**1****Servlet Overview**

Oracle Application Server Containers for J2EE (OC4J) enables you to develop and deploy standard J2EE-compliant applications. Applications are packaged in standard Enterprise archive (EAR) deployment files, which include standard Web archive (WAR) files to deploy the Web modules, and Java archive (JAR) files for any Enterprise JavaBeans (EJB) and application client modules in the application.

With Oracle Application Server 10*g* Release 2 (10.1.2), OC4J complies with *Java 2 Platform Enterprise Edition Specification, v1.3*, including full compliance with the Sun Microsystems *Java Servlet Specification, Version 2.3* in the OC4J servlet container. (Any mention of the servlet specification in this manual refers to this version unless otherwise noted.)

The most important concepts to understand about servlet development under OC4J are how a Web application is built and how it is deployed. If you are new to servlets, see [Chapter 2, "Servlet Development"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/develop.htm#g1036792). If OC4J is a new development environment for you, see [Chapter 5, "Deployment and Configuration Overview"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/deploy.htm#g1051990), to learn how applications are deployed under OC4J.

This chapter introduces the Java servlet and provides an example of a basic servlet. It also briefly discusses how you can use servlets in a J2EE application to address some server-side programming issues.

This chapter contains the following sections:

* [Introduction to Servlets](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1001303)
* [A First Servlet Example](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i999676)

|  |
| --- |
| **Note:**  Sample servlet applications are included in the OC4J demos, available from the following location on the Oracle Technology Network (requiring an OTN membership, which is free of charge):  <http://www.oracle.com/technology/tech/java/oc4j/demos/> |

**Introduction to Servlets**

The following sections offer a brief introduction to servlet technology:

* [Review of Servlet Technology](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1001676)
* [Advantages of Servlets](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1003017)
* [The Servlet Interface and Request and Response Objects](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1001377)
* [Servlets and the Servlet Container](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1001694)
* [Introduction to Servlet Sessions](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1001571)
* [Introduction to Servlet Contexts](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1001960)
* [Introduction to Servlet Configuration Objects](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1003139)
* [Introduction to Servlet Filters](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1003195)
* [Introduction to Event Listeners](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1001707)
* [JSP Pages and Other J2EE Component Types](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1002471)

|  |
| --- |
| **No****te:**  The terms *Web module* and *Web application* are interchangeable in most uses and are both used throughout this document. If there is a distinction, it is that "Web module" typically indicates a single component, whether or not it composes an independent application, while "Web application" typically indicates a working application that may consist of multiple modules or components. |

**Review of Servlet Technology**

In recent years, servlet technology has emerged as a powerful way to extend Web server functionality through dynamic Web pages. A servlet is a Java program that runs in a Web server, as opposed to an applet that runs in a client browser. Typically, the servlet takes an HTTP request from a browser, generates dynamic content (such as by querying a database), and provides an HTTP response back to the browser. Alternatively, the servlet can be accessed directly from another application component or send its output to another component. Most servlets generate HTML text, but a servlet may instead generate XML to encapsulate data.

More specifically, a servlet runs in a J2EE application server, such as OC4J. Servlets are one of the main application component types of a J2EE application, along with JavaServer Pages (JSP) and EJB modules, which are also server-side J2EE component types. These are used in conjunction with client-side components such as applets (part of the Java 2 Platform, Standard Edition specification) and application client programs. An application may consist of any number of any of these components.

Prior to servlets, Common Gateway Interface (CGI) technology was used for dynamic content, with CGI programs being written in languages such as Perl and being called by a Web application through the Web server. CGI ultimately proved less than ideal, however, due to its architecture and scalability limitations.

**Advantages of Servlets**

In the Java realm, servlet technology offers advantages over applet technology for server-intensive applications, such as those accessing a database. One advantage of running in the server is that the server is usually a robust machine with many resources, making the program more scalable. Running in the server also results in more direct access to the data. The Web server in which a servlet is running is on the same side of the network firewall as the data being accessed.

