JSTL) is a collection of tag libraries that implement general-purpose functionality common to many Web applications.

#### JSP Technology in the Java EE 5 Platform

The focus of Java EE 5 has been ease of development by making use of Java language annotations that were introduced by J2SE 5.0. JSP 2.1 supports this goal by defining annotations for dependency injection on JSP tag handlers and context listeners.

Another key concern of the Java EE 5 specification has been the alignment of its webtier technologies, namely JavaServer Pages (JSP), JavaServer Faces (JSF), and JavaServer Pages Standard Tag Library (JSTL).

The outcome of this alignment effort has been the Unified Expression Language (EL), which integrates the expression languages defined by JSP 2.0 and JSF 1.1.

The main key additions to the Unified EL that came out of the alignment work have been:

- A pluggable API for resolving variable references into Java objects and for resolving the properties applied to these Java objects,
- Support for deferred expressions, which may be evaluated by a tag handler when needed, unlike their regular expression counterparts, which get evaluated immediately when a page is executed and rendered, and
- Support for lvalue expression, which appear on the left hand side of an assignment operation. When used as an lvalue, an EL expression represents a reference to a data structure, for example: a JavaBeans property, that is assigned some user input.

The new Unified EL is defined in its own specification document, which is delivered along with the JSP 2.1 specification.

Thanks to the Unified EL, JSTL tags, such as the JSTL iteration tags, can now be used with JSF components in an intuitive way.

JSP 2.1 leverages the Servlet 2.5 specification for its web semantics.

#### What's New

[Maintenance Review on JavaServer Pages JSR 245](#). There are two sets of changes

- EL:
  - [Overview](#)
  - [Change log](#)
- JSP:



#### Related Links

##### Popular Downloads

- The GlassFish Project: Open Source Java EE 5 Application Server Implementation
- Java Web Services Developer Pack
- Java EE SDK
- Sun Java System Web Server
- Sun Java System Application Server

##### Technical Topics

- Performance
- Desktop
- Web Services

##### Products and Technologies

- Java Servlet Technology
- JavaServer Pages Standard Tag Library
- JavaServer Faces
- TLDDoc

##### Sun Resources

- Developer Technical Support
- Professional Training
- Professional Certification

- [Overview](#)
- [Change log](#)

Follow JSP development (API and implementation) at <https://jsp.dev.java.net> a sub-project of [GlassFish](#) project. Get the latest source or binaries there.

Follow EL development (API and implementation) at <https://uel.dev.java.net> a sub-project of [GlassFish](#) project. Get the latest source or binaries there.

**Final Release of JavaServer Pages Specification** Download the final release of the JavaServer Pages Specification, version 2.1. This version of JavaServer Pages technology is part of the Java EE platform.

**Web Tier to Go With Java EE 5: A Look at Resource Injection**

Read about the support for annotations by Java web tier technologies and how they can simplify access to resources, environment data, and life-cycle control.

**What's New in JSP Technology 2.1** In this first article in the Web Tier to Go with Java EE 5 series, we discuss the major contributions of JavaServer Pages technology version 2.1 to the Java EE platform.

**Learn about the New Unified Expression Language!** Read about the new unified expression language and what it offers all web-tier Java technologies. Also, see a demo that illustrates the capabilities of this powerful, yet easy-to-use expression language.  
» [Read more](#)

### Community

To learn about webtier technologies in GlassFish please visit the [GlassFish Webtier page](#) and post questions on [GlassFish Webtier forum](#) or send e-mail to [webtier@glassfish.dev.java.net](mailto:webtier@glassfish.dev.java.net)

JavaServer Pages Developers! [The JavaServer Pages Technology Forum](#) is a great place to learn more about developing with JSP technology.

### Events

**Sun Tech Days - Advance your development skills with in-depth technical training.**

Topics include Java EE, Java SE, Java ME, Tools, Solaris, Cool Stuff, plus bonus NetBeans Day and OpenSolaris Day. Attend in a city near you. » [Read More](#)

**Subscribe to Newsletters**

Oracle is reviewing the Sun product roadmap and will provide guidance to customers in accordance with Oracle's standard product communication policies. Any resulting features and timing of release of such features as determined by Oracle's review of roadmaps, are at the sole discretion of Oracle. All product roadmap information, whether communicated by Sun Microsystems or by Oracle, does not represent a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. It is intended for information purposes only, and may not be incorporated into any contract.



[About Sun](#) | [About This Site](#) | [Newsletters](#) | [Contact Us](#) | [Employment](#) | [How to Buy](#) | [Licensing](#) | [Terms of Use](#) | [Privacy](#) | [Trademarks](#)

© 2010, Oracle Corporation and/or its affiliates

### A Sun Developer Network Site

Unless otherwise licensed, code in all technical manuals herein (including articles, FAQs, samples) is provided under this License.

 [Sun Developer RSS Feeds](#)



---

## Google App Engine

### The Administration Console

The Google App Engine Administration Console gives you complete access to the public version of your application. Access the Console by visiting the following link in your web browser:

<https://appengine.google.com/>

Sign in with your Google account, or create one with an email address and password.

If you are using App Engine with your [Google Apps](#) account, you can sign in to App Engine for your domain using a URL similar to the following, where *your-domain.com* is your Google Apps domain:

<https://appengine.google.com/a/your-domain.com>

You can use the Administration Console to:

- Create a new application with a free appspot.com sub-domain, or a top-level domain name of your choosing.
- Invite other people to be developers for your application, so they can access the Console and upload new versions of the code.
- View request and error logs, and analyze traffic.
- Browse your application's datastore and manage indexes.
- [Administer your datastore](#)
- View your application's [instances](#).
- Manage task queues, allowing for pausing, purging, and deleting queues.
- Manage individual tasks in a task queue, allowing for viewing, deleting, or running individual tasks immediately.
- Test new versions of your application, and switch the version that your users see.

See [Uploading an App \(Java\)](#) or [Uploading an App \(Python\)](#) for information about uploading your application's code to App Engine and other features for interacting with your live application.

Except as otherwise [noted](#), the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

Java is a registered trademark of Oracle and/or its affiliates

©2011 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



( [Sign In/Register for Account](#) | [Help](#) )

United States ▼

Communities ▼

I am a... ▼

I want to... ▼



Secure Search



Sun Quick Links ▼

Products and Services

Downloads

Store

Support

Education

Partners

About

Oracle Technology Network

Oracle Technology Network > Articles

Application Development Framework

Application Express

Business Intelligence

Cloud Computing

Communications

Database Performance & Availability

Data Warehousing

.NET

Dynamic Scripting Languages

Embedded

Enterprise 2.0

Enterprise Architecture

Enterprise Management

Grid Computing

Identity & Security

Java

Linux

Service-Oriented Architecture

SQL & PL/SQL

Server and Storage Administration

Server and Storage Development

Systems Hardware and Architecture

Virtualization

## The Java Persistence API - A Simpler Programming Model for Entity Persistence

By [Rahul Biswas and Ed Ort](#), May 2006

[Articles Index](#)

The major theme of version 5 of the Java Platform, Enterprise Edition (Java EE, formerly referred to as J2EE) is ease of development. Changes throughout the platform make the development of enterprise Java technology applications much easier, with far less coding. Significantly, these simplifications have not changed the platform's power: The Java EE 5 platform maintains all the functional richness of the previous version, J2EE 1.4.

