Java Embedded Server™ Developer Guide

Version 2.0

Preliminary Early Chapters

Updates Available at
http://www.sun.com/software/embedderserver

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

August 2000, Revision 01

Send comments about this document to: jes-comments@sun.com

Please
Recycle

Contents

CHAPTER 1

JES Administration

This chapter describes how to use the framework console and the JES Management Panel to

- manage the bundle life cycle in the Java Embedded Server framework
- obtain information about installed bundles
- get and set system properties and OSGI environment variables

▼ Launching the Framework

1. **Set the *JES_JAVA_HOME* environment variable to the pathname of the Java installation you wish to use.**

   On Solaris, `setenv JES_JAVA_HOME` *path_to_java_install_dir*. For example,

   ```
   % setenv JES_JAVA_HOME /usr/bin/jdk1.2.2-006
   ```

   On Windows NT, `set JES_JAVA_HOME=`*path_to_java_install_dir*. For example,

   ```
   % set JES_JAVA_HOME=c:\pjava302
   ```

   If you are using the Java 2 platform, we recommend version 1.2.2-006 or greater.

2. **Use the `runjes` command to start the framework.**

   ```
   % cd jes_install_dir
   % bin/runjes
   ```

The JES welcome and framework prompt appear.

```
Java Embedded Server 2.0
Copyright 1998, 1999 and 2000 Sun Microsystems, Inc. All rights
reserved.
Use is subject to license terms.
Type 'h[elp]' for a list of commands.
>
```

Syntax for runjes and runjes.bat

`runjes[.bat] [-cacheDir` *pathname*`] [-dryrun] [-j` *jvm_option*`]...[`*commands*`]`

TABLE 1-1 Options to the runjes Command

Option	Description
`-cacheDir` *pathname*	The pathname of the directory JES should use for its cache. The default is `/home/`*user_name*`/jescache` on Solaris and Linux and %*HOME*%`\jescache` on Windows NT. The value of variable *HOME* must not contain any quotes. The value of *pathname* is assigned to the property `com.sun.jes.framework.cache.dir`.
`-dryrun`	Show what would be done without actually doing it. You may want to use the `-dryrun` option to examine what the runjes utility does when it is called.
`-j` *jvm_option*	Pass the *jvm_option* to the Java interpreter. By default, the following three system properties are defined. `com.sun.jes.framework.bundles.baseurl` is set to `file:`*install_dir*`/bundles`, where *install_dir* is the path to the JES installation directory. This allows you to use short names (instead of fully specifying the pathname) when referring to system bundles, for example, `install log servlet http`. `com.sun.jes.impl.keystore.store` is set to `file:`*install_dir*`/lib/tlscerts` and `com.sun.jes.impl.keystore.access` is set to *password* where *install_dir* is the path to the JES installation directory and *password* is the the encoded password needed to access the keystore. The keystore is an example keystore that is used in the SSL examples. You can override these properties to point to your own information.
*command*s	This can be the pathname or URL of a file from which to read JES commands, or a list of JES commands. If a command contains a blank, it must be quoted when used on the `runjes` command line. For example, `runjes "install log"`. If no commands are specified, then commands are read from stdin.

The Framework Cache

The framework maintains a cache in a directory on the local file system. This directory stores the actual bundle files that have been installed as well as the status of each bundle. When the framework instance is stopped, this cache directory is not removed. You can then start the framework again, and it restores itself with all bundles in the same state as they were just prior to shutdown, using the information in this cache directory.

The `runjes` command launches a framework instance as a new process. You should include the `cacheDir` option to explicitly specify which instance (that is, which cache) should be started. In this example, the framework that is dormant in `~/myCacheDir` is launched.

```
% runjes -cacheDir ~/myCacheDir
```

If there is no cache directory and you do not specify one, `runjes` creates one named *path_to_home*`/jescache` where *path_to_home* is the value of the Java system property `user.home`. You can change the default cache directory location by setting the `com.sun.jes.framework.cache.dir` system property.

You may find it useful to remove a cache directory, especially when developing bundles. You can remove the framework cache just as you would any directory in your file system.

Managing the Bundle Life Cycle

Once you have the framework running, you can begin using it to install, start, update, stop, and uninstall bundles, managing the bundle life cycle.

Bundle States

During its life cycle, a bundle may be in one of the following states.

TABLE 1-2 Bundle States

State	Description
INSTALLED	The bundle has been successfully installed.
RESOLVED	All Java classes and native code that this bundle requires have been made available.
STARTING	The bundle is being started

TABLE 1-2 Bundle States

State	Description
STOPPING	The bundles is being stopped
ACTIVE	The bundle has successfully started and is running.
UNINSTALLED	The bundle has been uninstalled.

Framework Commands

You can use the following framework commands

- on the `runjes` command line (see "Shortcuts to Starting Up" on page 15)
- in a file
- at the framework prompt.

TABLE 1-3 Framework Commands

Command Name	Description
`b[undles]`	List all installed bundles and their status.
`e[xportedpackages]`	List exported packages and the bundles that export and import them respectively.
`g[et]` *property_name*	Get the value of the specified property.
`h[elp]` [*command*]	List all commands and their options.
`i[nstall]` *bundle_url* `[, ...]`	Install the specified bundles.
`m[anifest]` *bundle_url* `|` *bundle_id*	Display values of bundle manifest headers.
`r[un]` *filename* `|` *url*	Execute commands read from the specified filename or URL.
`se[t]` *property_name*=*property_value*	Set the value for the specified property.
`ser[rvices]` [*filter*]	List all registered services whose properties match the given *filter* or all services if no filter is specified. The filter is the string representation of an LDAP search filter as defined in *RFC 1960: A String Representation of LDAP Search Filters*.
`sh[utdown]`	Shutdown the framework and exit.
`sta[rt]` *bundle_url*|*bundle_id* `[, ...]`	Install and start the specified bundles.

TABLE 1-3 Framework Commands

Command Name	Description
`sto[p]` *bundle_url*`|`*bundle_id* `[, ...]`	Stop the specified bundles.
`tty`	Read commands from stdin. Use tty when you are starting the framework and you want to continue to work interactively.
`un[install]` *bundle_url* `|` *bundle_id* `[, ...]`	Uninstall the specified bundles.
`up[date]` *bundle_url* `|` *bundle_id* `[`*bundle_update_url*`] [, ...]`	Update the specified bundle. *bundle_url* takes precedence over the bundle's Bundle-UpdateLocation manifest header. The variables *bundle_url* and *bundle_update_url* must specify the full URL of a bundle or a path relative to that specified by the system property `com.sun.jes.framework.bundles.baseurl`.

▼ Installing and Starting the Core Service Bundles

1. **From the framework prompt, install the log, servlet, and http service bundles.**

```
> install log.jar, servlet.jar, http.jar
```

Since by default, `runjes` sets the system property `com.sun.jes.framework.bundles.baseurl` to `file:`*install_dir*`/bundles`, where *install_dir* is the path to the JES installation directory, you can use short names (instead of fully specifying the pathname) when referring to system bundles. You can also leave off the `.jar` suffix, if you prefer.

2. **Get the IDs for the bundles you installed.**

```
> bundles
ID  STATE      LOCATION
--  ---------  -------------------------
1   INSTALLED  file:/home/mcm/jes2.0/bundles/log.jar
2   INSTALLED  file:/home/mcm/jes2.0/bundles/servlet.jar
3   INSTALLED  file:/home/mcm/jes2.0/bundles/http.jar
```

Note that the state of the bundles is INSTALLED.

3. **View the manifest for the HTTP bundle (optional).**

```
> manifest 3
Bundle-Vendor: Sun Microsystems, Inc.
Bundle-Version: 0.1
Bundle-Activator: com.sun.jes.impl.http.HttpActivator
Bundle-DocURL: http://java.sun.com/products/embeddedserver
Created-By: 1.2.2 (Sun Microsystems Inc.)
Bundle-Name: http
Manifest-Version: 1.0
Bundle-ContactAddress: jes-comments@sun.com
Export-Package:org.osgi.service.http;specification-version=1.0,
org.osgi.service.log; specification-version=1.0,
com.sun.jes.service.http,               com.sun.jes.service.ssl
Export-Service: org.osgi.service.http.HttpService
Bundle-Description: The HTTP Service
Import-Package: javax.servlet; specification-version=2.1.1,
javax.servlet.http; specification-version=2.1.1
```

Notice that the HTTP bundle imports the `javax.servlet` and `javax.servlet.http` packages. Though the HTTP bundle depends on the servlet bundle, you do not have to install or start the servlet bundle before the HTTP bundle. You can start bundles in groups (as demonstrated in the next step) and the framework will resolve dependencies automatically.

4. **Start the service bundles.**

```
> start 1,2,3
> bundles
ID  STATE      LOCATION
--  ---------  -------------------------
1   ACTIVE     file:/home/mcm/jes2.0/bundles/log.jar
2   ACTIVE     file:/home/mcm/jes2.0/bundles/servlet.jar
3   ACTIVE     file:/home/mcm/jes2.0/bundles/http.jar
```

Notice that the state of the bundles is now ACTIVE.

Examining Package Dependencies

- **To obtain a list of package dependencies, use the exportedpackages command.**

```
> exportedpackages
Package: javax.servlet.http (2.1.1)
  Exported by: 2 (file:/home/mcm/jes2.0/bundles/servlet.jar)
  Imported by: 3 (file:/home/mcm/jes2.0/bundles/http.jar)
Package: com.sun.jes.service.http (0.0.0)
  Exported by: 3 (file:/home/mcm/jes2.0/bundles/http.jar)
Package: org.osgi.service.log (1.0.0)
  Exported by: 1 (file:/home/mcm/jes2.0/bundles/log.jar)
  Imported by: 3 (file:/home/mcm/jes2.0/bundles/http.jar)
Package: com.sun.jes.service.ssl (0.0.0)
  Exported by: 3 (file:/home/mcm/jes2.0/bundles/http.jar)
Package: javax.servlet.jsp (2.1.1)
  Exported by: 2 (file:/home/mcm/jes2.0/bundles/servlet.jar)
Package: javax.servlet (2.1.1)
  Exported by: 2 (file:/home/mcm/jes2.0/bundles/servlet.jar)
  Imported by: 3 (file:/home/mcm/jes2.0/bundles/http.jar)
Package: org.osgi.service.http (1.0.0)
  Exported by: 3 (file:/home/mcm/jes2.0/bundles/http.jar)
```

Getting a List of Services

- **To obtain a list of all registered services, use the** `services` **command.**

```
> services
[com.sun.jes.service.http.auth.users.UserPasswordService]
        description=User and Password Management Service
[com.sun.jes.service.http.auth.basic.BasicSchemeHandler]
        description=The OSGI HTTP Basic Authentication Service
[org.osgi.service.http.HttpService]
        description=The OSGI HTTP Service
[com.sun.jes.service.http.HttpAdmin]
[org.osgi.service.log.LogService]
        description=The standard OSGi Log service
[org.osgi.service.log.LogReaderService]
        description=The standard OSGi LogReader service
```

- **To obtain a list of services that match specific properties, use the** `services` **command with the** *filter* **option. The example returns the services that do not have the string "OSGI" in their descriptions.**

```
> services (!(description=*OSGI*))
[com.sun.jes.service.http.auth.users.UserPasswordService]
        description=User and Password Management Service
[com.sun.jes.service.http.HttpAdmin]
```

Updating a Bundle

The `update` command replaces an existing bundle with a new one. The command requires the URL or ID of the bundle to update. You can also specify a URL for the bundle you want to use for the update. If you do not specify a URL, the framework uses the URL specified in the `Bundle-UpdateLocation` manifest header. If no Bundle-UpdateLocation is specified, the framework uses the location of the original bundle.