Servlet programming also offers advantages over earlier models of server-side Web application development, including the following:

* Servlets outperform earlier technologies for generating dynamic HTML, such as server-side "includes" or CGI scripts. After a servlet is loaded into memory, it can run on a single lightweight thread; CGI scripts must be loaded in a different process for every request.
* Servlet technology, in addition to improved scalability, offers the well-known Java advantages of security, robustness, object orientation, and platform independence.
* Servlets are fully integrated with the Java language and its standard APIs, such as JDBC for Java database connectivity.
* Servlets are fully integrated into the J2EE framework, which provides an extensive set of services that your Web application can use, such as Java Naming and Directory Interface (JNDI) for component naming and lookup, Java Transaction API (JTA) for managing transactions, Java Authentication and Authorization Service (JAAS) for security, Remote Method Invocation (RMI) for distributed applications, and Java Message Service (JMS). The following Web site contains information about the J2EE framework and services:
* <http://java.sun.com/j2ee/docs.html>
* A servlet handles concurrent requests (through either a single servlet instance or multiple servlet instances, depending on the thread model), and servlets have a well-defined lifecycle. In addition, servlets can optionally be loaded when OC4J starts, so that any initialization is handled in advance instead of at the first user request. See ["Servlet Preloading"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/develop.htm#i1006550).
* The servlet request and response objects offer a convenient way to handle HTTP requests and send text and data back to the client.

Because servlets are written in the Java programming language, they are supported on any platform that has a Java virtual machine (JVM) and a Web server that supports servlets. Servlets can be used on different platforms without recompiling. You can package servlets together with associated files such as graphics, sounds, and other data to make a complete Web application. This simplifies application development and deployment.

In addition, you can port a servlet-based application from another Web server to OC4J with little effort. If your application was developed for a J2EE-compliant Web server, then the porting effort is minimal.

**The Ser****vlet Interface and Request and Response Objects**

A Java servlet, by definition, implements the javax.servlet.Servlet interface. This interface specifies methods to initialize a servlet, process requests, get the configuration and other basic information of a servlet, and terminate a servlet instance.

For Web applications, you can implement the Servlet interface by extending the javax.servlet.http.HttpServlet abstract class. (Alternatively, for protocol-independent servlets, you can extend the javax.servlet.GenericServlet class.) The HttpServlet class includes the following methods:

* init(...): Initialize the servlet.
* destroy(...): Terminate the servlet.
* doGet(...): Execute an HTTP GET request.
* doPost(...): Execute an HTTP POST request.
* doPut(...): Execute an HTTP PUT request.
* doDelete(...): Execute an HTTP DELETE request.
* service(...): Receive HTTP requests and, by default, dispatch them to the appropriate do*XXX*() methods.
* getServletInfo(...): Retrieve information about the servlet.

A servlet class that extends HttpServlet implements some or all of these methods, as appropriate, overriding the original implementations as necessary to process the request and return the response as desired. For example, most servlets override the doGet() method, doPost() method, or both to process HTTP GET and POST requests.

Each method takes as input an HttpServletRequest instance (an instance of a class that implements the javax.servlet.http.HttpServletRequest interface) and an HttpServletResponse instance (an instance of a class that implements the javax.servlet.http.HttpServletResponse interface).

The HttpServletRequest instance provides information to the servlet regarding the HTTP request, such as request parameter names and values, the name of the remote host that made the request, and the name of the server that received the request. The HttpServletResponse instance provides HTTP-specific functionality in sending the response, such as specifying the content length and MIME type and providing the output stream.

**Servlets and the Ser****vlet Container**

Unlike a Java client program, a servlet has no static main() method. Therefore, a servlet must execute under the control of an external container.

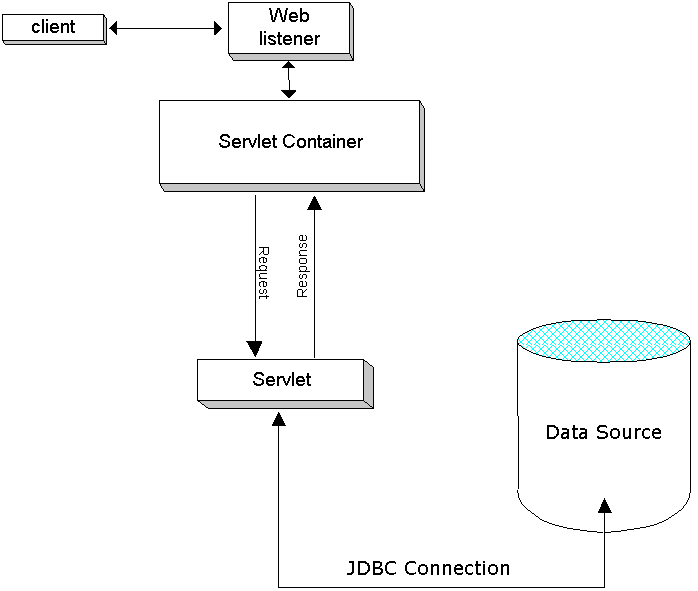
*Servlet containers*, sometimes referred to as *servlet engines*, execute and manage servlets. The servlet container calls servlet methods and provides services that the servlet needs while executing. A servlet container is usually written in Java and is either part of a Web server (if the Web server is also written in Java) or is otherwise associated with and used by a Web server. OC4J includes a fully standards-compliant servlet container.