Enterprise developers should notice dramatic simplification in [Enterprise JavaBeans \(EJB\) technology](#). Previous articles, such as [Introduction to the Java EE 5 Platform](#) and [Ease of Development in Enterprise JavaBeans Technology](#) have described the simplifications made in EJB 3.0 technology, an integral part of the Java EE 5 platform.

A major enhancement in EJB technology is the addition of the new Java Persistence API, which simplifies the entity persistence model and adds capabilities that were not in EJB 2.1 technology. The Java Persistence API deals with the way relational data is mapped to Java objects ("persistent entities"), the way that these objects are stored in a relational database so that they can be accessed at a later time, and the continued existence of an entity's state even after the application that uses it ends. In addition to simplifying the entity persistence model, the Java Persistence API standardizes object-relational mapping.

In short, EJB 3.0 is much easier to learn and use than was EJB 2.1, the technology's previous version, and should result in faster development of applications. With the inclusion of the Java Persistence API, EJB 3.0 technology also offers developers an entity programming model that is both easier to use and yet richer.

The Java Persistence API draws on ideas from leading persistence frameworks and APIs such as Hibernate, Oracle TopLink, and Java Data Objects (JDO), and well as on the earlier EJB container-managed persistence. The Expert Group for the [Enterprise JavaBeans 3.0 Specification \(JSR 220\)](#) has representation from experts in all of these areas as well as from other individuals of note in the persistence community.

This article supplements the earlier articles by focusing on entity-related code. Here you'll be able to examine EJB 2.1 entity beans in an application and compare them to EJB 3.0 entities in an equivalent application. More specifically, you'll be able to view side by side the source code for EJB 2.1 entity beans that use container-managed persistence and relationships and compare them to the source code for equivalently functioning EJB 3.0 entities written to the Java Persistence API. Note that in the Java Persistence API, what used to be called entity beans are now simply called entities. You'll see how much easier and streamlined the EJB 3.0 technology code is.

The article highlights some of the important features that simplify the EJB 3.0 version of the code. Although this article focuses on the Java Persistence API and its use in an EJB 3.0 container, the API can also be used outside the container -- for instance, in applications for the Java Platform, Standard Edition (Java SE, formerly referred to as J2SE). The API also provides support for pluggable, third-party persistence providers. For example, a persistence implementation from one vendor can be used with an EJB container from another vendor, provided that the container and the persistence implementation both conform to JSR 220.

This article assumes that you're familiar with the basic concepts of EJB technology that underlie EJB 2.1. If you're not, see the chapter "Enterprise Beans" in the [J2EE 1.4 Tutorial](#). For more information about EJB 3.0 and Java Persistence concepts, see the chapter "Enterprise Beans" in the [Java EE 5 Tutorial](#).

### Contents

- [The Application](#)
- [The Entities](#)
- [The Class Definition](#)
- [Persistent Fields and Properties](#)
- [Entity Identity](#)
- [Relationships](#)
- [Inheritance and Polymorphism](#)
- [Operations on Entities](#)
- [Transactions](#)
- [Queries](#)
- [Testing Entities Outside of the EJB Container](#)
- [Summary](#)
- [For More Information](#)

First, let's look at the application in use.

### The Application

This article uses the EJB 2.1 technology version of the application, the CMP Customer Sample Application, from the J2EE 1.4 samples bundle. You can download the samples bundle from the [J2EE 1.4 Downloads](#) page. The samples bundle includes instructions for installing and running the EJB 2.1 version of the application.

The EJB 3.0 version of the same application is called the Java Persistence Demo and is available for download [here](#). The download bundle includes instructions for installing and running the EJB 3.0 version of the application.

Both versions of the application do the same thing: They interact with a relational database to store and display information about customer subscriptions to periodicals. The database stores information such as a customer's name and address, as well as the type of periodical to which the customer is subscribed -- that is, magazine, journal, or newspaper. You can submit requests to the application to display subscription-related information. For example, you can get a list of all subscriptions for a particular customer.

Figure 1 shows what some of the interactions look like. Click to enlarge each image.

### Simplicity at a Glance

The Java Persistence API simplifies the programming model for entity persistence and adds capabilities that were not in EJB 2.1. Here's a quick list of its simplifications and additions:

- Requires fewer classes and interfaces
- Virtually eliminates lengthy deployment descriptors through annotations
- Addresses most typical specifications through annotation defaults
- Provides cleaner, easier, standardized object-relational mapping
- Eliminates the need for lookup code
- Adds support for inheritance, polymorphism, and polymorphic queries.
- Adds support for named (static) and dynamic queries.
- Provides a Java Persistence query language -- an enhanced EJB QL
- Makes it easier to test entities outside of the EJB container
- Can be used outside of the container
- Can be used with pluggable, third-party persistence providers

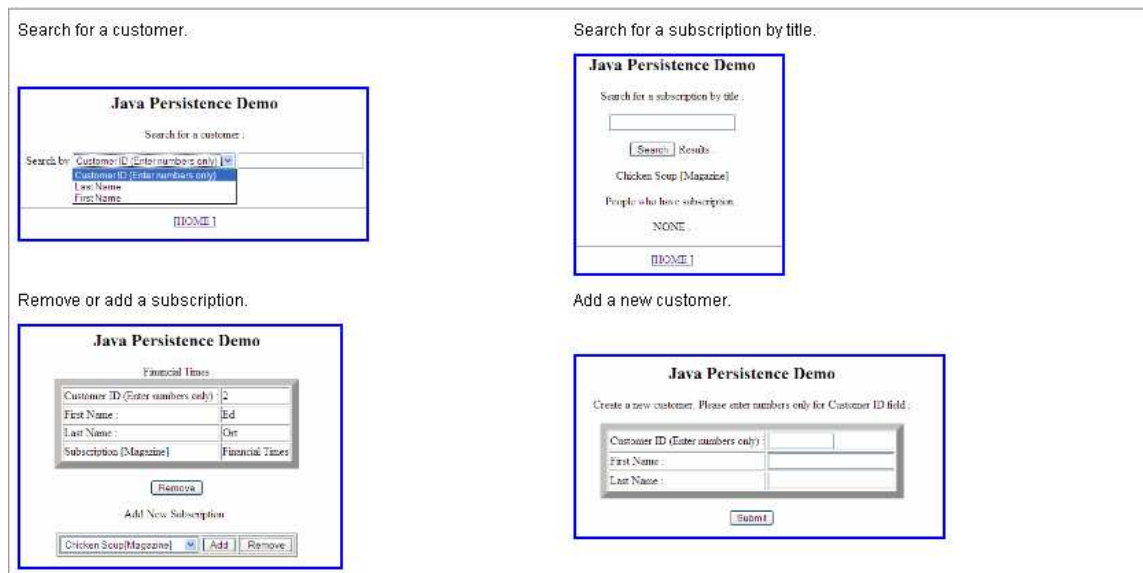


Figure 1: Java Persistence Demo Interactions

### The Entities

Let's examine the entity-related code within the application. First, let's look at what classes and interfaces are required for the entities in the application. Look at Figure 2. Compare the list in the EJB 2.1 version with that of the EJB 3.0 version.