Stopping Bundles

The `stop` command stops a running bundle. The command requires the URL or ID of the bundle to stop. When you stop a bundle, the framework unregisters the bundle's services, releases any services used by the bundle, and sets the bundle's state to RESOLVED.

Uninstalling Bundles

The `uninstall` command stops a bundle (if it is running), releases any persistent resources the bundle was holding, and sets the bundle's status to UNINSTALLED. Exported packages remain, however, until the framework is shutdown.

Setting and Getting System Properties

You can use the `get` and `set` commands to view and modify system properties on the framework command line. Most system properties changes take effect immediately, but `com.sun.jes.framework.cachedir` does not take effect until the framework is restarted. For a complete list of JES system properties and how to use them, see Appendix A.

You can also use the `get` command to view the following OSGi framework environment properties.

TABLE 1-4 OSGi Framework Environment Properties

Property	Description
`org.osgi.framework.version`	The version of the framework
`org.osgi.framework.vendor`	The vendor of this framework implementation.
`org.osgi.framework.language`	The language being used. See ISO 639 for possible values.
`org.osgi.framework.os.name`	The name of the operating system of the hosting computer.
`org.osgi.framework.os.version`	The version number of the operating system of the hosting computer.
`org.osgi.framework.processor`	The name of the processor of the hosting computer.

Using the JES Managment Panel

The JES Management Panel (JESMP) provides a graphical interface to a framework instance. Once you have a framework and core services running (see “Launching the Framework” on page 5 and “Installing and Starting the Core Service Bundles” on page 9), you can install and start the bundles required for the JESMP.

▼ Launching the JESMP

1. **Install and start the JESMP bundle and the bundles it depends upon.**

```
> start httpauth, tcatjspcruntime, httpusers, jesmp
> bundles
ID  STATE      LOCATION
--  ---------  -------------------------
1   ACTIVE     file:/home/mcm/jes2.0/bundles/log.jar
2   ACTIVE     file:/home/mcm/jes2.0/bundles/servlet.jar
3   ACTIVE     file:/home/mcm/jes2.0/bundles/http.jar
4    ACTIVE     file:/home/mcm/jes2.0/bundles/httpauth.jar
5    ACTIVE     file:/home/mcm/jes2.0/bundles/tcatjspcruntime.jar
6    ACTIVE     file:/home/mcm/jes2.0/bundles/httpusers.jar
7    ACTIVE     file:/home/mcm/jes2.0/bundles/jesmp.jar
```

   Notice that you do not have to type the `.jar` extension and that the `start` command was used to both install and start the bundles.

2. **Open the JES Management Panel in your browser.**

   Open the JES Management Panel in your browser.

   In the address window of your browser, type `http://`*host*`:`*port*`/admin`, where *host* is the host name specified in the `com.sun.jes.service.http.hostname` property (the default is the host where JES is running, or any host if that host is multi-homed), and *port* is the port number specified in the `com.sun.jes.service.http.port` property (default is 8080). For example, if neither property has been set, the JESMP can be accessed at this URL: `http://`*localhost*`:8080/admin`.

   You are prompted for a user name and password. By default, they are both `admin`.

Managing and Monitoring the Bundle Life Cycle

You can install, start, update, stop, and uninstall bundles, view the list of services a bundle has registered or is using, view package import and export information, read the JES log, and manage user accounts from the JESMP.

Bundles

The left pane of the *Bundles* tab lists all installed bundles. Click on a bundle name to view information about its state (INSTALLED, ACTIVE, RESOLVED), location, dependencies, and manifest headers and to perform operations such as starting, stopping, and updating the bundle.

You can also download and install a new bundle by typing the URL of the bundle in the first text box after the bundle list, then clicking *Install*.

You can browse to and install a bundle in a local file system by clicking the *Browse* button next to the second text box after the bundle list, then clicking *Install*.

Services

The left pane of the *Services* tab lists all registered services. Click on a service name to view information about its source bundle, client bundles, registration properties, and configuration properties. You can modify the configuration properties of a service as well. For example, you can change the `logSize` and `severityThreshold` properties for the `LogService`.

Packages

The left pane of the *Packages* tab lists all the packages currently in use. Click on a package name to view information about its exporter and importers.

View Log

The *View Log* tab lists the log messages as specified by the log system properties. To modify the number and type of messages that appear in the log, change the configuration properties listed for the `LogService` on the *Services* tab.

User Management

You can add and remove users from the *User Management* tab.

Shortcuts to Starting Up

Along with these utilities there are three scripts in the *jes_install_dir*`/bin` directory, `starthttp`, `startjesmp`, and `startall`, that automatically install and start the bundles required for the HTTP service, the JES Management Panel, or all JES services, respectively.

Using runjes and the start* Scripts

You can use either of the three start scripts, `starthttp`, `startjesmp`, and `startall`, with the `runjes` utility or with the framework `run` command.

Launching the Framework and Starting the HTTP Service

1. **Launch an interactive framework session with** `runjes`**.**

```
% cd install_dir
% bin/runjes tty
Java Embedded Server 2.0

Copyright 1998, 1999 and 2000 Sun Microsystems, Inc. All rights
reserved.
Use is subject to license terms.

Type 'h[elp]' for a list of commands.

>
```

2. **Install and start the bundles required for the HTTP service.**

```
> run install_dir/bin/starthttp
> bundles
ID  STATE   LOCATION
--  ------  ----------------------------
1   ACTIVE  file:/home/mcm/jes2.0/bundles/servlet.jar
2   ACTIVE  file:/home/mcm/jes2.0/bundles/http.jar
```

Launching the Framework and Starting the JES Management Panel

1. **Launch a noninteractive framework session and install and start the bundles required for the JESMP.**

```
% cd install_dir
% bin/runjes bin/startjesmp
% com.sun.jes.impl.http.ResourceServlet: init
com.sun.jes.impl.http.auth.users.UserAdminServlet: init
com.sun.jes.impl.http.auth.users.AddUserServlet: init
com.sun.jes.impl.http.auth.users.RemoveUserServlet: init
com.sun.jes.impl.http.auth.users.ErrorServlet: init
com.sun.jes.impl.http.auth.users.ChangePasswordServlet: init
com.sun.jes.impl.http.ResourceServlet: init
com.sun.jes.impl.jesmp.ui.MainPanel: init
com.sun.jes.impl.jesmp.ui.LRPanes: init
com.sun.jes.impl.jesmp.ui.BundleList: init
com.sun.jes.impl.jesmp.ui.BundleDetail: init
com.sun.jes.impl.jesmp.ui.ServiceList: init
com.sun.jes.impl.jesmp.ui.ServiceDetail: init
com.sun.jes.impl.jesmp.ui.PackageList: init
com.sun.jes.impl.jesmp.ui.PackageDetail: init
com.sun.jes.impl.jesmp.ui.LogDetail: init
com.sun.jes.impl.jesmp.ui.TabBar: init
```

2. **Open the JES Management Panel in your browser.**

   In the address window of your browser, type `http://`*host*`:`*port*`/admin`, where *host* is the host name specified in the `com.sun.jes.service.http.hostname` property (the default is the host where JES is running, or any host if that host is multi-homed), and *port* is the port number specified in the `com.sun.jes.service.http.port` property (default is 8080). For example, if neither property has been set, the JESMP can be accessed at this URL: `http://`*localhost*`:8080/admin`.

   You are prompted for a user name and password. By default, they are both `admin`.

CHAPTER 2

Using the HTTP Service

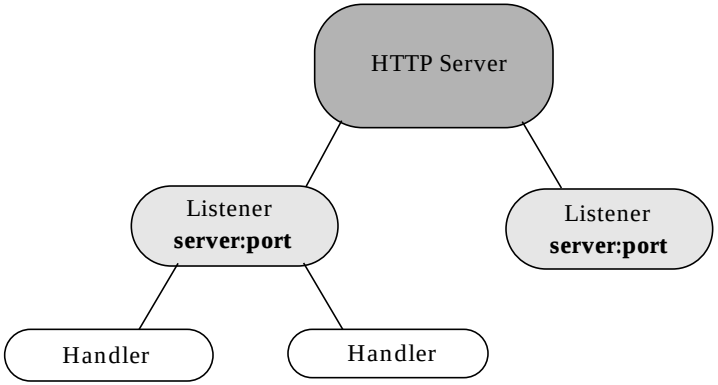
Overview of the JES HTTP Service

The Java Embedded Server™ framework includes a core HTTP service that allows you to write services that serve resources to the Internet. For example, you might be building a video camera service that would allow your customers to view certain locations in their home from a website.

The main job of the core HTTP service is to serve servlets and **resources** to the Internet. Servlets are Java™ classes based on the Java Servlet API, while resources can be HTML files, JSP files, or classes from bundles. The HTTP service maps resources to an URL **namespace**. A namespace is the part of the Internet's domain name system that the server controls, for example, all of the URL names that begin with http://myserver.com. This means that when a client requests an URL that falls within the server's namespace, the server delivers the corresponding resource using HTTP. The HTTP service can use either HTTP 1.1 or HTTP 1.0.

The HTTP service contains a Web server with **listeners** and **handlers**. Each listener corresponds to a port number for a Web server (for example, myserver.com:8080) and listens for HTTP requests. Handlers respond to the requests. The structure is that the HTTP server has a number of listeners and each listener has a number of handlers.