The servlet container provides the servlet with easy access to properties of the HTTP request, such as its headers and parameters. When a servlet is called, such as when it is specified by URL, the Web server passes the HTTP request to the servlet container. The container, in turn, passes the request to the servlet. In the course of managing a servlet, a servlet container performs the following tasks:

* It creates an instance of the servlet and calls its init() method to initialize it.
* It constructs a request object to pass to the servlet. The request includes, among other things:
  + Any HTTP headers from the client
  + Parameters and values passed from the client (for example, names and values of query strings in the URL)
  + The complete URI of the servlet request
* It constructs a response object for the servlet.
* It invokes the servlet service() method. Note that for HTTP servlets, the generic service method is usually overridden in the HttpServlet class. The service method dispatches requests to the servlet doGet() or doPost() methods, depending on the HTTP header in the request (GET or POST).
* It calls the destroy() method of the servlet to discard it, when appropriate, so that it can be garbage collected. (For performance reasons, it is typical for a servlet container to keep a servlet instance in memory for reuse, rather than destroying it each time it has finished its task. It would be destroyed only for infrequent events, such as Web server shutdown.)

[Figure 1-1](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1001227) shows how a servlet relates to the servlet container and to a client, such as a Web browser. When the Web listener is the Oracle HTTP Server (powered by Apache), the connection to the OC4J servlet container goes through the mod\_oc4j module. See the[*Oracle HTTP Server Administrator's Guide*](https://docs.oracle.com/cd/B14099_19/web.1012/b14007/toc.htm)for details.

***Figure 1-1 Servlets and the Servlet Container***

  
[Description of "Figure 1-1 Servlets and the Servlet Container"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/img_text/fc01.htm)

**Introduction to Serv****let Sessions**

Servlets use HTTP sessions to keep track of which user each HTTP request comes from, so that a group of requests from a single user can be managed in a stateful way. Servlet session tracking is similar in nature to session tracking in previous technologies, such as CGI.

This section provides an introduction to servlet sessions. See ["Servlet Sessions"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/develop.htm#i1005093) for more information and examples.

**Introduction to Ses****sion Tracking**

Servlets provide convenient ways to keep the client and a server session in synchronization, enabling stateful servlets to maintain session state on the server over the whole duration of a client browsing session.

OC4J supports the following session-tracking mechanisms. See ["Session Tracking"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/develop.htm#i1005050) for more information.

* Cookies

The servlet container sends a cookie to the client, which returns the cookie to the server upon each HTTP request. This process associates the request with the session ID indicated by the cookie. This is the most frequently used mechanism and is supported by any servlet container that adheres to the servlet specification.

* URL rewriting

Instead of using cookies, the servlet can call the encodeURL() method of the response object, or the encodeRedirectURL() method for redirects, to append a session ID to the URL path for each request. This process allows the request to be associated with the session. This is the most frequently used mechanism for situations in which clients do not accept cookies.

**Introduction to the HttpSes****sion Interface**

In the standard servlet API, each client session is represented by an instance of a class that implements the javax.servlet.http.HttpSession interface. Servlets can set and get information about the session in this HttpSession object, which must be of application-level scope.

A servlet uses the getSession() method of an HttpServletRequest object to retrieve or create an HttpSession object for the user. This method takes a boolean argument to specify whether a new session object should be created for the client if one does not already exist within the application.

See ["Features of the HttpSession Interface"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/develop.htm#i1009721) for more information.

**Introduction to Ser****vlet Contexts**

A *servlet context* is used to maintain information for all instances of a Web application within any single JVM (that is, for all servlet and JSP page instances that are part of the Web application). There is one servlet context for each Web application running within a given JVM; this is always a one-to-one correspondence. You can think of a servlet context as a container for a specific application.

**Servlet Context Basics**

Any servlet context is an instance of a class that implements the javax.servlet.ServletContext interface, with such a class being provided with any Web server that supports servlets.

A ServletContext object provides information about the servlet environment (such as name of the server) and allows sharing of resources between servlets in the group, within any single JVM. (For servlet containers supporting multiple simultaneous JVMs, implementation of resource-sharing varies.)