EJB 2.1	EJB 3.0
AddressBean LocalAddress LocalAddressHome CustomerBean LocalCustomer LocalCustomerHome SubscriptionBean LocalSubscription LocalSubscriptionHome	Address Customer Subscription

Figure 2: Required Classes and Interfaces  
[Click here](#) for a larger image

### You need to code fewer classes and interfaces

For an EJB 3.0 entity, you no longer need to code interfaces such as `LocalAddressHome` and `LocalAddress` -- or even a deployment descriptor. All you need to provide is an entity class. So in the application's EJB 3.0 version, what's required for an entity has been reduced from three classes -- for local interfaces and a business class -- to one entity class. This simplification is not limited to entities. EJB 3.0 technology has eliminated the requirement for a home interface in enterprise beans of any type. In addition, you no longer need to implement the `EJBObject` and `EJBLocalObject` interfaces. For example, a session bean now requires only a bean class and a business interface, which is a simple Java technology interface.

### The Class Definition

Let's now look at the code for the entities. Let's start by comparing some of the key parts of the code for the EJB 2.1 `AddressBean` class and the EJB 3.0 `Address` class. First, we'll look at the class definition. You can view the entire entity bean class and entity class by clicking on their names in the Figure 3.

EJB 2.1	EJB 3.0
<b>Entity Bean Class: AddressBean.java</b> <pre> public abstract class AddressBean implements EntityBean {     ....     public void setEntityContext(EntityContext ctx) {         ....     }     public void unsetEntityContext() {         ....     }     public void ejbRemove() {}     public void ejbLoad() {}     public void ejbStore() {}     public void ejbPassivate() {}     public void ejbActivate() {} }           </pre>	<b>Entity Class: Address.java</b> <pre> @Entity //name defaults to the unqualified entity class name public class Address implements java.io.Serializable{     ....     //Entity must have a no-argument     //public or protected constructor     public Address(){} }           </pre>

Figure 3: Comparing Class Definitions  
[Click here](#) for a larger image

### An entity is a Plain Old Java Object (POJO), so no boilerplate is required

The EJB 2.1 version of the entity bean class implements the `javax.ejb.EntityBean` interface. In fact, an EJB 2.1 entity bean class must implement this interface to interact with the EJB 2.1 container. Because of that, the bean class must also implement all of the methods in the interface: `ejbRemove`, `ejbActivate`, `ejbPassivate`, `ejbLoad`, `ejbStore`, `setEntityContext` and `unsetEntityContext`. The class must implement these callback methods even if it doesn't use them, as is the case for most of these methods in the EJB 2.1 example in Figure 3.

By comparison, an EJB 3.0 entity class is a simple, nonabstract, concrete class -- a POJO, a class that you can instantiate like any other simple Java technology class, with a `new` operation. It does not implement `javax.ejb.EntityBean` or any other container-imposed interface. Because the entity class no longer implements `javax.ejb.EntityBean`, you are no longer required to implement any callback methods. However, you can still implement callback methods in the entity class if you need them to handle life-cycle events for the entity. Also notice the no-argument public constructor. In EJB 3.0 technology, an entity class must have a no-argument public or protected constructor.

### Annotations minimize what needs to be specified

The `@Entity` metadata annotation marks the EJB 3.0 class as an entity. EJB 3.0 and the Java Persistence API rely heavily on [metadata annotation](#), a feature that was introduced in J2SE 5.0. An annotation consists of the `@` sign preceding an annotation type, sometimes followed by a parenthetical list of element-value pairs. The EJB 3.0 specification defines a variety of annotation types. Some examples are those that specify a bean's type, such as



less; those that specify whether a bean is remotely or locally accessible, such as `Remote` and `Local`; transaction attributes, such as `TransactionAttribute`; and security and method permissions, such as `MethodPermissions`, `Checked`, and `SecurityRoles`. The Java Persistence API adds annotations, such as `Entity`, that are specific to entities. The reliance on annotations in EJB 3.0 and the Java Persistence API demonstrates a significant design shift.

#### Defaults make things even easier

In many cases, the application can use defaults instead of explicit metadata annotation elements. In these cases, you don't have to completely specify a metadata annotation. You can obtain the same result as if you had fully specified the annotation. For example, the `Entity` annotation has a `name` element that is used to specify the name to be used in queries that reference the entity. The `name` element defaults to the unqualified name of the entity class. So in the code for the `Address` entity, an annotation of `Entity` is enough. There's no need to specify a name in the annotation. These defaults can make annotating entities very simple. In many cases, defaults are assumed when an annotation is not specified. In those cases, the defaults represent the most common specifications. For example, container-managed transaction demarcation -- in which the container, as opposed to the bean, manages the commitment or rollback of a unit of work to a database -- is assumed for an enterprise bean if no annotation is specified. These defaults illustrate the coding-by-exception approach that guides EJB 3.0 technology. The process is simpler for the developer. You need to code only when the default is inadequate.

#### Persistent Fields and Properties

Let's now compare how persistence is declared for fields and properties.

EJB 2.1	EJB 3.0
<p>Entity Bean Class: <code>AddressBean.java</code></p> <pre>... //access methods for cap fields public abstract String getAddressID(); public abstract void setAddressID(String id); ... }</pre> <p>Deployment Descriptor: <code>ejb-jar.xml</code></p> <pre>&lt;ejb-jar version="2.1" xsi:schemaLocation="..."   &lt;display-name&gt;Ejb1&lt;/display-name&gt;   &lt;enterprise-beans&gt;     &lt;entity&gt;       &lt;ejb-name&gt;AddressBean&lt;/ejb-name&gt;       &lt;cmp-field&gt;         &lt;field-name&gt;addressID&lt;/field-name&gt;       &lt;/cmp-field&gt;     &lt;/entity&gt;   &lt;/enterprise-beans&gt; &lt;/ejb-jar&gt;</pre>	<p>Entity Class: <code>Address.java</code></p> <pre>... private String addressID; ... public Address(String id, ...){     setAddressID(id);     setStreet(street); } ... @Column(name="addressID") public String getAddressID(){     ... } ... }</pre> <p>XML Descriptor: Not required</p>

Figure 4: Comparing Persistent Field and Property Declarations  
[Click here for a larger image](#)

#### Persistence declarations are simpler

In EJB 2.1 technology, you specify which fields of the class need to be persisted in a database by defining public abstract getter and setter methods for those fields and by making specifications in a deployment descriptor -- an approach that many programmers find clumsy and far from intuitive. EJB 3.0 technology does not require these specifications. In essence, persistence is built into an entity. The persistent state of an entity is represented either by its persistent fields or persistent properties.