FIGURE 2-1 Conceptual Structure of the HTTP Server



The handlers are dynamically assigned to the listeners and the number of handlers varies according to the load the HTTP server is experiencing. You can configure the minimum and maximum number of handlers by setting system properties, as described in Appendix A.

To use the JES HTTP service, you need to write at least two components:

1. A **client service** that calls the HTTP service's API (contained in the com.sun.jes.service.http.HttpService interface, which extends org.osgi.service.http.HttpService).
2. An **HTTP client**, the object that actually requests a resource from the HTTP service, which is usually a Java servlet. You can write the Java servlet directly or generate it from a JSP file using the JES 2.0 JSP runtime tool (about which you can find more information in README-tcatjsp.txt in the JES 2.0 build).

In other words, you often need to write both a service and a servlet, or write a service and use the JSP runtime tool to create a servlet. In either case, you'll need access to the Java Servlet 2.1 API, available on the Internet at http://java.sun.com/products/servlet/2.1. In addition, the JES HTTP service supports both HTTP 1.0 and HTTP 1.1.

Registering Servlets and Resources

Because the JES HTTP service allows you to register Java servlets, it is aware of the servlet request-response model. Essentially, a servlet is an HTTP response to an HTTP request.

MIME stands for MultiPart Internet Mail Extensions and is the standard describing the different types of messages that can be sent across the Internet. This means that the HTTP service sends the servlet an request with a MIME body type, and the servlet sends back a response with a MIME body type. (For more information on MIME, you can look up the HTTP Request for Comments documents (RFCs) 1521 and 1522 on the Internet.)

But the servlet model is essentially that simple: the server sends a request, and the servlet sends back a response. A servlet that works with the core HTTP service uses HTTP as its protocol. This means that it typically extends HttpServlet (from the javax.servlet.http package) and uses an HttpServletRequest object as its request and an HttpServletResponse object as its response.

The JES HTTP service requires that you register both servlets and resources (which are the HTML files, JSP files, images, or Java classes that the servlet might use to deliver content to the client). Both servlets and resources must be registered with an `HttpContext` object, which maps a resource name to an URL. This means that when you submit an URL to the HTTP service, it locates the corresponding resource and responds.

What Registering Servlets Does

Servlets written according to the Java Servlet API allow your service to deliver dynamic content to the World Wide Web. Specifically, servlets do this by sending `out.println` statements that contain valid HTML or that call resources to the client Web browser. (You can also write JSP files to deliver dynamic content, rather than writing servlets directly. If you're interested, you should investigate the JES JSP runtime compiler with the Tomcat Web server; see the `README-tcatjsp.txt` file for more information.) Resources include images, HTML files, or any other type of object the servlet needs to deliver the content.

To make servlets available to the JES HTTP service, your bundle must register them. In general, your bundle registers a servlet when the HTTP service is registered, listening for an event to detect this. The JES HTTP service in turn uses a context object (created with `HttpContext`) to get information about the servlet, particularly the servlet's URL. The HTTP service then starts the servlet by calling its `service` method, causing the servlet to respond to the HTTP request. Registering a servlet gives the servlet access to part of the server's URI namespace, so that you can access the servlet on the server.

You can define an implementation of `HttpContext` yourself, or you can pass a `null` value instead of an `HttpContext` object, causing the JES HTTP service to use a default value for the context object. In either case, the context object defines such things as the MIME type the servlet uses for its response to the client, the URL at which the servlet is registered, and whether the HTTP service should service the request.

What Registering Resources Does

Your servlet typically uses resources such as images of HTML files. Registering resources makes them visible in the server's URI namespace, so that a bundle has access to them by a certain URL. Resources can be packaged into a bundle's JAR file, available on a filesystem before the bundle is installed, or created by the bundle.

Registering resources in an HTTP client service is similar to registering servlets, except that resources are typically registered as bundles. When you register a resource, you still create an HttpContext object (created from org.osgi.service.http.HttpContext or a class that implements it) to give the JES HTTP service information about the resource. You then use the registerResources method from HttpService to map a resource name to an alias name, so that the resource can be retrieved by its alias.

Because the resource is within a bundle, it gets a special bundle URL that looks like bundle:*//id/path*, for example, bundle:*//1224/images/foo.gif*. The *id* is the unique bundle ID assigned when the bundle is registered and available with the getBundleId method. The *path* is the path to the resource within the bundle or on the filesystem.

Handling Service Dependencies

When you register resources and servlets, you should do so in response to events. Events mark a change in a service's lifecycle. For example, an event fires when a service is registered or unregistered. Your service depends on the JES HTTP service, so you should register servlets and resources when the HTTP service is registered and unregister them when the HTTP service is unregistered.

The HTTP service may already be registered when you start your bundle activator class, or it may become registered while your bundle activator is running. Your bundle activator needs to handle both situations. Specifically, you need to register both within the start method, after checking that the HTTP service is already running, and also within the serviceChanged method in response to a REGISTERED event.

You then unregister your servlets and resources in response to an UNREGISTERING event (CODE EXAMPLE 2-1 makes this more clear). When you register and unregister in response to events, the JES framework uses **symbol resolution** to resolve service dependencies.

The servlet and resource registrations last as long as the JES HTTP service is registered with the framework. You should always unregister servlets explicitly with the unregister method, which calls the servlet's destroy method and stops the servlet. If you don't use unregister, the servlet's alias would be unregistered, but the servlet itself and any threads accessing it still exist.

Because other services may in turn depend on your service, never call the BundleActivator.stop method in your bundle activator class. As CODE EXAMPLE 2-1 shows, you should declare it but not implement it. You should only use BundleActivator.stop in the context of Bundle.stop, so that your bundle is stopped correctly.

The httpRegister Method

The main work of CODE EXAMPLE 2-1 is done by the httpRegister method, which is defined within the TestBundle class. httpRegister does the work of registering the resources and servlets when a REGISTERED event occurs.

The first thing that httpRegister does is create an HttpContext object. You must always create an HttpContext object, as the JES HTTP service uses the HttpContext object to get information about the servlet's registration. To create an HttpContext object, you must implement its methods, handleSecurity, getMimeType, and getResource, either in the BundleActivator class or in another class.

The implementation of these methods in CODE EXAMPLE 2-1 is very simple. The handleSecurity method returns true so that the JES HTTP service will service the request; the getMimeType method returns null to allow the JES HTTP service to determine the MIME type the servlet returns; and the getResource method returns an URL. For a more complete implementation of these methods, especially handleSecurity, see "Using Basic Authentication" later in this chapter.

The httpRegister method then registers resources and servlets using the registerResources and registerServlet methods. These methods map a resource to an URL. For example, the method call

```
hs.registerServlet( "/block", block, null, null );
```

registers the servlet object named block to the alias /block, so that is available from the URL http://*yourServer*:*yourPortNumber*/block. The complete rules of alias mapping that you use in the registerServlet and registerResources methods are summarized in TABLE 2-1.

TABLE 2-1 Specifying Aliases in JES 2.0

Alias	Definition
Default host name	Defined in **com.sun.jes.service.http.hostname**. Otherwise, accepts connections from any host. A JES extension.
Default port number	Defined in **com.sun.jes.service.http.port**. Otherwise, the default port number is **8080** for **http** and **443** for **https**. On UNIX systems, you need root privilege to bind to a port number below **1024**. A JES extension.
/*a*	Specifies the alias */a* after the default host name and port number.
/*a/b*	Specifies the alias */a/b* after the default host name and port number.
http://*host*:*port*/*alias*	Specifies a host name, port number, and servlet alias. A JES extension.
http://*host*/*alias*	Uses the default port number from **com.sun.jes.service.http.port**. Otherwise, uses **8080** as the default port number for **http** and **443** as the default port number for **https**. A JES extension.
http://*:*port*/*alias*	Registers the servlet, using any local host, the specified port number, and the specified alias. A JES extension.

▼ To Register Servlets and Resources

1. **Write a bundle activator class that implements** BundleActivator **and** ServiceListener**.**
2. **Implement the** BundleActivator.start **method.**
3. **Within the** start **method, register a service listener for the JES HTTP service, get a reference to the service, get the service itself, and register your servlets and resources, if the HTTP service has been started before the bundle activator class.**
4. **Implement the** serviceChanged **method, listening for** REGISTERED **events and registering servlets and resources if the HTTP service is registered after the bundle activator class is started.**
5. **Also in** serviceChanged**, listen for an** UNREGISTERING **event and unregister your resources if the JES HTTP service is unregistered.**
6. **Declare the** BundleActivator.stop **method, but do not call it within your bundle activator class. You should only call this method within** Bundle.stop**.**

▼ Classes and Methods

TABLE 2-2 Classes and Methods for Registering Servlets and Resources

Class, Interface, or Package	Methods
org.osgi.framework.BundleActivator	start, stop
org.osgi.service.http.HttpContext	getMimeType, getResource, handleSecurity
org.osgi.service.http.HttpService	registerServlet, registerResources, unregister
org.osgi.framework.BundleContext	getServiceReference, getService, addServiceListener
org.osgi.framework.ServiceListener	serviceChanged
org.osgi.framework.ServiceEvent	getType
java.net.URL	openConnection
java.net.URLConnection	getInputStream

CODE EXAMPLE 2-1 TestBundle.java (OSGi-compliant)

	package test; import java.net.*; import java.io.*; import javax.servlet.*; import javax.servlet.http.*; import org.osgi.framework.*; import org.osgi.service.http.*;
Implement BundleActivator and ServiceListener	public class TestBundle implements BundleActivator, ServiceListener { private HttpService service; private ServiceReference ref; private BundleContext bc ; private byte[] buffer = new byte[2048]; private ByteArrayOutputStream baos = new ByteArrayOutputStream(2048);
	public void start(BundleContext bc) throws BundleException {
Get the bundle context from the framework	this.bc = bc;
Add a service listener for the HTTP service	try { bc.addServiceListener(this, "(objectClass=org.osgi.service.http.HttpService)"); } catch (InvalidSyntaxException e) { throw new BundleException("Could not register HttpService " + "event listener", e); }
Get a reference to the HTTP service and the service itself	ref = bc.getServiceReference("org.osgi.service.http.HttpService"); if (ref != null) { service = (HttpService)bc.getService(ref);
Register here if the HTTP service is already registered Call httpRegister(), defined below	httpRegister(); } }
Leave the stop method empty	public void stop(BundleContext bc) throws BundleException { }

	public void serviceChanged(ServiceEvent event) {
Always make these lines synchronized	synchronized (this) { int etype = event.getType(); if (etype == ServiceEvent.REGISTERED) {
Check to see if an HTTP service is already registered	if (service == null) { ref = bc.getServiceReference("org.osgi.service.http.HttpService");
If so, get the original HTTP service registered	if (ref != null) { service = (HttpService)bc.getService(ref);
Then, register servlets and resources	httpRegister(); } }
When a service is unregistered, get its service reference	} else if (etype == ServiceEvent.UNREGISTERING) { ServiceReference sr = event.getServiceReference();
If it's the same as the one originally registered, unregister servlets and resources	if (sr.equals(ref)) { service.unregister("/alias"); bc.ungetService(ref); service = null; } } } }
Define httpRegister	private void httpRegister() {
Implement the 3 methods required by HttpContext	HttpContext hc = new HttpContext () { public boolean handleSecurity(HttpServletRequest req, HttpServletResponse res) { return true; } public String getMimeType(String name) { return null; }