A servlet context provides the scope for the running instances of the application. Through this mechanism, each application is loaded from a distinct classloader and its runtime objects are distinct from those of any other application. In particular, the ServletContext object is distinct for an application, much as each HttpSession object is distinct for each user of the application.

Beginning with version 2.2 of the servlet specification, most implementations can provide multiple servlet contexts within a single host, which is what allows each Web application to have its own servlet context. (Previous implementations usually provided only a single servlet context with any given host.)

**How to Obtain a Servlet Context**

Use the getServletContext() method of a servlet configuration object to retrieve a servlet context. See ["Introduction to Servlet Configuration Objects"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1003139).

**Servlet Context Methods**

The ServletContext interface specifies methods that allow a servlet to communicate with the servlet container that runs it, which is one of the ways that the servlet can retrieve application-level environment and state information. Methods specified in ServletContext include those listed here. For complete information, refer to the Sun Microsystems Javadoc at the following location:

<http://java.sun.com/products/servlet/2.3/javadoc/index.html>

* void setAttribute(String name, Object value)

This method binds the specified object to the specified attribute name in the servlet context. Using attributes, a servlet container can give information to the servlet that is not otherwise provided through the ServletContext interface.

|  |
| --- |
| **Note:**  For a servlet context, setAttribute() is a local operation only. It is not intended to be distributed to other JVMs within a cluster. (This is in accordance with the servlet specification.) |

* Object getAttribute(String name)

This method returns the attribute with the given name, or null if there is no attribute by that name. The attribute is returned as a java.lang.Object instance.

* java.util.Enumeration getAttributeNames()

This method returns a java.util.Enumeration instance containing the names of all available attributes of the servlet context.

* void removeAttribute(String attrname)

This method removes the specified attribute from the servlet context.

* String getInitParameter(String name)

This method returns a string that indicates the value of the specified context-wide initialization parameter, or null if there is no parameter by that name. This allows access to configuration information that is useful to the Web application associated with this servlet context.

* Enumeration getInitParameterNames()

This method returns a java.util.Enumeration instance containing the names of the initialization parameters of the servlet context.

* RequestDispatcher getNamedDispatcher(String name)

This method returns a javax.servlet.RequestDispatcher instance that acts as a wrapper for the specified servlet.

* RequestDispatcher getRequestDispatcher(String path)

This method returns a javax.servlet.RequestDispatcher instance that acts as a wrapper for the resource located at the specified path.

* String getRealPath(String path)

This method returns the real path, as a string, for the specified virtual path.

* URL getResource(String path)

This method returns a java.net.URL instance with a URL to the resource that is mapped to the specified path.

* String getServerInfo()

This method returns the name and version of the servlet container.

* String getServletContextName()

This method returns the name of the Web application with which the servlet context is associated, according to the <display-name> element of the web.xml file.

**Introduction to Ser****vlet Configuration Objects**

A *servlet configuration object* contains initialization and startup parameters for a servlet and is an instance of a class that implements the javax.servlet.ServletConfig interface. Such a class is provided with any J2EE-compliant Web server.

You can retrieve a servlet configuration object for a servlet by calling the getServletConfig() method of the servlet. This method is specified in the javax.servlet.Servlet interface, with a default implementation in the javax.servlet.http.HttpServlet class.

The ServletConfig interface specifies the following methods:

* ServletContext getServletContext()

Retrieve a servlet context for the application. See ["Introduction to Servlet Contexts"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm#i1001960).

* String getServletName()

Retrieve the name of the servlet.

* Enumeration getInitParameterNames()

Retrieve the names of the initialization parameters of the servlet, if any. The names are returned in a java.util.Enumeration instance of String objects. (The Enumeration instance is empty if there are no initialization parameters.)

* String getInitParameter(String name)

This returns a String object containing the value of the specified initialization parameter, or null if there is no parameter by that name.

**Introduction to Servlet Filt****ers**

Request objects (instances of a class that implements HttpServletRequest) and response objects (instances of a class that implements HttpServletResponse) are typically passed directly between the servlet container and a servlet.

The servlet specification, however, allows *servlet filters*, which are Java programs that execute on the server and can be interposed between the servlet (or group of servlets) and the servlet container for special request or response processing.

If there is a filter or a chain of filters to be invoked before the servlet, these are called by the container with the request and response objects as parameters. The filters pass these objects, perhaps modified, or alternatively create and pass new objects, to the next object in the chain using the doChain() method.

See ["Servlet Filters"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/filters.htm#i1000648) for more information.

**Introduction to Ev****ent Listeners**

The servlet specification adds the capability to track key events in your Web applications through *event listeners*. This functionality allows more efficient resource management and automated processing based on event status.