Recall that an entity is a POJO. Like a POJO, it can have nonabstract, private instance variables, such as the `addressID` variable in the `Address` class. In the Java Persistence API, an entity can have field-based or property-based access. In field-based access, the persistence provider accesses the state of the entity directly through its instance variables. In property-based access, the persistence provider uses JavaBeans-style get/set accessor methods to access the entity's persistent properties. All fields and properties in the entity are persisted. You can, however, override a field or property's default persistence by marking it with the `Transient` annotation or the Java keyword `transient`.

#### No XML descriptor needed to declare persistent fields

EJB 2.1 technology, entity bean fields are identified as persistent fields in the bean's deployment descriptor, `ejb-jar.xml`, an often large and complex XML file. In addition to coding public accessor methods in the entity bean class, you must specify in the deployment descriptor a `cmp-field` element for each persistent field. In the Java Persistence API, you no longer need to provide a deployment descriptor, called an XML descriptor in the Java Persistence API, to specify an entity's persistent fields.

#### Default mappings are used where possible

In EJB 2.1 technology, you define the mapping between an entity bean's fields and their corresponding database columns in a vendor-specific deployment descriptor, such as `sun-ejb-jar.xml`. By contrast, the Java Persistence API does not require XML descriptors. Instead, you specify the mappings in the entity class by marking the appropriate persistent field or property accessor method with the `Column` annotation. You can, however, take advantage of annotation defaults. If you don't specify an `Column` annotation for a specific persistent field or property, a default mapping to a database column of the same name as the field or property name is assumed. Although you don't need to specify XML descriptors, you have the option of using them as an alternative to annotations or to supplement annotations. Using an XML descriptor might be useful in externalizing object-relational mapping information. Also, multiple XML descriptors can be useful in tailoring object-relational mapping information to different databases.

The Java Persistence API standardizes object-relational mapping, enabling fully portable applications.

#### Entity Identity

There are also simplifications in the way primary and composite keys for entities are specified.

EJB 2.1	EJB 3.0
<p>Deployment Descriptor: <code>ejb-jar.xml</code></p> <pre>&lt;ejb-jar version="2.1" xsi:schemaLocation="..."   &lt;display-name&gt;Ejb1&lt;/display-name&gt;   &lt;enterprise-beans&gt;     &lt;entity&gt;       &lt;ejb-name&gt;AddressBean&lt;/ejb-name&gt;       &lt;cmp-field&gt;         &lt;field-name&gt;addressID&lt;/field-name&gt;       &lt;/cmp-field&gt;       &lt;prim-key-class&gt;java.lang.String&lt;/prim-key-class&gt;       &lt;primaryKey-field&gt;addressID&lt;/primaryKey-field&gt;     &lt;/entity&gt;   &lt;/enterprise-beans&gt; &lt;/ejb-jar&gt;</pre>	<p>Entity Class: <code>Address.java</code></p> <pre>... @Id public String getAddressID(){ //primary key     return addressID; } public void setAddressID(String id){     this.addressID=id; } ... }</pre> <p>XML Descriptor: Not required</p>

Figure 5: Comparing Settings for Primary Keys  
[Click here for a larger image](#)

### No XML descriptor needed to specify the primary key

In EJB 2.1 technology, the primary key for an entity bean -- that is, its unique identifier -- is specified not in the entity bean class but rather in its deployment descriptor. In the Java Persistence API, you don't need to provide an XML descriptor to specify an entity's primary key. Instead, you specify the primary key in the entity class by marking an appropriate persistent field or persistent property with the `@Id` annotation. You can specify composite keys in two different ways, using an `@IdClass` annotation or an `@EmbeddedId` annotation.

### Relationships

So far, the focus has been on the coding simplifications that relate to basic persistence, that is, the persistence of entities and their fields or properties. Now, let's look at the coding simplifications that relate to entity relationships. To do that, let's focus on the `CustomerBean` class and compare it to the `CustomerBean` entity. The `CustomerBean` class has a one-to-many unidirectional relationship with `AddressBean` and a many-many bidirectional relationship with `SubscriptionBean`. Let's see how much easier it is to specify an equivalent set of relationships between the EJB 3.0 entities.

EJB 2.1	EJB 3.0
<pre>Entity Bean Class: CustomerBean.java  public abstract class CustomerBean implements EntityBean {     ...      //access methods for car fields     public abstract Collection getAddresses();     public abstract void setAddresses         (Collection addresses);      public abstract Collection getSubscriptions();     public abstract void setSubscriptions         (Collection subscriptions);     ... }</pre> <hr/> <pre>Deployment Descriptor: ejb-jar.xml  &lt;relationships&gt;   &lt;ejb-relation&gt;     &lt;ejb-relationship-role&gt;       &lt;ejb-relationship-role-name&gt;         CustomerBean-AddressBean       &lt;/ejb-relationship-role-name&gt;       &lt;multiplicity&gt;One&lt;/multiplicity&gt;       &lt;relationship-role-source&gt;         &lt;ejb-name&gt;CustomerBean&lt;/ejb-name&gt;       &lt;/relationship-role-source&gt;       &lt;car-field&gt;         &lt;car-field-name&gt;addresses&lt;/car-field-name&gt;         &lt;car-field-type&gt;java.util.Collection       &lt;/car-field-type&gt;       &lt;/car-field&gt;     &lt;/ejb-relationship-role&gt;     &lt;ejb-relationship-role&gt;       &lt;ejb-relationship-role-name&gt;         AddressBean-CustomerBean       &lt;/ejb-relationship-role-name&gt;       &lt;multiplicity&gt;Many&lt;/multiplicity&gt;       &lt;cascade&gt;delete&lt;/&gt;       &lt;relationship-role-source&gt;         &lt;ejb-name&gt;AddressBean&lt;/ejb-name&gt;       &lt;/relationship-role-source&gt;       &lt;/ejb-relationship-role&gt;     &lt;/ejb-relation&gt;   &lt;/relationships&gt;   &lt;ejb-relation&gt;     &lt;ejb-relationship-role&gt;       &lt;ejb-relationship-role-name&gt;         CustomerBean-SubscriptionBean       &lt;/ejb-relationship-role-name&gt;       &lt;multiplicity&gt;Many&lt;/multiplicity&gt;       &lt;relationship-role-source&gt;         &lt;ejb-name&gt;CustomerBean&lt;/ejb-name&gt;       &lt;/relationship-role-source&gt;       &lt;car-field&gt;         &lt;car-field-name&gt;subscriptions       &lt;/car-field-name&gt;       &lt;car-field-type&gt;java.util.Collection       &lt;/car-field-type&gt;       &lt;/car-field&gt;     &lt;/ejb-relationship-role&gt;     &lt;ejb-relationship-role&gt;       &lt;ejb-relationship-role-name&gt;         SubscriptionBean-CustomerBean       &lt;/ejb-relationship-role-name&gt;       &lt;multiplicity&gt;Many&lt;/multiplicity&gt;       &lt;relationship-role-source&gt;         &lt;ejb-name&gt;SubscriptionBean&lt;/ejb-name&gt;       &lt;/relationship-role-source&gt;       &lt;car-field&gt;         &lt;car-field-name&gt;customers       &lt;/car-field-name&gt;       &lt;car-field-type&gt;java.util.Collection       &lt;/car-field-type&gt;       &lt;/car-field&gt;     &lt;/ejb-relationship-role&gt;     &lt;/ejb-relation&gt;   &lt;/relationships&gt;</pre>	<pre>Entity Class: Customer.java  ... @Entity ... public class Customer implements java.io.Serializable{     ...      @OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER)     public Collection&lt;Address&gt; getAddresses(){         return addresses;     }     public void setAddresses (Collection&lt;Address&gt; addresses){         this.addresses=addresses;     } }  @ManyToMany(fetch=FetchType.EAGER ) @JoinTable(     name="CUSTOMERBEANSUBSCRIPTIONBEAN",     joinColumns=@JoinColumn(name="CUSTOMERBEAN_CUSTOMERID96",         referencedColumnName="customerid"),     inverseJoinColumns=@JoinColumn(         name="SUBSCRIPTIONBEAN_TITLE",         referencedColumnName="TITLE") ) public Collection&lt;Subscription&gt;getSubscriptions(){     return subscriptions; } public void setSubscriptions (Collection&lt;Subscription&gt; subscriptions){     this.subscriptions=subscriptions; } }</pre> <hr/> <pre>Entity Class: Subscription.java  ... @Entity ... public class Subscription implements java.io.Serializable{     ...      @ManyToMany(mappedBy="subscriptions")     public Collection&lt;Customer&gt;getCustomers(){         return customers;     }     public void setCustomers(Collection&lt;Customer&gt; customers){         this.customers=customers;     } }</pre> <hr/> <pre>XML Descriptor: Not required</pre>