Register resources and create an URL	``` public URL getResource(String path) { URL u = getClass().getResource(path); System.out.println("url = " + u); return u; } }; try { service.registerResources("/alias", "/resources", hc); URL url = new URL("http://laguna:8080/alias/foo.txt"); System.out.println("Opening URL connection ..."); URLConnection uc = url.openConnection(); InputStream is = uc.getInputStream(); System.out.println(new String(getBytes(is))); is.close(); } catch (Exception e) { e.printStackTrace(); } } private byte[] getBytes(InputStream is) throws IOException { int n; baos.reset(); while ((n = is.read(buffer, 0, buffer.length)) != -1) { baos.write(buffer, 0, n); } return baos.toByteArray(); } } ```

How To Write a Servlet

The Java Servlet API consists of two packages, javax.servlet and javax.servlet.http. The two most important parts of the Java Servlet API (in the context of the JES HTTP service) are the javax.servlet.Servlet interface and the javax.servlet.http.HttpServlet class. A servlet that you use with the JES HTTP service should always extend HttpServlet, which implements Servlet through its superclass, GenericServlet.

The Servlet interface declares the three most common servlet methods—init, service, and destroy. (Remember that the registerServlet method calls service and the unregister method calls destroy.) When you extend HttpServlet to write a servlet that receives requests and returns responses by way of HTTP, you should always override at least one of its doXXX methods.

HttpServlet has a default implementation of service that dispatches requests to doGet, doPost, doPut, and other methods, according to the HTTP command that the servlet has received. An HTTP command is exchanged between the client and the server at the beginning of the request. An HTTP GET command would look something like this:

```
GET /index.html HTTP/1.1
```

Because many requests are GET requests, a servlet you write for the JES HTTP service will typically extend HttpServlet, implement a doGet method, and return HTML to the client Web browser by a number of out.println statements.

CODE EXAMPLE 2-2, which shows the beginning of SnoopServlet.java, is a simple example of how to write an HttpServlet.

First, notice that SnoopServlet extends HttpServlet. This is standard practice for servlets that return HTML to a Web browser by HTTP. SnoopServlet also happens to implement org.osgi.service.http.HttpContext, because it defines methods that the JES HTTP service can call to get information about a servlet's registration.

After defining a default user name and password (admin for each) and a constructor, SnoopServlet starts with a doGet method. doGet is passed two objects: HttpServletRequest, which contains the information the user sent as a request, and HttpServletResponse, which allows the servlet to make a response.

CODE EXAMPLE 2-2 The Beginning of SnoopServlet.java (JES-compliant)

These lines are specific to the JES and OSGI APIs	package snoopbasic; import java.io.*; import java.util.*; import java.net.*; import javax.servlet.*; import javax.servlet.http.*; import org.osgi.framework.*; import org.osgi.service.http.*; import com.sun.jes.service.http.auth.basic.*; public class SnoopServlet extends HttpServlet implements HttpContext { private final String USER = "admin"; private final String PASS = "admin"; SnoopActivator bc; public SnoopServlet (SnoopActivator bundleContext) { bc = bundleContext; } public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException { PrintWriterout; res.setContentType("text/html"); out = res.getWriter (); out.println("<html>"); out.println("<head><title>Snoop Servlet</title></head>"); out.println("<body>"); out.println(" Request information: "); out.println(" "); out.println(" "); . . .

The doGet method does four things. First, it creates a PrintWriter object that will become the output stream for data sent back to the client. Next, it sets the response type to text/html, indicating that the servlet produces standard HTML as its output. Third, doGet uses the getWriter method of the HttpServletResponse object to send a response of type java.io.PrintWriter (remember that the response is encoded with MIME type text/html) back to the client.

Because the response object is created when the user's request is made, it points correctly to the client that made the request. Once all that is done, doGet uses out.println statements to send HTML tags that the client Web browser can interpret and display as a Web page. This pattern is typical of doGet methods.

To summarize, a servlet you write for the JES HTTP service usually does the following:

- Extends javax.servlet.http.HttpServlet
- Implements doGet or another doXXX method that receives requests from the servlet's default service method
- Sets the response type, usually to text/html
- Creates a PrintWriter object and sets it to the return value of the getWriter method to send the response
- Uses out.println statements to send HTML to the client

These points are all basic to HTTP servlets. In addition, SnoopServlet takes steps that are specific to the JES and OSGI APIs and that are explained in more detail in "Using Basic Authentication."

You may also want to check these excellent references for more information about writing Java servlets:

- *Java Servlet Programming*, Jason Hunter with William Crawford, O'Reilly & Associates, 1998
- *Java Enterprise in a Nutshell: A Desktop Quick Reference*, Servlets chapter, David Flanagan et al, O'Reilly & Associates, 1999
- Java Servlet Specification, Version 2.1, Sun Microsystems, November 1998, available at *http://java.sun.com/products/servlet*

Note that JES 2.0 supports the Java Servlet API 2.1 and does not include the concept of **sandboxed servlets** that have security restrictions on which files they can access.

Using Basic Authentication

HTTP 1.1 provides built-in support for **basic authentication**. Basic authentication, as describedin RFC 2617, is very simple and does not provide the highest level of security. However, all browsers support it, so it is used here as an example of how to implement the HttpContext.handleSecurity method.

Basic authentication is based on a challenge/response, username/password model. When you register servlets or resources, you can provide a special implementation of handleSecurity. When a client makes a request for some URL at which you have registered a servlet or resource, the server checks if the request has proper authentication information in its request headers. If not, the server provides a challenge to the client.

The challenge causes the Web browser to display a dialog box in which the user enters a name and password that are sent back to the server. If a user with the given name and password has been configured on the server, the HTTP service honors the request. With Java Embedded Server, basic authentication uses both the JES HttpService and the BasicSchemeHandler service.

Basic authentication has some inherent weaknesses. For instance, the user name and password are transmitted directly over the Internet in clear text. Anyone monitoring the network stream would have direct access to them. Basic authentication is simply used here as an example of how to implement handleSecurity.

The examples that follow show a servlet, SnoopServlet.java, that implements HttpContext and its three methods—getMimeType, getResource, and handleSecurity—and a bundle activator class, SnoopActivator.java, that uses an instance of SnoopServlet as its context object.

▼ To Use Basic Authentication

1. **Write a servlet that extends** HttpServlet **and implements** HttpContext**.**
2. **Specify a user name and password in the servlet.**
3. **Write a** doGet **method that checks the request headers for authorization information.**
4. **Write simple implementations of** HttpContext.getResource **and** HttpContext.getMimeType**.**
5. **Write an implementation of** handleSecurity **that uses a challenge/response model to request a user name and password.**
6. **Write a bundle activator class that implements** BundleActivator **and** ServiceListener**.**
7. **Get references to both the** HttpService **and** BasicSchemeHandler **service.**
8. **Use the references to obtain both the services.**
9. **Write a** serviceChanged **method that listens for events and calls a register method to register the servlet.**
10. **Write a register method that actually does the work of registering the servlet.**
11. **Declare a** stop **method, but leave its implementation empty.**

▼ Classes and Methods

Class, Interface, or Package	Methods
javax.servlet.http.HttpServlet	service, doGet
org.osgi.service.http.HttpContext	getMimeType, getResource, handleSecurity
org.osgi.framework.BundleActivator	start, stop
org.osgi.framework.ServiceListener	serviceChanged
org.osgi.framework.ServiceEvent	getServiceReference, getType
org.osgi.framework.BundleContext	addServiceListener
org.osgi.service.http.HttpService	registerServlet, registerResources, unregister
com.sun.jes.service.http.auth.basic.BasicSchemeHandler	getResponse, sendChallenge
com.sun.jes.service.http.auth.basic.BasicSchemeHandler.Response	getName, getPassword

CODE EXAMPLE 2-3 SnoopActivator.java (JES-compliant)

	package snoopbasic; import java.util.*; import java.net.*; import javax.servlet.*; import javax.servlet.http.*; import org.osgi.framework.*;
Import the http and http.auth. basic packages	import org.osgi.service.http.*; import com.sun.jes.service.http.auth.basic.*;
Implement BundleActivator and ServiceListener	public class SnoopActivator implements BundleActivator, ServiceListener { SnoopServlet snoop = null; HttpService http; BasicSchemeHandler basic; ServiceReference httpref; ServiceReference httpauthref;
Define the alias the servlet will use	final String SERVLET_ALIAS = "/snoopbasic"; BundleContext bundleContext;
Implement a start method	public void start(BundleContext bc) throws BundleException { bundleContext = bc; snoop = new SnoopServlet(this);
Get a reference to the HTTP service	httpref = bc.getServiceReference("org.osgi.service.http.HttpService"); if (httpref != null) { http = (HttpService) bc.getService(httpref); httpRegister(); }
Get a reference to the Basic Scheme Handler service	httpauthref = bc.getServiceReference("com.sun.jes.service.http.auth.basic.BasicSchemeHandler"); if (httpauthref != null) { basic = (BasicSchemeHandler) bc.getService(httpauthref); }

Add service listeners for both services

```
        try {
            bc.addServiceListener( this, "(objectClass=org.osgi.service.http.HttpService)" );
            bc.addServiceListener( this,
                "( objectClass=com.sun.jes.service.http.auth.basic.BasicSchemeHandler )" );
        }


        catch(InvalidSyntaxException ise) {
            ise.printStackTrace();
        }
    }
```

```
    public synchronized void serviceChanged(ServiceEvent event) {
```

Unregister servlets when the HTTP service is unregistered

```
        if ( event.getType() == ServiceEvent.UNREGISTERING ) {
            if ( event.getServiceReference().equals(httpref) ) {
                httpUnregister();
                httpref = null;
                http = null;
            }
            else if ( event.getServiceReference().equals(httpauthref) ) {
                httpauthref = null;
                basic = null;
            }
        }
```