When creating listener classes, you can implement standard interfaces for servlet context lifecycle events, servlet context attribute changes, HTTP session lifecycle events, and HTTP session attribute changes. A listener class can implement one, some, or all of the interfaces as appropriate.

An event listener class is declared in the web.xml deployment descriptor and invoked and registered upon application startup. When an event occurs, the servlet container calls the appropriate listener method.

See ["Event Listeners"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/filters.htm#i1000654) for more information.

**JSP Pages and Other J2EE Component Types**

In addition to servlets, an application may include other server-side components, such as JSP pages and EJBs. It is especially common for servlets to be used in combination with JSP pages in a Web application. Servlets are managed by the OC4J servlet container; EJBs are managed by the OC4J EJB container; and JSP pages are managed by the OC4J JSP container. These containers form the core of OC4J.

JSP pages also involve the servlet container, because the JSP container itself is a servlet and is therefore executed by the servlet container. The JSP container translates JSP pages into page implementation classes, which are executed by the JSP container and are also essentially servlets.

|  |
| --- |
| **Note:**  Wherever this manual mentions functionality that applies to servlets, you can assume it applies to JSP pages as well unless stated otherwise. |

For more information about JSP pages and EJBs, see the following:

* [*Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide*](https://docs.oracle.com/cd/B14099_19/web.1012/b14014/toc.htm)
* [*Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*](https://docs.oracle.com/cd/B14099_19/web.1012/b15505/toc.htm)

**A First Se****rvlet Example**

Looking at a basic example is the best way to demonstrate the general framework for writing a servlet.

**Hello World Code**

This servlet prints "Hi There!" back to the client. The comments note some of the basic aspects of writing a servlet.

// You must import at least the following packages for any servlet you write.

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet, the base servlet implementation.

public class HelloServlet extends HttpServlet {

// Override the base implementation of doGet(), as desired.

public void doGet (HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

// Set the MIME type for the response content.

resp.setContentType("text/html");

// Get an output stream to use in sending the output to the client.

ServletOutputStream out = resp.getOutputStream();

// Put together the HTML code for the output.

out.println("<html>");

out.println("<head><title>Hello World</title></head>");

out.println("<body>");

out.println("<h1>Hi There!</h1>");

out.println("</body></html>");

}

}

**Compiling and Deploying the Servlet**

To try out the sample servlet code in an OC4J standalone environment, save it as HelloServlet.java in the /WEB-INF/classes directory of the OC4J default Web application. (See ["OC4J Default Application and Default Web Application"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/deploy.htm#i1027892).)

Next, compile the servlet. First verify that servlet.jar, supplied with OC4J, is in your classpath. This contains the Sun Microsystems javax.servlet and javax.servlet.http packages.

|  |
| --- |
| **Note:**  For convenience during development and testing, use the OC4J auto-compile feature for servlet code. This is enabled through the setting development="true" in the <orion-web-app> element of the global-web-application.xml file in the OC4J configuration files directory. The source-directory attribute may also have to be set appropriately. With auto-compile enabled, after you change the servlet source and save it in the appropriate directory, the OC4J server automatically compiles and redeploys the servlet the next time it is invoked.  See ["Element Descriptions for global-web-application.xml and orion-web.xml"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/confdesc.htm#i1019005) for more information about development and source-directory. |

**Running the Servlet**

Assuming that the OC4J server is up and running and that invocation by class name is enabled with the servlet-webdir built-in default setting of "/servlet/", you can invoke the servlet and see its output from a Web browser as follows, where *host* is the name of the host that the OC4J server is running on and *port* is the Web listener port:

http://*host*:*port*/servlet/HelloServlet

(See ["Servlet Invocation by Class Name During OC4J Development"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/develop.htm#i1007463) for information about invocation by class name and about the OC4J servlet-webdir attribute.)

In an OC4J standalone environment, use port 8888 to access the OC4J Web listener directly. (See ["OC4J Standalone for Development"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/develop.htm#i1015926) for an overview.)

This example assumes that "/" is the context path of the Web application, as is true by default in OC4J standalone for the default Web application.

|  |
| --- |
| **Important:**  The way of invoking servlets that is shown here invokes directly by class name. This is suitable for a development environment but presents a significant security risk. Do not configure OC4J to operate in this mode in a production environment. See ["Servlet Invocation by Class Name During OC4J Development"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/develop.htm#i1007463) and ["Additional Security Considerations"](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/develop.htm#i1009827) for more information. |