Figure 6: Comparing Relationship Declarations  
[Click here for a larger image](#)

### Relationship declarations are simpler

Specifying container-managed relationships in EJB 2.1 technology can be quite complex. If entity bean A has a relationship with entity bean B, you must specify abstract getter and setter methods in entity bean A for the related entity bean B. In addition, you must provide a rather lengthy entry for the relationship in the `ejb-jar.xml` deployment descriptor. In the Java Persistence API, you specify the relationship as you would for any POJO -- through a reference to the related entity object. In addition, you specify annotations that describe the semantics of the relationship and any database table-related metadata. You do not need an XML descriptor to specify entity relationships.

Note, however, that unlike in EJB 2.1 with container-managed persistence, the application that uses the Java Persistence API is responsible for managing relationships. For example, unlike EJB 2.1 technology, the Java Persistence API requires the backpointer reference in a bidirectional relationship to be set. Assume entity A has a bidirectional relationship to entity B. In EJB 2.1 technology, all you need to do is set the relationship from A to B -- the underlying persistence implementation is responsible for setting the backpointer reference from B to A. The Java Persistence API requires the references to be set on both sides of the relationship. This means that you have to explicitly call `b.setA(a)` and `a.setB(b)`.



### Annotations specify multiplicity and related information

You declare the relationships in annotations that reference the related entities. Annotations on the persistent fields and properties specify the multiplicity of a relationship, such as one-to-many or many-to-many, and other information, such as cascade and eager fetching. (By comparison, in EJB 2.1 technology, this information is specified in vendor-specific settings.) For example, the `@OneToMany` annotation on the `getAddresses` accessor method in the `Customer` entity, specifies a one-to-many relationship with the `Address` entity.

The `Customer` entity is the owning side of the relationship. There is no associated annotation in the inverse side of the relationship. So this is a one-to-many unidirectional relationship with the `Address` entity. The value specified for the `cascade` element in the annotation specifies that life-cycle operations on the `Customer` entity must be cascaded to the `Address` entity, the target of the relationship. In other words, when a `Customer` instance is made persistent, any `Address` instances related to the `Customer` are also made persistent. It also means that if a `Customer` instance is deleted, the related `Address` instances are also deleted. The `FetchType` value specifies eager fetching -- this tells the container to prefetch the associated entities.

### The owning side specifies the mapping

Notice that the many-to-many relationship between the `Customer` and `Subscription` entity classes is specified using a `@ManyToMany` annotation in both classes and a `@JoinTable` annotation in the `Customer` class. Either `Customer` or `Subscription` could have been specified as the owning class. In this example, `Customer` is the owning class, and so the `@JoinTable` annotation is specified in the that class. The `@ManyToMany` annotation in the `Subscription` class refers to the `Customer` class for mapping information through a `mappedBy` element. Because this example uses a join table for the relationship, the `@JoinTable` annotation specifies the foreign key columns of the join tables that map to the primary key columns of the related entities.

### Default mappings are used for relationships where possible

You can take advantage of default mappings to simplify the coding for relationship mapping even further. If you look at the `create.sql` file under the `setup/sql` directory for the Java Persistence Demo, you will see a join table named `CUSTOMER_ADDRESS`. Notice that no annotations are needed in the `Customer` entity to specify the mapping of the `Customer` and `Address` entities to the columns in the `CUSTOMER_ADDRESS` table. That's because the table name and join column names are the defaults.

### Inheritance and Polymorphism

An important capability of the Java Persistence API is its support for inheritance and polymorphism, something that was not available in EJB 2.1. You can map a hierarchy of entities, where one entity subclasses another, to a relational database structure, and submit queries against the base class. The queries are treated polymorphically against the entire hierarchy.

The code in the EJB 3.0 column below shows an example of the support for inheritance and polymorphism. This sample is not taken from the application code because the Java Persistence Demo does not use this feature. Here, `ValuedCustomer` is an entity that extends the `Customer` entity. The hierarchy is mapped to the `CUSTOMER` table:

EJB 2.1	EJB 3.0
Support not available.	<pre>Entity Class: ValuedCustomer.java  @Entity @Table(name="CUST") @Inheritance(strategy=SINGLE_TABLE) @DiscriminatorColumn(name="DISC", discriminatorType=STRING) @DiscriminatorValue(name="CUSTOMER") public class Customer {...}  @Entity @DiscriminatorValue(name="VCUSTOMER") public class ValuedCustomer extends Customer {...}</pre>

Figure 7: Support for Inheritance and Polymorphism  
Click [here](#) for a larger image

### Annotations specify inheritance

The `@Inheritance` annotation identifies a mapping strategy for an entity class hierarchy. In this example, the strategy specified by the value of the strategy element is `SINGLE_TABLE`. This means that the base class `Customer` and its subclass `ValuedCustomer` are mapped to a single table. By the way, you can also specify a strategy that joins the base class and its subclasses or that maps them to separate tables. A single table strategy requires a discriminator column in the table to distinguish between rows in the table for the base class and rows for the subclasses. The `@DiscriminatorColumn` annotation identifies the discriminator column, in this case, `DISC`. The `discriminatorType` element specifies that the discriminator column contains strings. The `@DiscriminatorValue` annotation is used to specify the value of the discriminator column for the associated entity. Here, `ValuedCustomer` instances are distinguished from other `Customer` instances by having a value of `VCUSTOMER` in the discriminator column.