Register servlets when the HTTP service is registered

```
        else if ( event.getType() == ServiceEvent.REGISTERED ) {

            httpref = bundleContext.getServiceReference( "org.osgi.service.http.HttpService" );
                if ( httpref != null && http == null ) {
                    http = (HttpService) bundleContext.getService(httpref);
                    httpRegister();
                }
            }
```

Get the Basic Scheme Handler service

```
            httpauthref = bundleContext.getServiceReference(
                "com.sun.jes.service.http.auth.basic.BasicSchemeHandler");
            if ( httpauthref != null && basic == null ) {
                basic = (BasicSchemeHandler) bundleContext.getService(httpauthref);
            }
        }
```

Define the registration method

```
    private void httpRegister() {
```

Register the servlet using the servlet as the context object

```
        try {
            http.registerServlet( SERVLET_ALIAS, snoop, null, snoop );
        }
```

	catch(ServletException se) { se.printStackTrace(); } catch(NamespaceException nse) { nse.printStackTrace(); } }
Define the unregister method	void httpUnregister() { try { http.unregister(SERVLET_ALIAS); } catch(IllegalArgumentException iae) { } } BasicSchemeHandler getBasicSchemeHandlerRef() { return basic; }
Leave the implementation of stop blank	public void stop(BundleContext bc) throws BundleException { } }

CODE EXAMPLE 2-4 SnoopServlet.java

	package snoopbasic; import java.io.*; import java.util.*; import java.net.*;
Implement the servlet packages	import javax.servlet.*; import javax.servlet.http.*;
Implement the http and http. auth.basic packages	import org.osgi.framework.*; import org.osgi.service.http.*; import com.sun.jes.service.http.auth.basic.*;
Extend HttpServlet and implement HttpContext	public class SnoopServlet extends HttpServlet implements HttpContext {
Define a user name and password	private final String USER = "admin"; private final String PASS = "admin"; SnoopActivator bc; public SnoopServlet(SnoopActivator bundleContext) { bc = bundleContext; }
Implement a doGet method	public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException { PrintWriterout; res.setContentType("text/html"); out = res.getWriter (); out.println("<html>"); out.println("<head><title>Snoop Servlet</title></head>"); out.println("<body>"); out.println(" Request information: "); out.println(" "); out.println(" ");

Get info from the request headers sent by the client

```
        print(out, "Request method", req.getMethod());
        print(out, "Request URI", req.getRequestURI());
        print(out, "Request protocol", req.getProtocol());
        print(out, "Servlet path", req.getServletPath());
        print(out, "Path info", req.getPathInfo());


      print(out, "Path translated", req.getPathTranslated());
      print(out, "Query string", req.getQueryString());
      print(out, "Content length", req.getContentLength());
      print(out, "Content type", req.getContentType());
      print(out, "Server name", req.getServerName());


      print(out, "Server port", req.getServerPort());
      print(out, "Remote user", req.getRemoteUser());
      print(out, "Remote address", req.getRemoteAddr());
      print(out, "Remote host", req.getRemoteHost());
      print(out, "Authorization scheme", req.getAuthType());


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

      out.flush();
   }
```

Print the user's name

```
private void print (PrintWriter out, String name, String value) {
    out.print(" " + name + ": ");
    out.println(value == null ? "&lt;none&gt;" : value);
    out.println("<br>");
}


private void print (PrintWriter out, String name, int value) {
    out.print(" " + name + ": ");
    if (value == -1) {
        out.println("&lt;none&gt;");
    } else {
        out.println(value);
    }
}
```

Implement the three methods required by HttpContext

```
public URL getResource( String str ) {
    return null;
}

 public String getMimeType(String str) {
    return null;
}
```

Implement handleSecurity	public boolean handleSecurity(HttpServletRequest req, HttpServletResponse res) {
handleSecurity is called for each request to the servlet	BasicSchemeHandler basic = bc.getBasicSchemeHandlerRef(); BasicSchemeHandler.Response response = basic.getResponse(req);
Display a dialog box for the user to log in Return false to display the user name and dialog box	if (response == null) { try { basic.sendChallenge(res, "dummy"); } catch(IOException ioe) { ioe.printStackTrace(); } return false; }
Get the user name and password	String user = response.getName(); String password = response.getPassword();
If the user name and password don't pass the check, display the dialog box	if (! check(user, password)) { try { basic.sendChallenge(res, "dummy"); } catch(IOException ioe) { ioe.printStackTrace(); } return false; } return true; }
Define the check method	boolean check(String user, String pass) { if (USER.equals(user) && PASS.equals(pass)) { return true; } return false; } }

Using the HttpAdmin Service

Once you register your servlets and resources, you can get information about them by using the HttpAdmin service. HttpAdmin works with HttpService. The two services are always registered together, so HttpAdmin is always registered when HttpService is.

HttpAdmin exposes two methods—getResourceRegistrations and getServletRegistrations, each of which return an array of HttpRegistration objects. An HttpRegistration object has five basic parts: the alias name, a bundle object, an HttpContext object, a resource name or servlet object, and an URL. You extract these parts with the methods the HttpRegistration interface contains:

```
public java.lang.String getAlias()
public javax.servlet.Servlet getServlet()
public java.lang.String getResourceName()
public java.net.URL getURL(java.lang.String defaultHost) throws java.net.MalformedURLException
public HttpContext getHttpContext()
public Bundle getBundle()
```

CODE EXAMPLE 2-5 shows how to use an HttpRegistration object. The example is the Java servlet that displays the Home Portal that is shipped with the Java Embedded Server. This example happens to be a servlet, but you can use the HttpRegistration object outside of a servlet as well. The servlet also implements basic authentication, which you learned about in the previous section.

Remember that you must have the HttpService and HttpAdmin services running in order to run this example. In this release, the two services always start together.

▼ To Use the HttpRegistration Object in a Servlet

1. **Write a class that extends** HttpServlet **and implements** HttpContext**.**
2. **Get references to the** BasicSchemeHandler **and** UserPasswordService **services, then get the services themselves.**
3. **Write a** doGet **method to return data to the client. Be sure to set the content type and get a** PrintWriter **object.**
4. **Get a reference to the** HttpAdmin **service, then get the service itself.**
5. **Get the array of servlet registrations.**
6. **Move through the array, taking action on each servlet. Remember that you can use any of the methods in** HttpRegistration**.**
7. **Write a** doPost **method, if your servlet is likely to receive** POST **requests.**
8. **Implement** HttpContext **and its** getResource**,** getMimeType**, and** handleSecurity **methods. If you like, you can use basic authentication, as described in the previous section.**

▼ Classes and Methods

TABLE 2-3 Classes and Methods for Using an HttpRegistration Object

Class, Interface, or Package	Methods
javax.servlet.http.HttpServlet	doGet, doPost
org.osgi.service.httpHttpContext	getResource, getMimeType, handleSecurity
org.osgi.framework.BundleContext	getService, getServiceReference
com.sun.jes.service.http.HttpAdmin	getResourceRegistrations, getServletRegistrations
com.sun.jes.service.http.HttpRegistration	getAlias, getBundle, getHttpContext, getResourceName, getServlet, getURL
javax.servlet.ServletResponse	setContentType, getWriter
javax.servlet.ServletConfig	getInitParameter
javax.servlet.Servlet	getServletConfig

CODE EXAMPLE 2-5 HomePortalServlet.java (JES-compliant)

	package com.sun.jes.impl.homeportal; import org.osgi.framework.*; import org.osgi.service.http.*; import com.sun.jes.service.http.HttpRegistration; import com.sun.jes.service.http.HttpAdmin;
Import both HttpRegistration and HttpAdmin	
Import both the authorization packages	import com.sun.jes.service.http.auth.basic.*; import com.sun.jes.service.http.auth.users.*; import java.io.*; import javax.servlet.*; import javax.servlet.http.*; import java.util.*; import java.net.URL; public class HomePortalServlet extends HttpServlet implements HttpContext {
Get the localized strings for displaying in different locales	private static final LocalizedStrings ls = (LocalizedStrings) java.util.ResourceBundle.getBundle("com.sun.jes.impl.homeportal.LocalizedStrings"); private HttpServletRequest myRequest; private HttpServletResponse myResponse; private PrintWriter out; private BundleContext bc; private BasicSchemeHandler basic; UserPasswordService ups; private boolean isAdmin = false; public HomePortalServlet (BundleContext bc, ServiceReference anHTTPBasicReference, ServiceReference anHTTPUsersReference) { super(); this.bc = bc; basic = (BasicSchemeHandler) bc.getService(anHTTPBasicReference); ups = (UserPasswordService) bc.getService(anHTTPUsersReference); }
When passed references, get the basic and user password authentication services	

Implement a doGet method	public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { this.myRequest = request; this.myResponse = response; this.myResponse.setContentType("text/html"); this.out = new java.io.PrintWriter(response.getWriter());
These four methods are all defined later	htmlBegin(); this.displayServices(); displayChangePassword(); htmlEnd(); }
Start the HTML output	void htmlBegin() { this.out.println("<HTML><HEAD><TITLE>"); this.out.println(ls.welcomeHomePortal()); this.out.println("</TITLE></HEAD>"); this.out.println("<BODY BACKGROUND='/images/homeportal/backrnd_tile.gif'>"); this.out.println("<img src='/images/homeportal/dotcom_title.gif'> "); }
	private void displayServices() { try {
Get HttpAdmin before you can get an HttpRegistration object	HttpAdmin httpService = (HttpAdmin)this.bc.getService(bc.getServiceReference("com.sun.jes.service.http.HttpAdmin")); HttpRegistration[] regs = httpService.getServletRegistrations();
Move through the servlets in the array	for (int i=0; i<regs.length; i++) { Servlet servlet = regs[i].getServlet(); ServletConfig servletConfig = servlet.getServletConfig();
Check for init parameters that make the servlet a home portal	String presentationStr = servletConfig.getInitParameter("com.sun.jes.service.homeportal.displayName"); String presentationImageAlias = servletConfig.getInitParameter("com.sun.jes.service.homeportal.displayImageURL");
If the servlet has a display name, get its URL	if (presentationStr != null) { URL aServletURL = regs[i].getURL("*");

If the servlet has an image, display it

```
        if ( presentationImageAlias != null ) { // present the icon too
              this.out.println( "<img src='" + presentationImageAlias + "'>" );
        }
```