### Defaults can be used for various inheritance specifications

As is the case in many other parts of EJB 3.0 technology and the Java Persistence API, you can rely on defaults to further simplify coding. For example, `SINGLE_TABLE` is the default inheritance strategy, so the code sample does not need to specify it. In fact, you don't need to specify the `@Inheritance` annotation if you're using single-table inheritance. The default for the `discriminatorType` element in the `@DiscriminatorColumn` annotation is `STRING`, so the specification in the code sample is not necessary there either. Also, for string discriminator columns, you don't need to specify a discriminator value. The default is the entity name.

### Entities can inherit from other entities and from non-entities

The EJB 3.0 example illustrates inheritance from an entity. However inheritance from POJOs that are not entities -- for behavior and mapping attributes -- is also allowed. The base entity or nonentity can either be abstract or concrete.

### Operations on Entities

Another pronounced simplification in the Java Persistence API is the way entity operations are performed. For example, look at the way a subscription is removed in the `editCustomer.jspx` file and the EJB 2.1 `CustomerBean` compared to the way this is done in the business interface for the EJB 3.0 session bean, `CustomerSession`.

EJB 2.1	EJB 3.0
<p><b>JSP File: editCustomer.jsp</b></p> <pre>String cid = request.getParameter("cid"); LocalCustomer customer = null;  try {     InitialContext ic = new InitialContext();     Object o = ic.lookup("java:comp/env/CustomerBeanRef");     LocalCustomerHome home = (LocalCustomerHome) o;     customer = home.findByPrimaryKey(cid); } ... String sub_title = request.getParameter("subscription"); ... String remove = request.getParameter("removeSubscription"); if (sub_title != null &amp;&amp; !"".equals(sub_title)) {     try (...)     else if ("Remove".equals(remove)) {         customer.removeSubscription(sub_title);     }     ... }</pre> <p><b>Entity Bean Class: CustomerBean.java</b></p> <pre>public void removeSubscription (String subscriptionKey) {     try {         Context ic = new InitialContext();         LocalSubscriptionHome home =             (LocalSubscriptionHome)             ic.lookup             ("java:comp/env/ejb/SubscriptionRef");         LocalSubscription subscription =             home.findByPrimaryKey(subscriptionKey);          getSubscriptions().remove(subscription);     } catch (Exception ex) {         ex.printStackTrace();     } }</pre>	<p><b>Session Bean Class: CustomerSession.java</b></p> <pre>public class CustomerSession implements     CustomerSessionLocal, CustomerSessionRemote{      @javax.persistence.PersistenceContext(         unitName="persistence_sample")     private EntityManager em ;      public Customer removeCustomerSubscription(         String cust, String subs)         throws SubscriptionNotFoundException{          Customer customer =             (Customer)em.find(Customer.class, cust);         Subscription subscription =             (Subscription)em.find(Subscription.class, subs);         ...          customer.getSubscriptions().remove(subscription);         subscription.getCustomers().remove(customer);         ...     } }</pre>

Figure 8: Comparing Operations on Entities  
Click [here](#) for a larger image

#### Entity operations are performed directly on the entity

In EJB 2.1 technology, a client must use the Java Naming and Directory Interface (JNDI) to lookup the bean. In doing this, JNDI acquires an object that represents the bean's home interface. In response, the EJB container creates an instance of the bean and initializes its state. The client can then call methods on the JNDI-acquired object to perform operations on the bean. In the EJB 2.1 technology code sample above, JNDI is used to acquire an object that represents the `LocalSubscriptionHome` interface. In the `removeSubscription()` business method of the `CustomerBean`, the `findByPrimaryKey()` method finds the pertinent `SubscriptionBean` instance and removes the reference to it. This relationship update is synchronized to the database when the transaction in which this `CustomerBean`'s business method takes part commits. Many developers view this process as needlessly indirect and complex.

By comparison, the Java Persistence API requires no JNDI-based lookup. An `EntityManager` instance is used to find the pertinent `Customer` and `Subscription` entities. The reference to the `Subscription` instance is then removed from the `Customer`'s subscription list. Also, the reference to the `Customer` instance is removed from the `Subscription`'s customer list -- just as with POJOs.

#### An entity manager manages the entities

An `EntityManager` instance is used to manage the state and life cycle of entities within a persistence context. The entity manager is responsible for creating and removing persistent entity instances and finding entities by the entity's primary key. It also allows queries to be run on entities.

#### Dependencies can be injected

As illustrated in the EJB 3.0 technology code example in Figure 8, JNDI is no longer required to get references to resources and other objects in an enterprise bean's context. Instead, you can use resource and environment reference annotations in the bean class. These annotations inject a resource on which the enterprise bean has a dependency. In the EJB 3.0 technology code sample, the `@PersistenceContext` annotation injects an `EntityManager` with a transactional persistence context, on which the session bean has a dependency. The container then takes care of obtaining the reference to the needed resource and providing it to the bean. Dependency injection can dramatically simplify what you have to code to obtain resource and environmental references.

#### Transactions

Transaction-related specifications are also simplified.

EJB 2.1	EJB 3.0
<p><b>Deployment Descriptor: ejb-jar.xml</b></p> <pre>&lt;assembly-descriptor&gt; &lt;container-transaction&gt; &lt;method&gt; &lt;ejb-name&gt;AddressBean&lt;/ejb-name&gt; &lt;method-intf&gt;Local&lt;/method-intf&gt; &lt;method-name&gt;getZip&lt;/method-name&gt; &lt;/method&gt; &lt;trans-attribute&gt;Required&lt;/trans-attribute&gt; &lt;/container-transaction&gt; &lt;container-transaction&gt; &lt;method&gt; &lt;ejb-name&gt;AddressBean&lt;/ejb-name&gt; &lt;method-intf&gt;LocalHome&lt;/method-intf&gt; &lt;method-name&gt;remove&lt;/method-name&gt; &lt;method-param&gt; &lt;method-param&gt;java.lang.Object &lt;/method-param&gt; &lt;/method-param&gt; &lt;/method&gt; &lt;trans-attribute&gt;Required&lt;/trans-attribute&gt; &lt;/container-transaction&gt; ...</pre>	<p><b>Entity Class: CustomerSession.java</b></p> <pre>@TransactionManagement(     value=TransactionManagementType.CONTAINER)  public class CustomerSession implements     CustomerSessionLocal, CustomerSessionRemote {...}  ...  //This is the default @TransactionalAttribute(     TransactionAttributeType.REQUIRED) public void remove(Object obj){     Object mergedObj = em.merge(obj);     em.remove(mergedObj); }  ...  XML Descriptor: Not required</pre>

Figure 9: Comparing Transaction-Related Specifications  
Click [here](#) for a larger image

#### No XML descriptor needed to specify transaction attributes

In EJB 2.1 technology, you specify the transaction attributes for container-managed transactions in an often lengthy and complex deployment descriptor. In EJB 3.0 technology, an XML descriptor is not needed to specify transaction attributes. Instead, you can use the `@TransactionManagement` annotation to specify container-managed transactions, as well as to specify bean-managed transactions, and the `@TransactionalAttribute` annotation to specify transaction attributes. In the EJB 3.0 technology code sample, the `@TransactionManagement` annotation specifies container-managed transactions for the session bean. Because container-managed transactions are the default type of transaction demarcation, the annotation is not necessary here. The `@TransactionalAttribute` annotation on the `remove()` method specifies a transaction attribute of `REQUIRED`, which is the default transaction type. So this annotation is not necessary either. You would need the annotation, however, to specify another transaction type, such as "Mandatory" or "Supports".

#### Container-managed entity managers are Java Transaction API entity managers

In the Java Persistence API, transactions involving `EntityManager` operations can be controlled in two ways, either through JTA or by the application through the `EntityManager` API. An entity manager whose transactions are controlled through JTA is called a JTA entity manager. An entity manager whose transactions are controlled by the `EntityManager` API is called a resource-local entity manager. A container-managed entity manager must be a JTA entity manager. JTA entity manager transactions are started and ended outside of the entity manager, and the entity manager methods share the transaction context of the session bean methods that invoke them.

## Queries

Support for queries has been significantly enhanced in the Java Persistence API. Some of these enhancements are shown in Figure 10.

EJB 2.1	EJB 3.0
<p>Deployment Descriptor: <code>ejb-jar.xml</code></p> <pre>&lt;entity&gt; ... &lt;query&gt; &lt;query-method&gt; &lt;method-name&gt;findByLastName&lt;/method-name&gt; &lt;method-param&gt; &lt;method-param&gt;java.lang.String&lt;/method-param&gt; &lt;/method-param&gt; &lt;/query-method&gt; &lt;ejb-ql&gt;SELECT DISTINCT OBJECT(c) FROM Customer c WHERE c.lastName=?1&lt;/ejb-ql&gt; &lt;/query&gt;  &lt;query&gt; &lt;query-method&gt; &lt;method-name&gt;findByFirstName&lt;/method-name&gt; &lt;method-param&gt; &lt;method-param&gt;java.lang.String&lt;/method-param&gt; &lt;/method-param&gt; &lt;/query-method&gt; &lt;ejb-ql&gt;SELECT DISTINCT OBJECT(c) FROM Customer AS c WHERE c.firstName=?1&lt;/ejb-ql&gt; &lt;/query&gt; &lt;/entity&gt;</pre>	<p>Entity Class: <code>Customer.java</code></p> <pre>@Entity @NamedQuery(     value={@NamedQuery(name="findCustomerByFirstName",         query="select object(c) from Customer c         where c.firstName=:firstName"),         @NamedQuery(name="findCustomerByLastName",         query="select object(c) from Customer c         where c.lastName=:lastName")} )  public class Customer implements java.io.Serializable{ ... }</pre> <p>Session Bean Class: <code>CustomerSession.java</code></p> <pre>... public List findCustomerByFirstName(String firstName){     List customers =         em.createNamedQuery("findCustomerByFirstName")             .setParameter("firstName", firstName)             .getResultList();     return customers; }  public List findCustomerByLastName(String lastName){     List customers =         em.createNamedQuery("findCustomerByLastName")             .setParameter("lastName", lastName)             .getResultList();     return customers; } ... }</pre> <p>XML Descriptor: Not required</p>

Figure 10: Comparing Query Specifications  
Click [here](#) for a larger image

## No XML descriptor needed to specify queries

In EJB 2.1 technology, you define a query for an entity bean using Enterprise JavaBeans query language (EJB QL). You specify the query in the deployment descriptor for the bean, and associate it there with a finder or select method for the bean. In the Java Persistence API, you can define a named -- or static -- query in the bean class itself. You also have the option of creating dynamic queries. To create a named query, you first define the named query using the `NamedQuery` annotation. Then you create the previously-defined query using the `createNamedQuery` method of the `EntityManager`. In the EJB 3.0 technology example, two named queries are defined in the `Customer` entity class: `findCustomerByFirstName` and `findCustomerByLastName`. The named queries are created in the session bean, `CustomerSession`, which provides the client code for the entity. To create a dynamic query, you use the `createQuery` method of the `EntityManager`. The Java Persistence API provides a Java Persistence query language that extends EJB QL. You can use Java Persistence query language or native SQL in named or dynamic queries.

## Support for dynamic queries and named queries is added

The Java Persistence API provides a Query API to create dynamic queries and named queries. For example, the `CustomerSession` class uses the Query API on the entity manager to create the named queries `findCustomerByFirstName` and `findCustomerByLastName`. The Query method `setParameter` binds an argument to a named parameter. (By the way, support for named parameters is also a new feature in the Java Persistence API: both named queries and dynamic queries can use named parameters as well as positional parameters, although a single query cannot mix both types of parameters.) For example, the `setParameter` method in `findCustomerByFirstName` binds the `firstName` argument to the named parameter `: firstName` in the named query definition. The `getResultList` method returns the query results.

## All queries that use the Java Persistence API are polymorphic

This means that when a class is queried, all subclasses that meet the query criteria are also returned.

## The Java Persistence Query Language is an enhanced query language

EJB QL has been a very popular facet of EJB technology. However, despite its popularity, EJB QL has lacked some of the features of a full structured query language, such as bulk update and delete operations, outer join operations, projection, and subqueries. The Java Persistence query language adds those features. It also adds support for outer join-based prefetching. The full range of the Java Persistence query language can be used in static or dynamic queries. However the Java Persistence Demo does not use the Java Persistence query language enhancements.

## Testing Entities Outside of the Container

Although this can't be demonstrated in a side-by-side code comparison, testing entities outside an EJB container is now much easier. Previously, the entity bean component model -- with its requirements for home and component interfaces, abstract entity bean classes, and virtual persistent fields -- made it difficult to test entity beans outside of the container. The Java Persistence API removes the requirement for these interfaces. The only thing required for an entity bean is a concrete entity bean class -- a POJO -- that has persistent fields or persistent properties. In addition, an entity's life cycle is controlled through the entity manager, not through a home interface whose life-cycle methods are implemented by the EJB container. Because all of this makes entities less dependent on intervention by the EJB container, the entities can be more easily tested outside of the container.

## Summary

The aim of the new Java Persistence API is to simplify the development of persistent entities. It meets this objective through a simple POJO-based persistence model, which reduces the number of required classes and interfaces. You model your data using POJOs, and then annotate them to tell the container about an entity's characteristics and the resources it needs. You also use annotations for object-relational mappings and entity relationships, as well as deploy-time instructions. This annotation-based approach removes the need for often long and complex, XML-based descriptors. In many cases, the annotations' defaults are enough. You code specific attributes of the annotations only when the defaults are inadequate.

Beyond these simplifications, the Java Persistence API adds capabilities that were not in EJB 2.1 technology, giving you additional power and flexibility in developing and using persistent entities. You can take advantage of query language enhancements and new features such as inheritance and polymorphism to perform more powerful and encompassing queries. You can exercise more control over queries, and perform query optimizations specific to your needs. In short, using the Java Persistence API is much simpler and more intuitive than it's predecessors, yet it offers a more robust API for creating, managing, and using persistent entities.