Use a null target to ensure the same browser

```
           this.printLink( aServletURL.getFile(), presentationStr,null );
           this.out.print("<BR><BR>");
        }
```

If the servlet has an image, but no text ...

```
        else if ( presentationImageAlias != null ) {
```

Display the image linked to the servlet's URL

```
           URL aServletURL = regs[i].getURL("*");
           this.out.println("<a href='");
           this.out.println(aServletURL.getFile());
           this.out.print( "'>" );
           this.out.println( "<img src='" + presentationImageAlias +"'>" );
           this.out.print( "</A>" );
        }
     }
```

Check if the user is an administrator

```
        if(isAdmin == true) {
```

Then, display a link to the JES Management Panel

```
           this.out.print("<BR><BR>");
           this.out.print( "<img src='/images/homeportal/sel.gif' WIDTH='20' HEIGHT='22'
              ALIGN='BOTTOM' BORDER='0'>" );
           this.printLink("/admin", "  JES Management Panel", null);
           this.out.print("<BR><BR>");
        }
```

printLink is defined below

```
     } catch ( Exception e ) {
        e.printStackTrace();
     }

  }
```

Display the Change Password Web page

```
  void displayChangePassword() {

     this.out.print("<BR><BR>");
     this.out.print( "<img src='/images/homeportal/sel.gif' WIDTH='20' HEIGHT='22'
        ALIGN='BOTTOM' BORDER='0'>" );
     this.printLink("/chgp", "  Change Password", null);
     this.out.print("<BR><BR>");

  }
```

Complete the HTML output	void htmlEnd() { this.out.println("</BODY></HTML>"); this.out.close(); }
Implement a doPost method to handle POST requests	public void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException { this.doGet (req, res); }
Define printLink here	protected void printLink(String location, String label, String target) { this.out.print("<A HREF='"); this.out.print(location); if (target != null) { this.out.print("' TARGET='"+target); } this.out.print("'>"); this.out.print(""); this.out.print(label); this.out.print(""); this.out.print("</A>"); }
Implement the 3 HttpContext methods	public URL getResource(String str) { return null; } public String getMimeType(String str) { return null; }
Implement basic authentication	public boolean handleSecurity(HttpServletRequest req, HttpServletResponse res) { BasicSchemeHandler.Response response = basic.getResponse(req); if (response != null) {
Get the user name and password from the request headers	String user = response.getName(); String password = response.getPassword();

check is defined below	```if (check(user, password)) { if (ups.isAdmin(user)) isAdmin = true; return true; } } try { basic.sendChallenge(res, "homeportal"); } catch(IOException ioe) { ioe.printStackTrace(); }```
Returning false causes the browser to display the login dialog box	```return false; }```
check is defined here	```boolean check(String user, String pass) { boolean result = false; try { result = ups.checkPassword(user, pass); } catch(IllegalArgumentException iae) { iae.printStackTrace(); result = false; } return result; } }```

CHAPTER 3

Using the Log Service

Overview of the JES Log Service

When you start the JES log service, it runs along with the JES framework, in the background, waiting for event messages. The log service runs when you start the JES framework and then enter a command like this one:

> **start log**, servlet, http, tcatjspcruntime, httpauth, httpusers, jesmp

An event message can come from any type of event, including a BundleEvent, FrameworkEvent, or ServiceEvent, or from any bundle. The log resides in memory as long as the JES framework is running and can hold up to 1,000,000 entries. However, the log is not stored persistently and is destroyed when the framework is shut down.

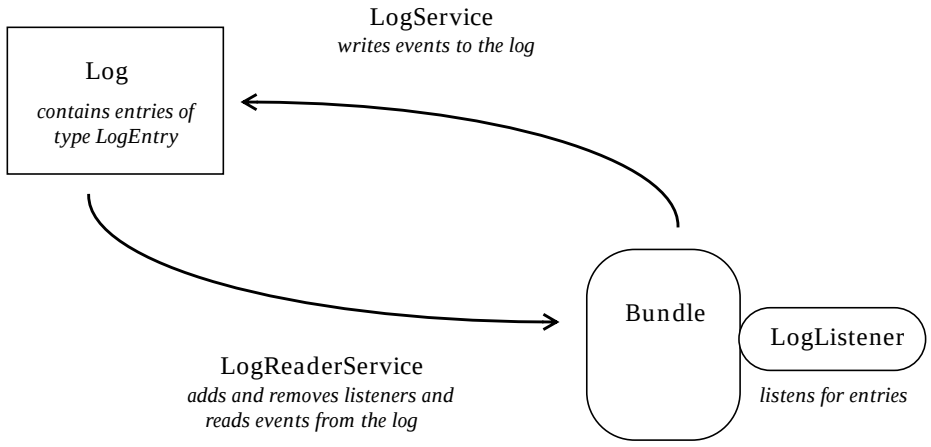
Most of the log service API is contained in the package org.osgi.service.log, which has four interfaces—LogEntry, LogListener, LogReaderService, and LogService.

You can think of the interfaces in the log service API this way:

- LogEntry creates an entry to add to the log.
- LogService puts the entry into the log.
- LogListener creates a listener so that a bundle is notified when an entry is added to the log.
- LogReaderService adds and removes listeners and reads messages stored in the log.

If you were to draw the log service, it might look something like FIGURE 3-1.

FIGURE 3-1 The Interfaces in the Log Service API



The JES framework also has a configurable interface with features that add to the OSGi specification; that is discussed later, in "Discovering the Configurable Interface."

Adding Entries to the Log

Each log entry is a a message that describes an event. A message always includes a log level and a String, and can also include a Throwable exception, a ServiceReference object, or both. You log messages using one of these forms of the log method:

LogService

public void **log**(int level, String message)
public void **log**(int level, String message, Throwable exception)
public void **log**(ServiceReference sr, int level, String message)
public void **log**(ServiceReference sr, int level, String message, Throwable exception)

Each entry contains a log level. The log levels are also in LogService and range from 1 to 4, with these descriptions:

Log Level	Description	Meaning
1	LOG_ERROR	An error message indicating that the bundle or service may not be functional. Highest level.
2	LOG_WARNING	A warning message indicating that the bundle or service is still functioning but may experience problems in the future as a result of the warning condition.
3	LOG_INFO	An informational message added as a result of some change in the bundle or service that does not indicate a problem.
4	LOG_DEBUG	A debugging message that is used to describe a problem and may be meaningless to anyone but the developer. Lowest level.

In this list, the highest (or most severe) log level is 1 (LOG_ERROR), and the lowest (or least severe) is 4 (LOG_DEBUG).

When you add a Throwable exception to a log entry, it's usually to an entry of type 1, LOG_ERROR. When you give an entry a ServiceReference object, the entry allows other bundles to discover information about the bundle that logged the message.

In addition to the log level, log string, Throwable exception, and ServiceReference object that you add, the log service also timestamps log entries. If all of this is sounding like a very long String for one entry, you're right. A log entry, which is a single String object, can contain multiple lines.

▼ To Log Entries

1. **Get the bundle context from the framework.**
2. **Get a reference to the log service, then the service itself.**
3. **Use the** log **method to put messages into the log.**

CODE EXAMPLE 3-1 demonstrates how to add messages to the log.

CODE EXAMPLE 3-1 Logging Messages (*LogExample.java*)

	package logexample;
Import the LogService interface	import org.osgi.framework.BundleContext; import org.osgi.framework.ServiceReference; import org.osgi.service.log.LogService;
	public class LogExample {
Get the bundle context from the framework	LogExample(BundleContext bc) { try {
Use the bundle context to create the log service reference	getServices(bc);
log is defined in getServices, below Check for a null in case getServices is called repeatedly	if (log == null) { System.err.println("Unable to get the Log service reference!"); return; }
Generate some fake log messages	log.log(LogService.LOG_ERROR, "Log error"); log.log(LogService.LOG_WARNING, "Log warning"); log.log(LogService.LOG_INFO, "Log info"); log.log(LogService.LOG_DEBUG, "Log msg");
Set log to null to avoid memory leaks	log = null; } catch (Exception e) { System.err.println(e.getMessage()); }
	}
getServices is defined here	public final static String LOG_CLASS = "org.osgi.service.log.LogService";

A reference to the log service object	private LogService log = null;
Synchronized to prevent the log service from being uninstalled while this bundle is trying to get it	synchronized void getServices(BundleContext bc) { ServiceReference sr = null;
Get the service reference and the log service	if (log == null) { if ((sr = bc.getServiceReference(LOG_CLASS)) != null) { log = (LogService) bc.getService(sr); } } } }

Getting Information from a Log Entry

Because each entry in the log is a LogEntry object, you can use the LogEntry methods to get information from the entry. You can extract the log level, log string, Throwable exception, ServiceReference object, or timestamp that are part of the entry. You can also retrieve the bundle that logged the message. The timestamp is returned in the number of milliseconds since January 1, 1970, 00:00:00 GMT.

The methods you use to retrieve information from a LogEntry object are these:

LogEntry

public Bundle **getBundle**()
public ServiceReference **getServiceReference**()
public int **getLevel**()
public String **getMessage**()
public Throwable **getException**()
public long **getTime**()

But you may also want to read the entries that were already added to the log before you created the listener. To do this, you enumerate through the entries and read them.

▼ To Enumerate and Read the Entries in the Log

1. **Get references to the log and log reader services.**
2. **Get the services themselves.**
3. **Use the** getLog **method from the log reader service to get the entries in the log as an enumeration.**
4. **Move through each entry in the enumeration.**
5. **Use the methods from LogEntry to read the entries.**

CODE EXAMPLE 3-2 shows you how to enumerate through the log.

CODE EXAMPLE 3-2 Enumerating Through the Log (*LogEnumeration.java*)

	package logenumeration; import org.osgi.framework.Bundle; import org.osgi.framework.BundleContext; import org.osgi.framework.ServiceReference;
Import the log classes	import org.osgi.service.log.LogEntry; import org.osgi.service.log.LogService; import org.osgi.service.log.LogReaderService; import java.util.Date; import java.util.Enumeration;
	public class LogEnumeration { private static final String API_PREFIX = "API Test "; LogEnumeration(BundleContext bc) { try {
Get references to the LogService and LogReaderService	getServices(bc);
Display a message if a service is null	if (log == null) { System.err.println("Unable to get Log service reference!"); return; } if (logreader == null) { System.err.println("Unable to get LogReader service reference!"); return; }
	// API test #1 System.out.println("public void log(int level, String msg)");
Log entries with just a level and a message	log.log(LogService.LOG_ERROR, API_PREFIX + "1: Log error"); log.log(LogService.LOG_WARNING, API_PREFIX + "1: Log warning"); log.log(LogService.LOG_INFO, API_PREFIX + "1: Log info"); log.log(LogService.LOG_DEBUG, API_PREFIX + "1: Log debug");

Log entries with a level, message, and Throwable exception

Log entries with a level, message, and service reference object

Log entries with a level, message, service reference object, and Throwable exception

Enumerate through the log and show each entry's contents

```
        // API test #2

        System.out.println( "public void log(int level, String msg, Throwable e)" );

        log.log( LogService.LOG_ERROR,   API_PREFIX + "2: Log error",
           new IllegalArgumentException("error") );
        log.log( LogService.LOG_WARNING, API_PREFIX + "2: Log warning",
           new IllegalArgumentException("warning") );
        log.log( LogService.LOG_INFO,    API_PREFIX + "2: Log info",
           new IllegalArgumentException("info") );
        log.log( LogService.LOG_DEBUG,   API_PREFIX + "2: Log debug",
           new IllegalArgumentException("debug") );


        // API test #3

        System.out.println( "public void log(ServiceReference sr, int level, String msg)" );

        log.log( sr, LogService.LOG_ERROR,   API_PREFIX + "3: Log error" );
        log.log( sr, LogService.LOG_WARNING, API_PREFIX + "3: Log warning" );
        log.log( sr, LogService.LOG_INFO,    API_PREFIX + "3: Log info" );
        log.log( sr, LogService.LOG_DEBUG,   API_PREFIX + "3: Log debug" );


        // API test #4

        System.out.println( "public void log(ServiceReference sr, int level, String msg,
           Throwable e)" );

        log.log( sr, LogService.LOG_ERROR,   API_PREFIX + "4: Log error",
           new IllegalArgumentException("error") );
        log.log( sr, LogService.LOG_WARNING, API_PREFIX + "4: Log warning",
           new IllegalArgumentException("warning") );
        log.log( sr, LogService.LOG_INFO,    API_PREFIX + "4: Log info",
           new IllegalArgumentException("info") );
        log.log( sr, LogService.LOG_DEBUG,   API_PREFIX + "4: Log debug",
           new IllegalArgumentException("debug") );


        System.out.println();

        for( Enumeration e = logreader.getLog(); e.hasMoreElements(); ) {
           showEntry ( (LogEntry) e.nextElement() );
        }

      } catch (Exception e) {
        System.err.println( e.getMessage() );
      }

    }
```

	private final static String LOG_CLASS = "org.osgi.service.log.LogService"; private final static String LOG_READER_CLASS = "org.osgi.service.log.LogReaderService"; private LogService log = null; private LogReaderService logreader = null; ServiceReference sr = null; synchronized void getServices(BundleContext bc) { if (log == null) {
Reference to the LogService object	if ((sr = bc.getServiceReference(LOG_CLASS)) != null) { log = (LogService) bc.getService(sr); }
Reference to the LogReaderService object	if ((sr = bc.getServiceReference(LOG_READER_CLASS)) != null) { logreader = (LogReaderService) bc.getService(sr); } } }
Show the contents of a log entry	private boolean showEntry(LogEntry entry) { Bundle b = entry.getBundle(); int level = (int) entry.getLevel();
Returns true if there's an error in the entry	long time = entry.getTime(); String message = entry.getMessage(); ServiceReference sr = entry.getServiceReference(); Throwable e = entry.getException();
Show only those entries that were logged by this bundle	if (message.startsWith(API_PREFIX) == true) {
Show the severity level information	System.out.print(" Level " + level + ": "); switch (level) { case LogService.LOG_ERROR: System.out.print("ERROR "); break;

	case LogService.LOG_WARNING: System.out.print("WARNING"); break; case LogService.LOG_INFO: System.out.print("INFO "); break; case LogService.LOG_DEBUG: System.out.print("DEBUG "); break; default: System.out.print(" ??? "); return true; }
Show the time that the message was logged	System.out.println(" at " + new Date(time).toString());
Show the message that was logged	System.out.println(" Message: " + message);
Show the service reference information and any exception messages	System.out.print("Service, Exception: " + sr); if (e == null) { System.out.println(", None."); } else { System.out.println(", "" + e.getMessage() + """); } System.out.println(); } // if return false; } }

Creating a Listener

To create a listener for your bundle, you implement both the LogListener and LogReaderService interfaces. LogListener creates the listener that allows your bundle to "hear" entries logged by the JES framework or by other bundles as soon as they have been logged. LogReaderService allows you to add a listener, remove it, or read the entries already in the log in the form of an Enumeration.

LogListener and LogReaderService include these methods:

LogListener

public void **logged**(LogEntry entry)

LogReaderService

public void **addLogListener**(LogListener listener)
public void **removeLogListener**(LogListener listener)
public Enumeration **getLog**()

▼ To Create a Listener

1. **Write a class that implements LogListener.**
2. **Use the bundle context to create a reference to the log and log reader services.**
3. **Use the references to get the services themselves.**
4. **Add the log listener to the bundle.**
5. **Make the bundle sleep briefly so that framework events can be heard first.**
6. **Log the messages.**
7. **Remove the log listener at the end of the class, so that it is removed when the bundle shuts down.**

CODE EXAMPLE 3-3 shows you how to create a log listener.

CODE EXAMPLE 3-3 Creating a Log Listener (*LogListening.java)*

	package loglistening; import org.osgi.framework.BundleContext; import org.osgi.framework.ServiceReference;
Import the log service classes	import org.osgi.service.log.LogEntry; import org.osgi.service.log.LogListener; import org.osgi.service.log.LogService; import org.osgi.service.log.LogReaderService;
Implement LogListener	public class LogListening implements LogListener { LogListening(BundleContext bc) { try {
Use the bundle context to create the log service reference	getServices(bc); if (log == null) { System.err.println("Unable to get Log service reference!"); return; } if (logreader == null) { System.err.println("Unable to get LogReader service reference!"); return; }
Set up a listener	logreader.addLogListener(this);
Make the bundle sleep briefly so framework events can be heard first	Thread.sleep(10);
Create some messages to be logged	System.out.println("Using: public void log(int level, String msg)"); log.log(LogService.LOG_ERROR, "Sample LOG_ERROR"); log.log(LogService.LOG_WARNING, "Sample LOG_WARNING"); log.log(LogService.LOG_INFO, "Sample LOG_INFO"); log.log(LogService.LOG_DEBUG, "Sample LOG_DEBUG");
Remove the listener so we can't detect messages any more	logreader.removeLogListener(this);
Messages that follow should not be logged	log.log(LogService.LOG_ERROR, "ERROR: You shouldn't see this");

Remove the log listener when the bundle shuts down

```
if (log == null) {
        if ((sr = bc.getServiceReference( LOG_CLASS )) != null) {
            log = (LogService) bc.getService( sr );
        }
        if ((sr = bc.getServiceReference( LOG_READER_CLASS )) != null) {
            logreader = (LogReaderService) bc.getService( sr );
        }
    }
  }
```

```
  private final static String LOG_CLASS = "org.osgi.service.log.LogService";
  private final static String LOG_READER_CLASS = "org.osgi.service.log.LogReaderService";
```

Reference to the LogService object

```
  private LogService log = null;
```

Reference to LogReaderService

```
  private LogReaderService logreader = null;
```

Here's getServices

```
  synchronized void getServices( BundleContext bc ) {
    ServiceReference sr = null;
```

Implement the logged method from LogListener

```
  public void logged( LogEntry entry ) {
    try {
      System.out.println( entry.getMessage() );
    } catch (Exception e) {
        System.err.println( e.getMessage() );
        return;
    }
  }
```

Gets messages as they are logged and displays them

```
}
```

Discovering the Configurable Interface

The JES log service includes methods you can use to configure the size and severity threshold of the log. Please note that these methods are not made publicly available in the JES API. You must discover them by using reflection with a configuration object (an object that implements org.osgi.framework.Configurable).

The methods that you use to configure the log are listed below:

```
public int getLogSize()
public synchronized setLogSize( int size )
public int getSeverityThreshold()
public void setSeverityThreshold( int threshold )
```

When you use reflection to discover these methods, the output of the reflection looks like this:

```
Method: getLogSize
Current Log size = 20 LogEntry elements.

Method: setLogSize
Size should be set to 50

Method: getLogSize
New Log size = 50 LogEntry elements.

Method: getSeverityThreshold
Current severity threshold = 4

Method: setSeverityThreshold
Severity threshold should be set to 2

Method: getSeverityThreshold
New severity threshold = 2
```

When you configure the log's size, remember that its minimum size of is 1, its maximum size is 1,000,000, and its default size is 20. If you set the size of the log less than the current size, only the most recently logged messages are kept. If you make the size of the log greater than the current size, the log service creates a new log, copies all previous messages to it, and adds space for the additional entries.

The severity threshold is the log level at or below which entries are added to the log. For example, if you set the severity level to 4, entries with a log level of 4 or lower (that is, all entries) are added to the log. The severity threshold can be between 1 and 4, unless you extend the LogService interface to add more levels.

When you set the severity threshold to a new value, future log entries are only placed in the log if their levels are at or below the severity threshold level. When the threshold level is 4 (LOG_DEBUG, the default), entries of all levels are added to the log.

The log size and severity threshold are stored in a properties file on the embedded server's filesystem (if one exists) after the JES framework is shut down.

You can find an example of how to use reflection to discover the configurable interface in your jes2.0/docs/examples/LogReflect.

Using the Log Properties

You can also set the log size and severity threshold when you start the JES framework. Start JES 2.0 from the command line, using a java command with the -D option, followed by a property name and value, for example:

```
java -Dcom.sun.jes.service.log.size=200000 -jar framework.jar
```

The properties that affect the log service and their values are listed below:

com.sun.jes.service.log.size	Minimum 1, maximum 1000000, default 20
com.sun.jes.service.log.threshold	1, 2, 3, or 4; default 4

CHAPTER 4

Device Discovery and Access

Overview

The Java Embedded Server provides support for automatically discovering and accessing devices. It contains a device manager designed to comply with the OSGi specification, as well as a sample implementation of the `DriverLocator` service. The device manager discovers new devices by listening for registration of `Device` services with the framework registry. Once a device is registered, the device manager uses the `DriverLocator` service to find and download the drivers necessary to access the various representations of the device. The JES implementation of the `DriverLocator` service works in conjunction with a web server where it is assumed that all drivers are present. Typically, device manufacturers and gateway operators provide the necessary driver and device services. A sample implementation has been provided to demonstrate how device discovery and access works.