This article only highlights the simplifications and enhancements that the Java Persistence API offers. You'll find more extensive information about the API, as well as other simplifications made in EJB 3.0 technology, by examining the [Enterprise JavaBeans 3.0 Specification](#). Now is a good time to try out EJB 3.0 technology and the Java Persistence API. See the [Try It Out!](#) box at the beginning of this article for some good places to start.

## For More Information

[EJB 3.0 Specification \(JSR-220\)](#)

[Enterprise JavaBeans Technology](#)

[Introduction to the Java EE 5 Platform](#)

[Ease of Development in Enterprise JavaBeans Technology](#)

[The Java EE 5 Tutorial](#)

[Annotations](#)

[Java EE 5 SDK Preview](#)

[Preview: NetBeans IDE 5.5 with NetBeans Enterprise Pack 5.5](#)

[Project GlassFish](#)

#### About the Authors

**Rahul Biswas** is a member of the Java Performance Engineering group at Sun. He is currently involved in the development of a generic performance benchmark for Java Persistence and the performance improvement of the persistence implementation in GlassFish.

**Ed Ort** is a staff member of [java.sun.com](http://java.sun.com). He has written extensively about relational database technology, programming languages, and web services.

#### Rate and Review

Tell us what you think of the content of this page.

☐ **Excellent** ☐ **Good** ☐ **Fair** ☐ **Poor**

Comments:

</td>  
</tr>  
<tr>



[HOME](#) [PRODUCTS](#) [FORUM](#) [ISSUES](#) [WIKI](#) [BLOG](#) [DOWNLOAD](#)
[Home](#) | [Products](#) | [Plugins](#) | [Plugin Points](#) | [Development](#)


## Overview

### Welcome

[Download](#)  
[Publicity](#)  
[Used by](#)  
[Performance](#)  
[Presentations](#)  
[Blog](#)  
[License](#)  
[References](#)

## Products

[AccessPlatform](#)

## Support

[Support](#)  
[Services](#)  
[Problem Reporting](#)  
[Contributing](#)  
[Donations](#)  
[Release Schedule](#)

## Third-Party Tools

[Third Party Products](#)  
[Open Source](#)  
[JDOCurrent](#)  
[Eclipse Dali](#)  
[EMSoft Data JDO](#)  
[Closed Source but Free](#)  
[Kumquat DAO](#)  
[Commercial](#)  
[Starwind - High Availability Software](#)

## DataNucleus

The DataNucleus project provides products for the management of application data in a Java environment. Our aim is to provide good quality open source products to handle data in all of its forms, wherever it is stored. This goes from **persistence of data into heterogeneous datastores**, to providing methods of **retrieval using a range of query languages**, and eventually we will move on to the analysis of data and tools for managing data quality. Your use of DataNucleus products will mean that you don't need to take significant time in learning the oddities of particular datastores, or query languages and instead use a **single common interface for all of your data**, and instead your team can concentrate their application development time on adding business logic and let DataNucleus take care of data management issues. **DataNucleus products comply with any pertinent standards in their problem domain**, and all are provided under the [Apache 2 open source license](#). This means that you are **safeguarding your applications future to standards and to openness** rather than coding to some proprietary API and have to suffer a refactor lag whenever you decide to change vendor of data persistence for example. DataNucleus is truly independent unlike all significant competing software that is associated with a large commercial organisation that could stop development at any moment. **It's time to free up your data access layer**



**Access Platform** is our baseline product. It is the most standards-compliant Open Source Java persistence product in existence. It is fully compliant with the JDO1, JDO2, JDO2.1, JDO2.2, JDO3, JPA1 and JPA2 Java standards. It also complies with the OGC Simple Feature Specification for persistence of geospatial Java types to RDBMS. It utilises an OSGi-based plugin mechanism meaning that it is extremely extensible. It is also free for use. The benefits for your business are significant.

[\[Read More\]](#) | [\[Download\]](#) | [\[Get Started\]](#)

## DataNucleus News

17/January/2011	<a href="#">AccessPlatform 2.2.1</a> is released
18/December/2010	<a href="#">AccessPlatform 2.1.4</a> is released
10/December/2010	<a href="#">AccessPlatform 2.2.0 RELEASE</a> is released
12/November/2010	<a href="#">AccessPlatform 2.2.0 M3</a> is released



( [Sign In/Register for Account](#) | [Help](#) )

United States ▼

Communities ▼

I am a... ▼

I want to... ▼

Secure Search



Sun Quick Links ▼

[Products and Services](#)

[Downloads](#)

[Store](#)

[Support](#)

[Education](#)

[Partners](#)

[About](#)

[Oracle Technology Network](#)

[Oracle Technology Network](#) > [Java](#)

[Java SE](#)

[Java for Business](#)

[Java Embedded](#)

[Java EE](#)

[Java ME](#)

[JavaFX](#)

[Java DB](#)

[Web Tier](#)

[Java Card](#)

[Java TV](#)

## JavaServer Pages Standard Tag Library

The JavaServer Pages Standard Tag Library (JSTL) encapsulates as simple tags the core functionality common to many Web applications. JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.

The JSTL 1.2 Maintenance Release aligns with the Unified Expression Language (EL) that is being delivered as part of the JavaServer Pages (JSP) 2.1 specification. Thanks to the Unified EL, JSTL tags, such as the JSTL iteration tags, can now be used with JavaServer Faces components in an intuitive way.

JSTL 1.2 is part of the Java EE 5 platform.

[What's New](#)

**[JSTL Project](#)** Go to the JSTL project for the latest API and implementation of JSTL 1.2.

**[JSTL 1.2 Maintenance Review Specification Available!](#)** Find out how changes to the JSTL specification help support the alignment of Java-based web-tier technologies.

[Community](#)

To learn about Webtier technologies in GlassFish, please visit the [GlassFish Webtier page](#) and post questions on the [GlassFish Webtier forum](#) or send e-mail to [webtier@glassfish.dev.java.net](mailto:webtier@glassfish.dev.java.net)

JSP Tag Library Developers! The [JavaServer Pages Technology Forum](#) is a great place to learn more about developing with JSP technology.

### Java SDKs and Tools

[Java SE](#)

[Java EE and Glassfish](#)

[Java ME](#)

[JavaFX](#)

[Java Card](#)

[NetBeans IDE](#)

### Java Resources

[New to Java?](#)

[APIs](#)

[Code Samples & Apps](#)

[Developer Training](#)

[Documentation](#)

[Java BluePrints](#)

[Java.com](#)

[Java.net](#)

[Student Developers](#)

[Tutorials](#)

**Hardware and Software**  
Engineered to Work Together

[About Oracle](#) | [Oracle and Sun](#) | [Subscribe](#) | [Careers](#) | [Contact Us](#) | [Site Maps](#) | [Legal Notices](#) | [Terms of Use](#) | [Your Privacy Rights](#)