How Device Discovery and Access Works

To demonstrate device discovery and access, we describe how a JES gateway might discover and access a Hewlet Packard Laserjet printer with a USB connector. For this example, we assume a gateway manufacturer has provided a `USBDriver` service with the JES gateway.

1. **The `USBDriver` service listens for any device that attaches to a USB port. When the printer is hooked up, it communicates some of its basic properties. The driver discovers the printer and registers a `USBDevice` service with those properties. The `USBDevice` service represents the printer as a generic USB device, with no methods specific to printers.**

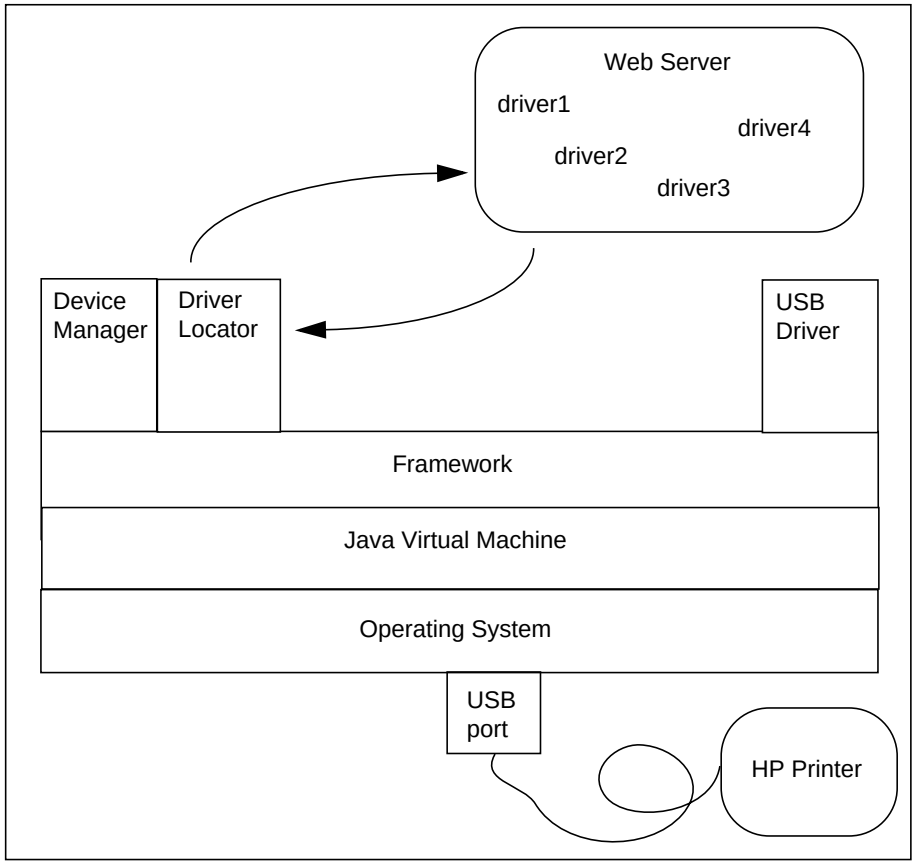
2. **The device manager listens for registration of any services that implement the** `Device` **interface. When the** `USBDevice` **service is registered, the device manager calls the** `findDrivers` **method on all registered** `DriverLocator` **services, passing the properties registered with the** `USBDevice` **service to the** `DriverLocator`**.**

3. **The** `DriverLocator` **service queries a web server for drivers that may be appropriate for the USB device. The** `findDrivers` **method returns zero or more** `DRIVER_ID` **values. The device manager tells the DriverLocator to download the bundles. The bundles are downloaded, installed, and started.**

4. **The device manager calls** `match` **on each newly downloaded driver and then calls** `attach` **on the driver that matches best, the** `HPPrinterDriver`**. The** `HPPrinterDriver.attach` **method creates a dependency between the driver and the** `USBDevice` **service. The chosen driver then registers a new** `HPPrinterDevice` **service with methods for accessing printer functionality.**

The process for finding, downloading, matching and attaching drivers and registering devices services (step 2, 3 and 4 in this sequence) continues until no more refined drivers can be found.

FIGURE 4-1 Detecting a Printer



APPENDIX **A**

JES 2.0 System Properties

JES System Properties

Java applications use system properties instead of environment variables to gather system information, because environment variables are platform dependent. The Java interpreter (the `java` command used to run an application) uses a standard list of system properties that you can find in most Java programming guides.

JES 2.0 also defines system properties that you can use when starting the JES framework. You can set system properties in several different ways:

- When you run the `runjes` script:
  ```
  runjes -j -Dcom.sun.jes.service.http.minHandlers=0
  ```
- At the JES framework command prompt:
  ```
  > set com.sun.jes.service.http.minHandlers=0
  ```
- When you start JES from the command line with the `java` command:
  ```
  java -Dcom.sun.jes.service.http.minHandlers=2
      -jar framework.jar
  ```

You can also get the current value of a system property from the JES framework command line, like this:

- `> get com.sun.jes.service.http.minHandlers`

The JES 2.0 system properties and their allowed values are listed in TABLE A-1.

TABLE A-1 System Properties You Can Set for JES 2.0

Service	Property Name	Description and Value
HTTP	**com.sun.jes.service.http.hostname**	The host name used in the URL of a registered servlet or resource. If you do not specify a value, the default is the host on which JES is running. If that host is multi-homed, the HTTP service accepts connections on any host.
	com.sun.jes.service.http.port	The port number used in the URL of a registered servlet or resource. Default values are **8080** for HTTP and **443** for HTTPS.
	com.sun.jes.service.http.minHandlers	An integer specifying the minimum number of handlers the HTTP service uses. Default value is **0.**

TABLE A-1 System Properties You Can Set for JES 2.0

	com.sun.jes.service.http.maxHandlers	An integer specifying the maximum number of handlers the HTTP service uses. Default value is **4**.
	com.sun.jes.service.http.use.log.service	A Boolean value that indicates whether messages are displayed on **stderr** or sent to the log. The only valid value is **true**. If the value is **true** and the log service is registered, the JES HTTP service directs exception messages to the log. If the value is **true** and the log service is not registered, messages are displayed on **stderr**. By default, the value is not set and messages are displayed on **stderr**.
SSL	**com.sun.jes.service.ssl.isSupported**	A Boolean value that determines whether your JES installation enables Secure Sockets Layer (SSL) connections. If the value is **false**, the SSL service starts but doesn't do anything. Default value is **true**.
	com.sun.jes.impl.keystore.store	An URL indicating where to get keystore contents for the SSL service's keystore. When you start JES, the **runjes** script sets this property to the example keystore file **lib/tlscerts**. If the value is empty or incorrect, the SSL service fails. No default value.
	com.sun.jes.impl.keystore.access	An encoded password for the example keystore **lib/tlscerts**. If the value is empty or incorrect, the SSL service fails. To set the password, you must use the JDK **keytool** to change it in the keystore, then the JES **bin/encodepw** script to encode it and add it to this property. No default value.
User	**com.sun.jes.service.http.auth.users.defaultAdminName**	The login name for the default administrator of the User Management bundle. If you set the administrator name without setting the password (below), the password is the same as the administrator name. This property is only used once, when the **httpusers** bundle is started, and it cannot be used by other bundles. Default value is **admin**.

TABLE A-1 System Properties You Can Set for JES 2.0

	com.sun.jes.service.http.auth.users.defaultAdminPassword	The password for the default administrator of the User Management bundle. This property is only used once, when the **httpusers** bundle is started, and it cannot be used by other bundles. Default value is **admin**.
	com.sun.jes.service.http.auth.users.defaultUserName	The login name for the default user of the User Management bundle. If you set the user name without setting a password (below), the password has the same value as the name. This property is only used once, when the **httpusers** bundle is started, and it cannot be used by other bundles. Default value is **guest**.
	com.sun.jes.service.http.auth.users.defaultUserPassword	The password for the default user of the User Management bundle. This property is only used once, when the **httpusers** bundle is started, and it cannot be accessed by other bundles. Default value is **guest**.
Log	**com.sun.jes.service.log.size**	An integer specifying the number of entries in the JES log. Minimum value is **1**, maximum is **1000000**, default is **20**.
	com.sun.jes.service.log.threshold	The level at or below which entries are added to the log. The allowed values are: • **1** for an error message • **2** for a warning message • **3** for an informational message • **4** for a debugging message. Default value is **4**, meaning that entries with a log level of 4 or below are logged.
Device	**com.sun.jes.device.debug**	A value that activates the debugging mode for device access and displays messages to output. To turn on debugging, use the value **on**. Any other value turns debugging off. No default value.

TABLE A-1 System Properties You Can Set for JES 2.0

	com.sun.jes.driverlocator.servletURL	The URL of the JES **FindDriverServlet** servlet on the Web server where device drivers are located. **FindDriverServlet** finds and loads device drivers from the Web server. No default value.
Framework	**com.sun.jes.framework.filesystem.available**	A Boolean value specifying whether a local filesystem is available for the JES framework to use. If set to false, the local disk cache and the **jescache** directory are disabled. Support for platforms not having a filesystem will be added in a future release. Default value is **true**.
	com.sun.jes.framework.cache.dir	The pathname to the cache directory, set in the Java system property **user.home**. Default value is the directory **jescache** in your home directory (for Solaris) or on your default drive (for Windows).
	com.sun.jes.framework.debug	A comma-separated list of the following options: • **help** - prints messages about debug options • **all** - turns on all debugging • **bcl** - turns on debugging for bundle classloading • **event** - turns on debugging for event handling • **fw** - turns on debugging for the framework • **filter** - LDAP filter parsing and evaluation Examples: **com.sun.jes.framework.debug=all** **com.sun.jes.framework.debug=bcl,fw**
	com.sun.jes.framework.bundles.baseurl	An URL of any type that points to the location of bundles. This property is ignored when the URL starts with **./** or **.**. Default value is **file:***/currentDirectory*.

TABLE A-1 System Properties You Can Set for JES 2.0

com.sun.jes.framework.exception.trace	If **true**, the exception stack trace (including that of a nested **BundleException**) is displayed. If **false**, just the exception name and message (including those of a nested **BundleException**) are displayed. Default value is **true**.
com.sun.jes.framework.max.open.jars	An integer specifying the maximum number of bundle JAR files that can be kept open in memory. Default value is **4.**
com.sun.jes.framework.events.synchronous	A flag indicating whether events are delivered synchronously. If **true**, all events are delivered synchronously; if **false**, only **ServiceEvent** events are. Default value is **false**.